# MODELING WEB HANDLING SYSTEMS AND A METHOD FOR DETERMINING FEEDBACK CONTROLLER GAINS

By

BENJAMIN DAVID REISH

Bachelor of Science in Mechanical Engineering
Oklahoma Christian University
Edmond, Oklahoma
2004

Master of Science in Mechanical and Aerospace Engineering
Oklahoma State University
Stillwater, Oklahoma
2015

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
DOCTOR OF PHILOSOPHY
July, 2020

MODELING WEB HANDLING SYSTEMS AND A METHOD

FOR DETERMINING FEEDBACK

CONTROLLER GAINS

Dissertation Approved:

Karl N. Reid

Dissertation Adviser

Jamey Jacob

Rushikesh Kamalpurkar

Lisa Mantini

Martin Hagan

# ACKNOWLEDGMENTS

I would like to thank the Web Handling Research Center (WHRC) at Oklahoma State University for supporting me through this venture. The resources and people that it made available to me were invaluable. Without the WHRC, I would not have graduated. I was saddened to see it close in December of 2019.

I would like to thank Tom Manning, Kelsey Chung, and Tom Estep from Dupont for their work testing samples of Tyvek for me. Our contact through the Web Handling Research Center made this collaboration possible. I am thankful that they provided me with the results of their time and experimental apparatus.

I would like to thank Dr. J. Keith Good for being my committee chair after Dr. Reid's retirement and until his own retirement. I am thankful that he pushed me to finish. Without his support and suggestions, I believe I would still be plodding along, searching for the proper time for a Qualifying exam. I am also thankful for his suggestion to extend my work to the High Speed Web Line because of its availability.

I would like to thank Dr. Karl Reid for being my advisor, chair, mentor, and friend during my time at Oklahoma State University as a doctoral student. I am very glad he came along side me and suggested (strongly) that web handling was a good place for me. His editorial work greatly improved my papers and helped me think in terms of what the reader would need to know.

I would like to thank my wife, Melissa, for her continued, patient support of my advanced degrees. She started me on the path to graduate degrees and has persisted in encouraging me to my goal. She has taken a lot of responsibility for my household and my family while I focused on research. Without her support I would neither have started nor finished.

Name: BENJAMIN DAVID REISH

Date of Degree: JULY, 2020

Title of Study: MODELING WEB HANDLING SYSTEMS AND A METHOD FOR DETERMINING FEEDBACK CONTROLLER GAINS

Major Field: MECHANICAL AND AEROSPACE ENGINEERING

Abstract: Modeling the longitudinal dynamics and control of web handling systems requires accurate models of the primitive elements involved, and an understanding of the processes used for speed and tension control. Primitive elements in a web line are the motors, the rolls of material, the idle rollers, and the web itself. Each primitive element has a simplified mathematical model at its heart. Modeling feedback control systems used for accurate control of speed and longitudinal tension in a web line require models of the web line. A systematic method for determining the controller gains for the speed and tension control systems in a web handling system is discussed along with the gain calculation method residing in the web line software, the Rockwell method. The Rockwell and the Routh Approximation methods are compared and found to have similar performance with Rockwell having better tension control while the Routh Approximation method has better speed control.

Results from experiments on the Euclid and High-Speed Web Lines are used to validate the models of the primitive elements. Simulations of experiments are used to validate the simulation tool. Simulation is used to show the effect on tension from a parameter study on span lengths and the effect on tension of changing a feedback device from a load cell to a dancer.

Slip is not distinctly a roller or a web problem, but a problem at the interface of the web and roller, which affects the assumptions used to derive primitive elements. Experiments show the presence of slip in certain circumstances and that different mechanisms of slip occur depending on operating conditions and certain parameters. Simulations of the Euclid Web Line show the effect of including the Ducotey-Good traction model. Experiments show the Ducotey-Good traction model is the appropriate mechanism of slip based on the speed of the roller at certain conditions, while at other conditions the Sliding-Friction Driven Roller model is the mechanism. The research finds a unified slip model is needed.

# TABLE OF CONTENTS

Chapter Page

Chapter Page

Chapter Page

Chapter Page

# LIST OF TABLES

Table Page

Table Page

# LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Figure Page

Figure Page

# NOMENCLATURE

| Symbol | | Description |
|---|---|---|
| $A$ | - | Cross-sectional area of the web (width $\times$ thickness) |
| $a_1$ | - | Bump magnitude for non-circular roll |
| $B_{fn}$ | - | Bearing friction |
| $Bn_{fn}$ | - | Nip roller bearing friction |
| $C_{pn}$ | - | Dancer torsional damping constant |
| $C_{mn}$ | - | Motor damping |
| $C_{xn}$ | - | Center of roller in x-direction |
| $C_{yn}$ | - | Center of roller in y-direction |
| $c_n$ | - | X-Y pair describing the location of the center of a roller |
| $D_n$ | - | Constant for S-wrap dancer calculations |
| $E$ | - | Young's Modulus |
| $e$ | - | Mathematical constant $\approx 2.718$ |
| $e_n$ | - | Offset magnitude of eccentricity in a roll or roller |
| $F_n$ | - | Normal force for calculating the friction force |
| $f_{qn}$ | - | Dancer input torque |
| $g$ | - | Acceleration of gravity $\approx 32.2$ ft/s$^2$ |
| $\mathsf{g}_{\mathsf{r}n}$ | - | Gear Ratio between motor and shaft in contact with the web (number of shaft rotations per motor rotation) |
| $h_n$ | - | Height difference between rollers used in pendulum dancer primitive element |
| $Ind_n$ | - | Slip indicator for roller $n$ which takes values of $-1, 0, 1$ |
| $J_{core}$ | - | Inertia of the paper tube the web is wound on, but plastic and metal are also used |
| $J_{mn}$ | - | Motor Inertia |
| $J_n$ | - | Combined inertia of the wound web, shaft, and the motor inertia reflected through the gear ratio |
| $J_{pn}$ | - | Dancer arm inertia |
| $J_{sn}$ | - | Inertia of the spindle shaft the web is wound on |
| $J_{web}$ | - | Inertia of the wound web |
| $Jn_n$ | - | Inertia of the nip roller |
| $K_c$ | - | Conversion factor from rotational speed to linear speed |
| $K_{dt}$ | - | Derivative gain for tension loop |
| $K_{is}$ | - | Integral gain for speed loop |
| $K_{it}$ | - | Integral gain for tension loop |
| $K_{mn}$ | - | Motor constant (torque per amp supplied current) |
| $K_{pn}$ | - | Dancer torsional spring constant |
| $K_{ps}$ | - | Proportional gain for speed loop |

| Symbol | | Description |
|---|---|---|
| $K_{pt}$ | - | Proportional gain for tension loop |
| $k_{fvt}$ | - | Final value of tension multiplier |
| $k_r$ | - | Real root multiplier in third-order characteristic equations where they are defined as a real root and a second-order dynamic system |
| $k_{TLsr}$ | - | Tension loop per speed ratio scaling factor |
| $L_{an}$ | - | Dancer arm length |
| $L_{arm}$ | - | Dancer arm length |
| $L_{en}$ | - | Distance from the rotational center to a roller center on an S-wrap dancer |
| $L_n$ | - | Free span length |
| $L_n^e$ | - | Effective length of the span when slipping occurs |
| $L_{nc}$ | - | Initial, unstretched span length used in pendulum dancer primitive element |
| $L_{qn}$ | - | Moment arm length of applied force to an S-wrap dancer |
| $L_{sn}$ | - | Horizontal length from the rotational center of an S-wrap dancer to where the span leaves a roller |
| $l_{cg}$ | - | Dancer center of gravity location from pivot |
| $M_d$ | - | The set of denominator polynomial coefficients produced by the Routh Approximation method |
| $M_n$ | - | The set of numerator polynomial coefficients produced by the Routh Approximation method |
| $m_p$ | - | Dancer pendulum mass |
| $n$ | - | Index number |
| $P_x$ | - | Center of rotation of S-Wrap dancer fixture in x-direction |
| $P_y$ | - | Center of rotation of S-Wrap dancer fixture in y-direction |
| $R_b$ | - | Build-up ratio |
| $R_{ec}$ | - | Empty core radius of the unwind roll |
| $R_n$ | - | Roller or roll radius |
| $Rn_n$ | - | Nip roller radius |
| $R_{pr}$ | - | Pull roll radius |
| $r_{1,0}$ | - | Roll radius without a bump in non-circular roll primitive element |
| $S(\theta)$ | - | Span length as a function of roll position |
| $s$ | - | Laplace operator |
| $T_{ld}$ | - | Lead time constant of a lead-lag filter |
| $T_{lg}$ | - | Lag time constant of a lead-lag filter |
| $T_n(s)$ | - | Web span tension in the Laplace domain |
| $\mathsf{T}_n$ | - | Torque in Laplace domain |
| $t$ | - | time, independent variable |
| $t_n$ | - | Tension in the web span |
| $t_n^a$ | - | Tension in the span when assuming full adhesion |
| $t_n^c$ | - | Tension in the region of contact between web and roller |
| $t_n^s$ | - | Tension in the span when slip occurs |
| $t_r$ | - | Rise time of a second-order dynamic system |
| $t_{web}$ | - | Web thickness |
| $U_n(s)$ | - | Input Current to the motor in the Laplace domain |

| Symbol | | Description |
|---|---|---|
| $u$ | - | Lead-lag filter input |
| $u_n$ | - | Input Current to the motor |
| $V_n(s)$ | - | Roller speeds in Laplace domain |
| $V_{n,0}$ | - | Roller steady-state speeds |
| $v_n$ | - | Roller speeds |
| $\mathsf{w}_{web}$ | - | Web width |
| $x$ | - | Lead-lag filter state |
| $y$ | - | Lead-lag filter output |
| $z$ | - | Discrete time domain operator |
| $\alpha$ | - | Mapping for non-circular roll shape definition |
| $\alpha_0$ $\alpha_1$ | - | Constants used in calculation of gains |
| $\beta$ | - | Angle from the entry of contact with the roller to the beginning of the exit region of slip |
| $\beta_0$ | - | Constant used in calculation of gains |
| $\Gamma(s)$ | - | Dancer position in Laplace domain |
| $\gamma$ | - | Dancer position |
| $\delta$ | - | Angle to the resultant force in calculating the normal force |
| $\varepsilon$ | - | Strain in the web |
| $\zeta$ | - | Damping ratio of a second-order dynamic system |
| $\theta_{in}$ | - | Angle from the right-hand x-axis to the point on a roller where the incoming span contacts the roller |
| $\theta_n$ | - | Angular position of roller $n$ |
| $\theta_{pr}$ | - | Angular position of the pull roll |
| $\theta_{on}$ | - | Angle from the right-hand x-axis to the point on a roller where the outgoing span exits the roller |
| $\theta_{wn}$ | - | Angle of wrap of web on a roller |
| $\mu_n$ | - | Coefficient of friction between web and roller |
| $\mu_T$ | - | Coefficient of traction between web and roller |
| $\xi$ | - | Angle from the entry of contact with the roller to the beginning of the region of adhesion |
| $\pi$ | - | Mathematical constant $\approx 3.14159$ |
| $\rho_A$ | - | Area density of the web in mass per unit area |
| $\rho_{web}$ | - | Density of web in mass per unit volume |
| $\phi$ | - | Ratio of web properties across a roller |
| $\tau_n$ | - | Torque input input to a motor |
| $\Omega(s)$ | - | Motor rotation rate in Laplace domain |
| $\omega_{ld}$ | - | Lead-Lag filter lead frequency |
| $\omega_{lg}$ | - | Lead-Lag filter lag frequency |
| $\omega_{pr}$ | - | Pull roll angular speed |
| $\omega_n$ | - | Motor rotation rate |
| $\omega_{n,ref}$ | - | Reference angular speed |
| $\dot{\omega}_{n,ref}$ | - | Reference angular acceleration |
| $\omega_{nat}$ | - | Natural frequency of a second-order dynamic system |

# CHAPTER I

# INTRODUCTION

Companies operate in a swiftly moving marketplace today, and a missed deadline means more market share for a competitor. In the web processing industry, costs associated with building a web processing line or retooling an existing line in terms of money, manpower, and margin must be calculated before beginning such a project. Companies navigate the stream by leveraging modeling and simulation to make decisions before hardware locks in certain options [1]. When a company must move in a radically new direction, sometimes the experience base is not there to build upon and at that point modeling and simulation become paramount.

Web handling systems or web lines are a manufacturing tool used to convert raw materials stored on a roll to either finished products, like newspapers, or parent rolls for another web handling process. Figure 1.1 shows a generic web line to help define terms used in this research. The storage rolls of material are called parent rolls. They can be quite large and one is shown in Figure 1.1 called the Unwind (green). The unwind is driven either directly or through a systems of gears and/or belts or chains by a motor. Being powered by a motor makes the roll a driven roller. The material being unwound from the parent roll or unwind is called web. Web is very long in one direction compared to either its width or thickness. The direction the web is traveling is changed by crossing over an idle roller or idler. One is indicated in Figure 1.1 with the web passing underneath it, but there are several idle rollers shown in the figure. They may also be used to simply support the web. A pendulum dancer is indicated in the figure which means it swings in an arc proscribed by its arm from a pivot. The end that contacts the web is a simple idle roller. The ability of that roller to translate as well as rotate makes the dancer able to adjust the tension in the web.

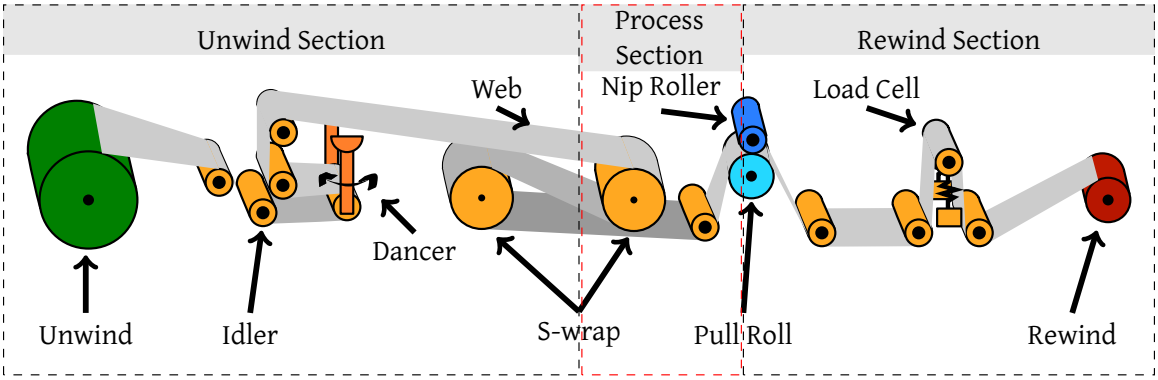



Figure 1.1: Generic Web Line. The Unwind (green) feeds web into the process. The dancer controls tension in the Unwind Section. The Process Section goes from the S-wrap to the Pull Roll (light blue). Tension is not controlled in the Process Section. The Pull Roll has a nip roller (dark blue) to enforce the no-slip condition at the Pull Roll. The Rewind Section tension is controlled by a load cell and the Rewind (red) winds web onto a roll for storage.

There is an external torque applied to the dancer arm which sets the tension in the web between the unwind and the S-wrap. The S-wrap is a pair of driven rollers which rotate at specific speeds and the web forms an 'S'-shaped path around them. The Unwind Section is the set of all rollers from the Unwind parent roll to the first S-wrap roller (called the lead) and is under tension set by the dancer. Following the S-wrap lead roller is the S-wrap follow roller which is also a driven roller. The web travels to the pull roll, as indicated in the figure which is a nipped roller, sometimes called a bridle. The nip presses the web onto the surface of the of the pull roll greatly increasing the friction and decreasing the chance of the web slipping over the pull roll. Slip is the event of the web moving at a different speed than the roller it is traversing while remaining in contact with the roller. The Process Section is the set of all rollers and spans between the S-wrap lead roller and the pull roll. The process section contains the value-adding process that converts the web from a raw material to the next step in its life-cycle. Then the web travels to the rewind roll (red) by passing over a load cell. The web is wound onto the rewind parent roll for storage or transport to another web handling process.

Commonly, the direction of web travel from the unwind toward the rewind is referred to as downstream or the machine direction. The direction of web travel from the rewind toward the unwind is called upstream. The direction across the width of the web is called the cross-machine direction. This research deals with the downstream or machine direction dynamics. Being that the web is long in that direction, the research focuses on modeling, simulating, and controlling

longitudinal dynamics of the web.

Controlling tension in a web means maintaining a set amount of force in the web itself. In Figure 1.1, the Unwind Section is a tension zone and has a pendulum dancer to set the tension. The dancer does so by exerting a force on the web wrapped around the dancer roller that is in opposition to the tension in the spans. The force is applied to the dancer arm by an air cylinder, hydraulics, or even simple gravity. In the Rewind Section, another tension zone in the figure, the tension is set by the load cell. The load cell does not apply a force to the web like the dancer does. It has to have a connection to the rewind motor. The load cell is a feedback device meaning it measures the tension in the web and sends a signal to the motor. The motor interprets the signal and reacts by speeding up if tension is too low or by slowing down if the tension is too high. The dancer in the Unwind Section is also connected, but to the unwind motor. The dancer measures its angular position and transmits that to the motor. The unwind motor interprets the signal and speeds up if the angular position is too far upstream or slows down if the angular position is too far downstream.

The interpretation of the signal sent from the dancer or load cell to the motor is control. The signal being sent is called feedback. Together, control and feedback form a feedback control which is more accurate than not having feedback. Not having feedback is called open-loop control which is like the valve on a hydrant or hose bib for a watering hose. Turning the valve half-way open allows some water through (controls), but not all the water that could go through the valve. There is nothing telling the valve to open more or less after it is initially opened. The valve has no feedback. Feedback in the case of a water valve would be like the float in a toilet valve. The float is a mechanical feedback device that tells the valve to close when there is enough water.

In many industrial applications, the interpreting is accomplished by a micro-computer or a programmable logic circuit. The default method used in these systems to interpret a feedback signal is a Proportional plus Integral (PI) controller. The PI controller initially subtracts the feedback signal from a set point (the value the user wants the controller to maintain) forming the error signal. The PI controller multiplies the error signal by a value, called a proportional gain, and adds that to the integral of the error signal multiplied by a value called the integral gain. The chal-

lenging part is determining the values of the proportional and integral gains because the values determine how the controller responds to the error signal. The process of selecting controller gains is called tuning. Another PI controller can be added in series with the first one and the pair of controllers together is called a cascaded control. Block diagrams of three different versions of a cascaded control are given in Figures 1.2, 1.3, and 1.4.

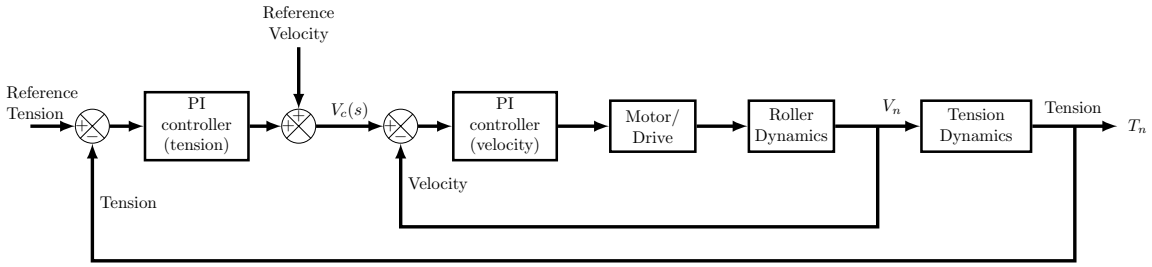



Figure 1.2: Speed based web tension control system using load cell feedback

## 1.1 Motivation

Feedback controllers are used in web lines to accurately control motor speeds and web tensions inside tension zones. A web line is divided up into several tension zones based on where driven rollers and feedback devices are positioned. The Unwind Section in Figure 1.1 is a tension zone because there is a driven roller, the unwind, controlled with a feedback device which is upstream of the next driven roller. The Process Section in the same figure is a different tension zone. The Rewind Section is a third tension zone. If the feedback controllers are not properly designed or tuned, and not independent, web tension may not be controlled adequately [2]. The results may include web breakage, web slackness, overstretching, wrinkling, printing errors, lamination curling, cracked coatings, loss of traction, sluggish operation, and web tension fluctuations and instabilities. Accuracy of tension control in a coating line is one example of a particularly serious case. Accurate tension control without tension fluctuations is often essential to evenness and quality of coating. Properly designed feedback systems may also be essential to running a line faster when profitability demands the use of thinner web materials and running different materials through the same line. Proper design of the feedback systems often starts with good modeling and simulation before "flying" the system. However, solutions to two major problems must be addressed:

(i) how to select the gains for each independent control system and (ii) what performance criteria should be used to guide the selection of gains. This research describes modeling and simulation of web handling systems and proposes a systematic method for selecting PI controller gains based on modeling the web line and a pairing with performance criteria that are easy to use.

In general, there are three types of controllers used in most web lines: speed control, tension control, and torque control. Torque control is outside the scope of this work. In the generic web line shown in Figure 1.1, the driven rolls such as the S-wrap rolls are under pure speed control, while the controllers for the unwind and rewind rolls use both speed control and tension control. The pull roll in Figure 1.1 is speed and tension controlled by a feedback device in the Process section that is not shown. Figure 1.2 shows a speed-based tension controller with load cell feedback that can be used at the rewind of the generic web line. An inner loop provides speed control of the rewind roll (or unwind roll), and an outer tension loop that provides a correction to the speed reference [3]. Figure 1.3 shows a speed-based tension controller with dancer position feedback instead of load cell feedback [3], that can be used at the rewind.

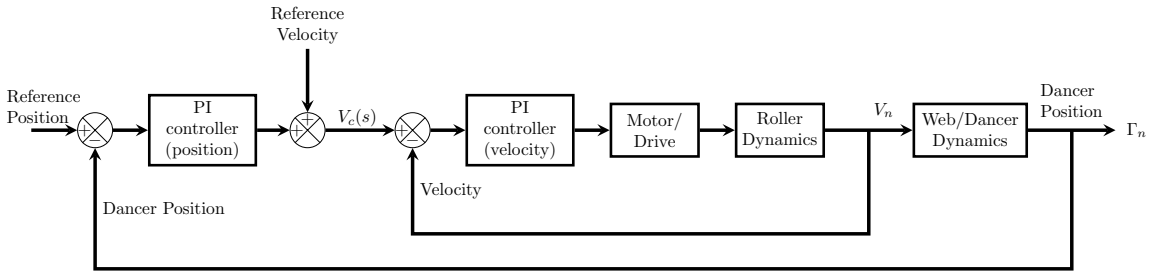



Figure 1.3: Speed based web tension control system using dancer position feedback

Both Figures 1.2 and 1.3 leave out some detail that may become important to a web line simulation. Figure 1.4 shows the control loop for the alternating current (AC) motors on the High-Speed Web Line (HSWL) which is slightly more involved than the previous two controller block diagrams. The two PI controllers only operate on the error signals and output a trim to the 'forced equilibrium' or 'equilibrium input' (so termed by Pagilla, Dwivedula, and Siraskar in [4–6]) calculated at the top of the figure. The forced equilibrium is found by solving the torque balance of a motor using a reference acceleration and speed for the input torque. The need for input torque is because the method of AC motor operation requires two set points: a speed set point and a torque

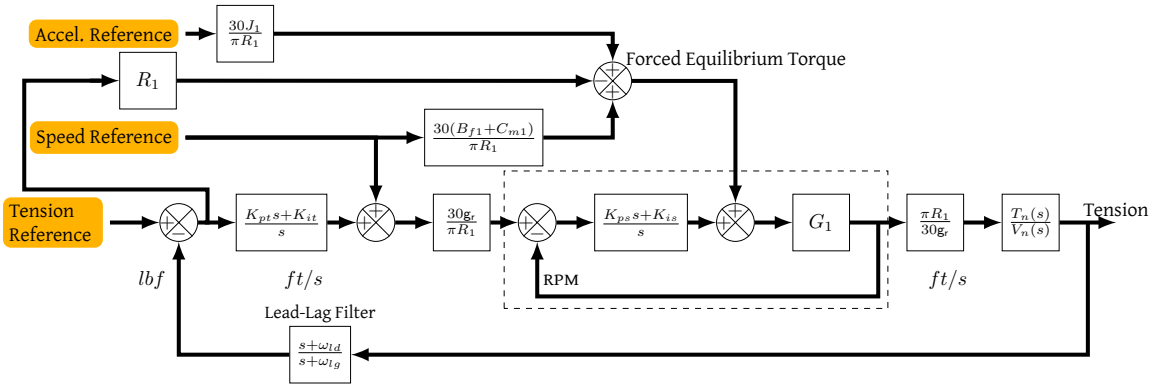



Figure 1.4: Rockwell Control Block Diagram for Use with AC Motors. The Tension Reference is input by the user along with the line speed, but the Accel. Reference and the Speed Reference are from an Industrial S-Curve using the line speed.

set point. As long as the desired pair is not faster than the motor's maximum safe operating speed and not above the motor's maximum torque capability, any combination of speed and torque are possible and the torque is delivered in microseconds. This is quite different than the control of a direct current (DC) motor where current is proportional to motor torque via a constant as in [7,8].

There is a distinction between modeling for control and modeling for dynamic simulation. A model for a controller should exhibit a smooth and seamless response to stimuli rather than react to erratic inputs. However, a model used for simulation of a physical system must be able to incorporate nonlinear and non-ideal effects to respond like the physical system would. Some of the reasons for dynamic simulation of a physical system include determining how the system responds to various stimuli and non-ideal effects, the sensitivity of the dynamic behavior of the system to parameter variations, finding appropriate control strategies, and discovering resonant frequencies to avoid in operation. Much of the literature in section 1.6 deals with control of physical systems, and not about modeling for simulation. In the case of simulating web lines, there is a lack of literature on developing models for physical systems that includes non-ideal elements and effects.

'Non-ideal' effects in a web processing line come in several forms. Non-circular parent rolls and eccentric rollers are often present in web lines. The effects of these two elements on the dynamic performance of a web line vary with the speed of the line. Dancers and eccentric rollers or non-circular rolls introduce non-ideal effects by changing the length of a span or spans that

attach to them. The length change affects the tension in those spans. A dancer may be one of two basic types: a translational (or linear) dancer where the gross movement of the roller is along a single line, and a pendulum dancer where the gross movement follows an arc described by the pendulum's arm and pivot point. In the case of a pendulum dancer, the length change of the spans attached introduces a nonlinearity into the model. The idea of a dancer can be applied to accumulators as well by increasing the number of rollers on a linear or translational dancer. The material of the web may not be of consistent density or volume across it's width and along the length of the web. Properties of the web can change with temperature and visco-elasticity can be introduced in places like dryers. The process that adds value to the web may add additional non-ideal effects: e.g., adding moisture, heating, cooling, coating, slitting, laminating, hole punching. This study does not include the process effects. The ideal situation would be to have perfectly circular rolls and spans that are constant length, density, and volume, but web lines are not ideal.

The remainder of the introduction will include a description of the web handling lines owned by the Web Handling Research Center (WHRC), a literature survey, and a discussion of the existing controller gain calculation methods on the WHRC web lines. Chapter 2 describes the primitive elements used to mathematically model web lines, both linear and nonlinear, with a section on the importance of nonlinear models. Chapter 3 details the systematic method for determining controller gains called the Routh Approximation Method for gain calculation. Chapter 4 describes experiments comparing the tension and speed responses of a web line using the new method to those using the existing (Rockwell) method and the modeling and simulation of web lines with comparison to experimental results. Chapter 5 describes Whitworth's slip model, the Sliding-Friction Driven Roller Model for web-roller slip, and the Ducotey-Good (DG) Traction model for web lines and experiments showing the presence of slip in a web line and Chapter 6 reports the conclusion and future work.

## 1.2 The Euclid Web Line (EWL)

The Euclid Web Line (EWL) was originally a research web line at Reliance Motors in Euclid, OH. The company was bought by Rockwell Automation and its research line was no longer needed.

The web line was given to the WHRC. It was capable of 500 FPM, forward or backward, at up to 50 lbf tension on 16 inch wide material which gave it a maximum of 2.7 PLI on 18 inch wide material. The unwind section and the S-Wrap are shown in Figure 1.6 and the schematic of the whole line is shown in Figure 1.5. The Euclid Web Line (EWL) had heavy, 3 inch diameter, steel idle rollers (shown in Figure 5.4 before and after polishing) and hollow steel drums for the S-Wrap rollers and Pull roll. Tension feedback was available for the unwind section by either load cell or pendulum dancer. Tension feedback via load cell was used in the rewind section. Lateral web position control was achieved by a Roll2Roll Technologies™ modified Fife™ web guide in the unwind section and a translating platform to which the rewind motor and spindle were mounted in the rewind section. The pendulum dancer, web guide, and load cell for the unwind section are pictured in Figure 1.6. The idlers on this line are 3 inch diameter, steel sleeved rollers.

The EWL had at one time been able to operate with either Direct Current (DC) motors or Alternating Current (AC) motors. Only the AC motors were operational during the research herein. The unwind motor and rewind motor were both 15 hp motors while the two S-Wrap motors and the Pull roll motor were all 5 hp motors. The unwind motor is located at R1 in Figure 1.5 and its roll diameter varies as web material is unwound from it. The S-Wrap motors are located at R10 and R11 and are controlled together in pure speed control. The pull roll motor was located at R16 and is a nipped roller, also known as a bridle, to ensure that the web does not slip while traversing the Pull Roll. The rewind motor is located at R25 and its roll diameter also varies based on how much web has been wound around it.

The EWL was divided up into three tension zones for control. The web from the unwind roll (R1) to the S-Wrap lead (R10) was called the Unwind Section. The web from the S-Wrap lead (R10) to the pull roll (R16) was termed the Process section. The web from the pull roll (R16) to the rewind roll (R25) was called the Rewind Section. The software interface was an Allen Bradley HMI and RSLogix 5000® software. One configuration of the EWL uses a speed-based tension control with load cell feedback at both the unwind and rewind. Another configuration uses a speed-based web tension control system with dancer position feedback at the unwind, and a speed-based system with load-cell feedback at the rewind.

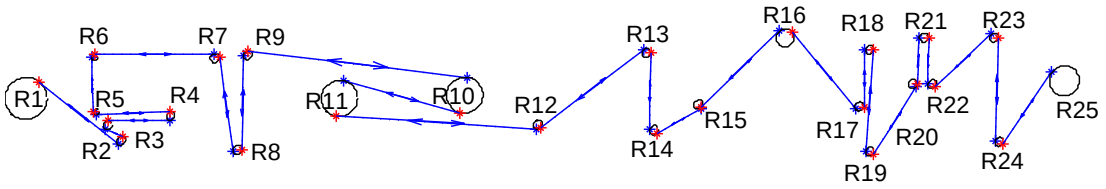



Figure 1.5: The Euclid Web Line schematic diagram with three sections from left to right: the unwind (R1 – R10), the process section (R10 – R16), and the Rewind section (R16 – R25). Rollers R3, R9, and R18 and are load cells which are used for measurement and/or control.

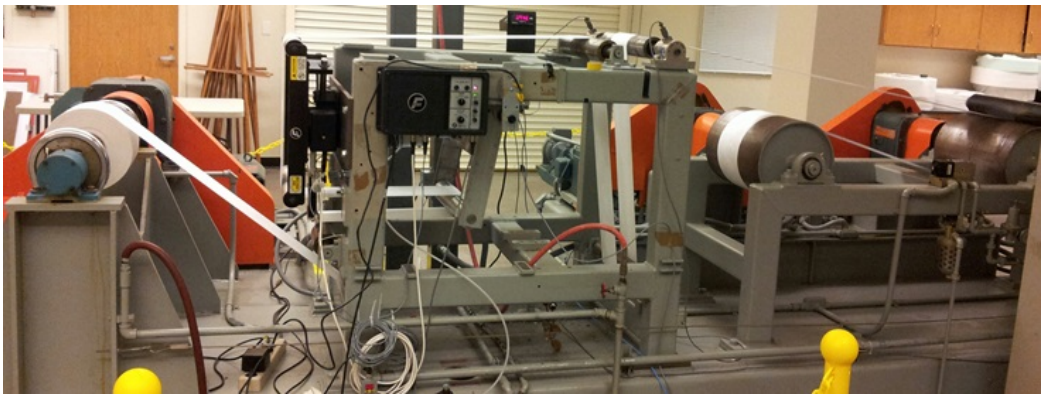


Figure 1.6: Unwind section of Euclid Web Line at Oklahoma State University. The unwind roll is at the left and the forward process direction is to the right, ending with the S-wrap (2 large drum rollers on the right).

## 1.3 The High-Speed, Low-Tension Web Line

Some web processing lines are expected to run 24 hours a day, 7 days a week which means a mechanism to change out expiring rolls with full ones has to be used. One method for this is to incorporate an accumulator and zero-speed splice operation into the web line. Other types of winders and unwinders can pivot a full roll into position and attach the new roll to the old by means of adhesive on the fly. The High-Speed, Low-Tension (HSLT) web line (schematic in Figure 1.7) is a research web line with a Martin Automatics™ accumulator to allow for zero-speed splice research as well as research in low-tension dynamics and research at higher speeds than are attainable with the EWL. Also it has a rewind that can be configured to run to as a center winder or a surface winder. It can run at line speeds up to 1700 FPM and tensions of 6 lbf. So, it can run at four times the speed of the Euclid line but with only a fraction of the tension. The idlers are smaller and made of carbon fiber or aluminum instead of steel. The inertia of these rollers is far less than that of the idlers for the Euclid line and there are two different types: one made by Martin Automatics™ that

came with the accumulator (aluminum) and one created by the WHRC for the process section and winder section of the line (carbon fiber). Also, unlike the EWL, the HSLT line can only run in one direction. Lateral control of the web is only applied in the process section of the HSLT by a web guide.

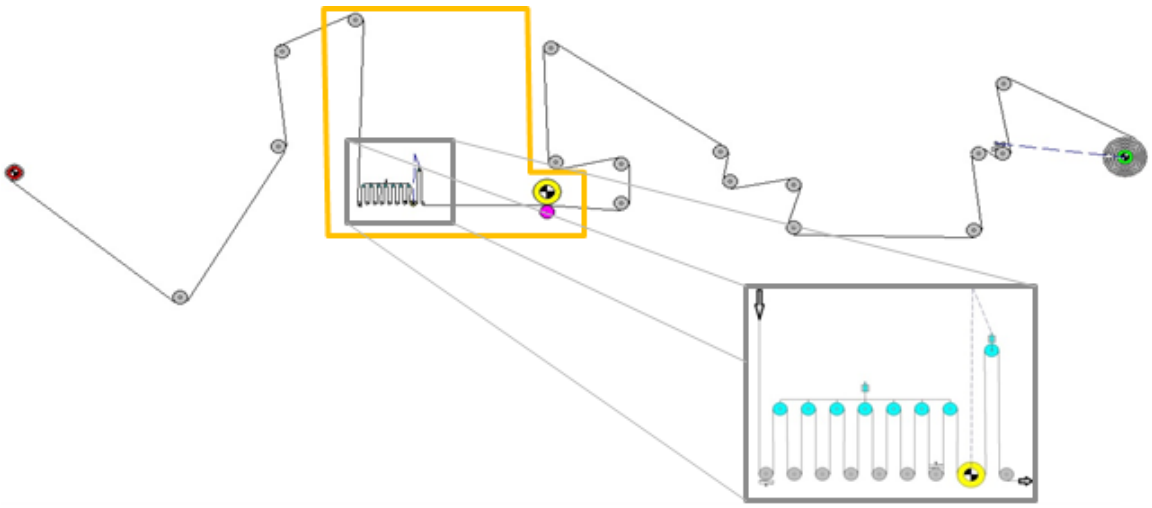


Figure 1.7: The High-Speed, Low-Tension Web Line Schematic. The accumulator and nipped pull roll are indicated in yellow. The accumulator is shown zoomed in in the gray box to show the presence of the driven roller right at the end of the accumulator, before the dancer.

The web line is controlled by a conglomeration of Martin Automatics™ code and Rockwell Automation™ code stitched together. The control software code has been studied and the mechanisms used for such things as feedback and tension control are only vaguely understood. The HSLT line can automatically execute a zero speed splice operation. The general process is laid out in [9–11] with emphasis usually on designing the (multi-faceted) control or making observations to help the accumulator designer make decisions. The zero speed splice process sequences through 5 steps: normal operation, filling the festoon, splicing, emptying the festoon, and returning to normal operation because accumulators are generally not operated in their full state (lots of equal length, long spans is a bad idea, see [12]). Good and Markum recorded tension data from a load cell placed in the accumulator (see Figure 1.8) and the data is shown in Figure 1.9 where the tension trace over all five steps of the splice can be seen (they indicate four steps: they group the splice and the emptying of the accumulator to one step). The tension is doubled during the splice at one point and drops below 1 lbf at another during the splice.

There are many non-ideal effects in a splice operation. Control methods change as the process steps through different stages of the splice. The rolls could be non-circular and are allowed to be bigger in diameter than on the EWL so the speed impacts of the roll shape would be bigger, but angular speed is lower due to the larger radius. The splice event requires the web to come to a full stop at the point of the splice while the web at the other end of the accumulator is continuing at line speed. Every span in the accumulator is changing length and speed. There is air drag on the web throughout the line, and there may be slip on one or more of the idle rollers in the accumulator. The festoon can rotate, spans can slip on their rollers as the festoon descends, and a state of slack has been observed in one or more of the accumulator spans on occasion. The resident control methods seem to attempt to not break the web and meet back at the ranch for dinner, but tension is only vaguely controlled. Line speed is maintained on the output side of the accumulator by a driven roll followed by a dancer which is immediately followed by a nipped driven roll in the process section. The load cell for post accumulator tension verification is at the far end of the process section next to the rewind so the tension levels just as the web leaves the accumulator are unknown. Pagilla, Singh, and Dwivedula proposed a method of control for accumulator tension in [13]. They studied a nonlinear control scheme and arrived at asymptotic convergence for their proposed control method, but only had simulations. They did not present experimental verification of their scheme working and their model was of an ALCOA processing plant with rather larger values than the one the WHRC has in the HSLT line.


Figure 1.8: Load Cell in Accumulator on HSLT Line

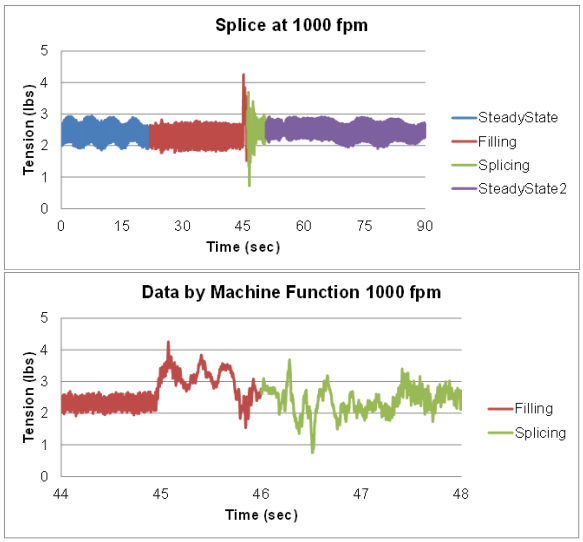



Figure 1.9: Load Cell Tension Over Whole Splice Operation (top) and Detailing the Filling and Splicing stages (bottom) at 1000 FPM from the HSLT line. Courtesy of Good and Markum.

## 1.4 The High Speed Web Line

The High-Speed Web Line (HSWL) is the third full scale web line owned by the WHRC and located in the basement of the Advanced Technology and Research Center in Stillwater, OK, on OSU campus. It is capable of 2500 FPM, with 30 inch wide material at tension levels up to 3 PLI. On 30 inch wide material that means 90 lbf of tension. Original research in web wrinkling was planned for that line and many web paths through the machine were possible. It could accept 60 inch diameter rolls of parent material which is important when the line speed is 2500 FPM.

A schematic of about half the High-Speed Web Line (HSWL) is shown in Figure 1.10. This is the 'bypass path' on the machine. There are two more modules to the right of the layout that is shown: the wrinkle module and the second nip stand that are not used in the 'bypass path' layout (shown in Figure G.1 of the Appendix). Between the wrinkle module and the second nip stand is the process section of the HSWL. There are six 15- or 30-hp Reliance RPM motors (bought out by Baldor®) which have high-precision encoders that allow for high speed operation and accurate speed tracking at low speeds. The unwind and rewind motors (rolls 1 and 29 in Figure 1.10) are 30 hp and the pull roll is 15 hp (roll 9) and it is a nipped roller. There is no gearing between the motors and the spindles on the HSWL. The unwind section has two options for tension feedback: a load cell (roller 3) or a linear dancer (roller 8). The rewind section has two options for tension

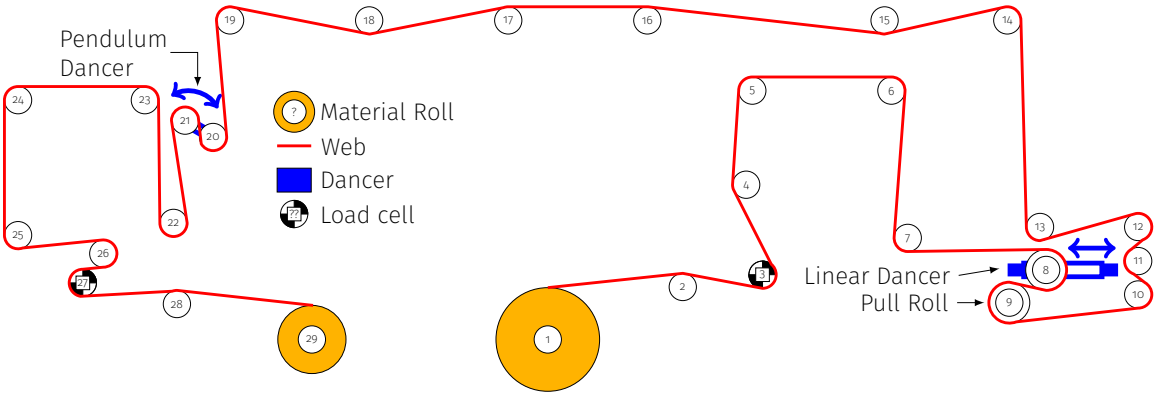



Figure 1.10: High Speed Web Line schematic. The Unwind Section is from rollers 1–9, the pull roll is number 9, and the Rewind Section is from 10–29. Rollers 20 and 21 are part of an S-Wrap dancer that rotates about a central pivot between the two rollers. Roller 8 is part of a linear dancer.

feedback: a load cell (roller 27) or an S-Wrap dancer (rollers 20 and 21). Lateral control is executed by two Fife™ web guides that incorporate rollers 5 and 6 and rollers 23 and 24. The Unwind section is from roller 1 through roller 9. The 'bypass path' bypasses the Process section made up of the Wrinkle module and the second Nip stand. The Rewind section is from roller 9 through roller 29.

The HSWL is controlled with an HMI for user input and RSLogix 5000® software for the programmatic details. RSLogix 5000® connects between the HMI and the PowerFlex 700S drives. where the high power electronics convert 480 VAC into the correct frequency and voltage to attain the desired speed and tension on the web line. RSLogix 5000® does calculations, data recording, monitors emergency stop buttons, and enforces sequencing through logical states of the web line operation.

## 1.5 Web Line Selection

Of the three lines described above, the research was mostly completed on the Euclid Web Line (EWL) and the High-Speed Web Line (HSWL). The Web Handling Research Center (WHRC) closed in December of 2019 which caused scheduling problems with the web lines as some of them were dismantled and removed. The HSLT web line was going to be the focus of research after the EWL, but other researchers needed access to it as well and there was no demand for the HSWL so that line became the focus of research. It did not have the accumulator, but it was technically able to attain much higher speeds than the EWL or the HSLT line and could handle larger width webs. The web path length is longer on the HSWL than on either of the other two lines.

## 1.6 Literature Survey

The literature survey is split up into four parts. The first part relates historical modeling of web handling components and linear models. The second part discusses the nonlinear models of web handling components. The third part surveys papers that modeled a web line in order to develop various control strategies. The fourth part describes the method of gain calculation residing on the EWL and HSWL called the Rockwell method herein.

### 1.6.1 Historical Literature

The following works detail the growth of modeling components of a web line as interest in the subject grew. First motors and spans were modeled with tension only. Then, tension control elements and strain in the web were analyzed.

Tension control has been the subject of many papers over the years. The early papers cited here have been cited in most papers that have followed. Campbell developed a dynamic model for tension in a web assuming small strains and Hooke's Law, and discussed several methods of tension control when a web is transported [14]. King modeled a small portion of a newspaper press (roller, web span, nip) assuming a linear elastic web, and demonstrated that an unbalance in the roller results in oscillations of tension in the web span. The magnitude of the tension oscillations is dependent on the web speed [15]. Grenfell modeled a simple nipped roller-web span-nipped roller system and showed the effects of disturbances in the roller speeds on the tension in the web [16]. Up to this point, the models have had one tension element. Brandenburg developed dynamic models that take into account spatial variations of parameters to analyze web lines where print registration is critical [17]. Shelton developed models for use in web tension control, and compared two methods of tension control - torque control and velocity control of a roller or rewinding roll of material [18]. Shin developed the concept of "primitive elements" in a web line, and used them to model web lines [19].

Reid and Shin considered the rewind of a web line where the motor drives the plant directly, and demonstrated that a web line that uses a variable gain Proportional, Integral, Differential (PID)

controller may have superior performance to a line with fixed gain controllers when there are significant time varying parameters [8]. The use of variable gains is referred to as "gain scheduling." A simplified model of the hypothetical system is a third order transfer function relating tension reference to output tension. Gains can be calculated based on performance criteria involving natural frequency and damping ratio.

Figure 1.11 is a block diagram of the system they studied. The tension error is an input to a PID controller $G_c$. The motor drive takes the output current of the controller and converts it to roll speed, while the plant uses the roll speed to calculate span tension and roll diameter. The method used to determine the gains for the PID controller will be discussed in the next section in this chapter.

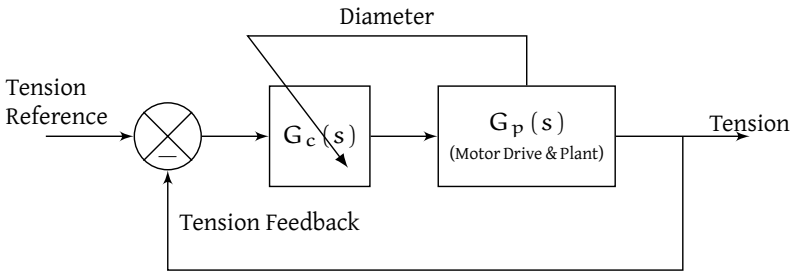



Figure 1.11: Block Diagram Describing the Gain Scheduling Control from [8]. The Plant has 2 outputs, tension and diameter. The diameter is a parameter used to vary the gains in the controller and the tension is the feedback signal.

Reid and Shin describe a roller, span, driven roller system like that shown in Figure 1.12. A web tension measurement is compared to a reference tension and the difference (error $e$) is fed to a PID controller that drives the a rewind roll at speed $V_2$ [8]. Reid, Shin, and Lin modeled basically the same thing in [7,20] and showed the power of a "build-up ratio" and variable gains. The system is modeled by making assumptions that produce a linearized set of equations for the system.

Reid and Shin show that the Laplace Transform transfer function of the system in Figure 1.11 is:

$$\frac{T_2(s)}{T_{2ref}(s)} = \frac{\beta_0 K_{dt} s^2 + \beta_0 K_{pt} s + \beta_0 K_{it}}{s^3 + (\alpha_1 + \beta_0 K_{dt}) s^2 + (\alpha_0 + \beta_0 K_{pt}) s + \beta_0 K_{it}} \tag{1.1}$$

where

$$\alpha_0 = \frac{V_{2,0}(B_{f2} + C_{m2}) + AER_2^2}{L_2 J_2} \tag{1.2}$$

$$\alpha_1 = \frac{V_{2,}0}{L_2} + \frac{B_{f2} + C_{m2}}{J_2} \tag{1.3}$$

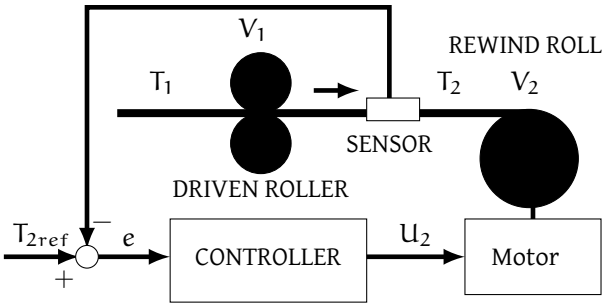



Figure 1.12: Physical Model of the Rewind System Considered in [8].

$$\beta_0 = \frac{K_{m2} A E R_2}{L_2 J_2} \tag{1.4}$$

By selecting a damping ratio, $\zeta$, and natural frequency, $\omega_{nat}$, for the desired response, the values of the derivative gain, $K_{dt}$, the proportional gain, $K_{pt}$, and the integral gain, $K_{it}$ can be found. The characteristic equation for (1.1) can be factored into

$$(s + k_r \zeta \omega_{nat})(s^2 + 2\zeta\omega_{nat} s + \omega_{nat}^2) \tag{1.5}$$

where $k_r$ determines a real pole. If $k_r > 10$, the effect of the first order root ($s = -k_r \zeta \omega_{nat}$) is small compared to the effect of the roots of ($s^2 + 2\zeta\omega_{nat} s + \omega_{nat}^2$ ) [21]. Comparing coefficients of the characteristic equations (denominators) of (1.1) and (1.5), the values of $K_{dt}$, $K_{pt}$, and $K_{it}$ are found as follows:

$$K_{it} = \frac{k_r \zeta \omega_{nat}^3}{\beta_0} \tag{1.6}$$

$$K_{pt} = \frac{1}{\beta_0}\left(\frac{2\zeta\omega_{nat}\beta_0 K_{it}}{\omega_{nat}^2} + \omega_{nat}^2 - \alpha_0\right) \tag{1.7}$$

$$K_{dt} = \frac{1}{\beta_0}\left(\frac{\beta_0 K_{it}}{\omega_{nat}^2} + 2\zeta\omega_{nat} - \alpha_1\right) \tag{1.8}$$

Then Reid and Shin introduce a "build-up ratio," $R_b$, defined to indicate the amount of web material on the roll. Equation (1.9) defines the build-up ratio where $R_{2,0}$ is the initial radius of the roll. Reid and Shin show that the moment of inertia, and equations (1.2), (1.3), and (1.4) can be stated as functions of build-up ratio, (1.9).

$$R_b = \frac{R_2}{R_{2,0}} \tag{1.9}$$

The concept of a build-up ratio and equations (1.6) and (1.7) will be used in chapter 3.

### Tension-Based Models

D. L. King reports on "J.A.R. Chrisholm's"[1] mathematical modeling of one of the first web systems in [15] (though, belts and capstans came earlier) that incorporates two kinds of friction, inertia, and eccentricity of the rollers. King used Hooke's Law to relate stress in a web to strain in a web, which was paper in this case. King reports a second-order differential equation as the model for the entire idle roller-span-nipped roller system (see Figure 1.13 (a)). Chrisholm multiplied strain by modulus so that he was dealing with tension and modulus instead of the quantity $(1+\varepsilon)$. King's work is at the heart of the tension model used in this research. Grenfell models a nipped roller-span-nipped roller model (see Figure 1.13 (b)) under draw control to highlight the need for high accuracy in the speed controls of paper handling machinery in [16]. He ignored tension transfer from one span to another by stating that the tension downstream of the span in consideration does not affect the current span's tension. With a no-slip condition assumed, tension from downstream spans does not affect upstream spans. Campbell analyzed weighted rolls (passive dancers) for tension control during dynamic situations in [14] using a model of the system he created and found weighted rolls wanting which Shelton agrees with in [22].

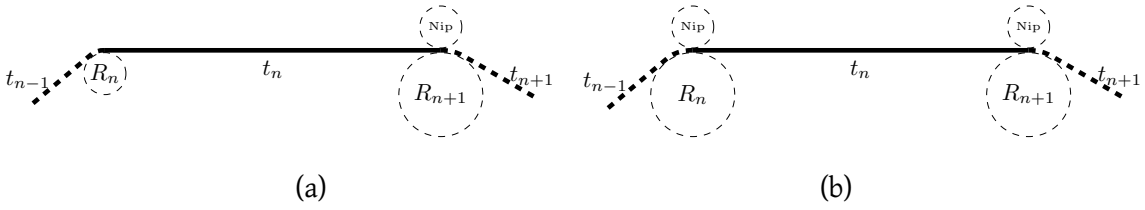



Figure 1.13: Tension-Based Model Systems, (a) King's model and (b) Grenfell's model

### Strain Transport

Shelton also identified strain transport from the previous span to the current span in [18]. Shelton does not contradict Grenfell because Shelton found that the previous span affects the next span, not that the next span affects the previous. He assumed small strains which may not be the case in some special applications today. Strain transport will be assumed in this research. Shin built

[1] King states the he is reporting the work of "J.A.R. Chrisholm" from Cambridge University. In looking for a follow-on paper that King mentions, the author found a J.S.R. Chrisholm who is a mathematical physicist and worked at Cambridge during the time frame of the paper. 'A' and 'S' are close together on a keyboard/typewriter. The mentioned follow-on paper was not found.

on Shelton's idea and developed a set of "primitive elements" to model specific components of a web line and then built a program to put together the components for simulations in [19]. His work with primitive elements will be used in chapter 2. Dwivedula and Pagilla described a situation in [23] where disturbances downstream could flow backwards into previous span when slip occurs, but that breaks one of the assumptions Grenfell was working under for his claims. The Dwivedula and Pagilla situation requires relaxing one of the assumptions that Grenfell, Shelton, and Shin all made, namely, that the web adheres to the roller. For good modeling of physical systems, the friction, stiction, and viscous friction model in [23] will be used later in this research and in the simulations in chapter 4.

### Variable Gains (Gain Scheduling)

Reid, Shin, and Lin modeled an unwind or rewind roll so that the change in radius and inertia as web is paid out is included in [7, 8, 20]. They showed that a PID controller optimized for one set of conditions would not be optimum for the duration of the roll. The changing radius of a winding or unwinding roll will be included in this research. They referred to it as a "build-up ratio." Young et al. described lateral and longitudinal web dynamics with a historical background and included variable gain PID control and a description of how to calculate gains for PID controllers in [24]. The PID gain calculation method is equivalent to the one in [8].

### Dancers

Carlson lays out guidelines and practical advice for choosing between a dancer and a load cell for tension control in a web line in [25]. He points back to Shelton's methods of unitless number (dimensional) analysis for making comparisons between different methods. The unitless quantities he shows may be helpful for comparisons of web lines or different methods of solving those web line models. Raul followed the dimensional analysis path to compare between dancer control and load cell control in [26]. Active dancers, where the resisting force of the dancer to the tension in the spans is varied dynamically, and passive dancers were studied by Dwivedula et al. in [27]. The authors of that paper define a dancer where the set point of its torque or position is changed

real-time as an active dancer. The tension zone requires a second tension measuring device like a load cell. The Euclid line and the HSWL both use a passive dancer under this definition. Pagilla and Dwivedula then show tension disturbance attenuation with an active dancer in [28]. The accumulator festoon could be considered a multi-roller dancer while it is controlled, under this definition. Martin improved the dancer by manufacturing dancers so that the rotary inertia was roughly equivalent to the translational inertia. The effects of these two inertias that every dancer has are opposing and the idea is that by making them equal, they will cancel each other out [29]. Martin patented the idea applied to an accumulator in [30].

### Minimum Resonant Frequency of Mulit-Span Systems

Pagilla and Diao dive into the system of idle roller equations for a web line, focusing only on the idlers and spans between driven rollers. They develop the system of equivalent springs and masses to find the minimum resonant frequency of the system of idlers in order to avoid exciting this lowest frequency in [12]. The minimum resonant frequency analysis was used to help understand frequencies found in the FFT analysis of the Euclid Web Line in Appendix C.

### Single Spans

Fox and Lilley modeled a single span using the wave equation in [31]. They model the web as a continuum defined by partial differential equations and use the finite difference method over a fixed set of points in the continuum to model the out-of-plane effects of the traveling wave equation. It is a hyperbolic Partial Differential Equation (PDE) so this method works. They show the effect of refining both the mesh and the time-step and then move to elongating the time-step for more time-efficient running and compare the results. Their method may become more important if the out-of-plane dynamics become important. In a similar vein, Brown looked at tension transferring down the length of the web in [32]. He does not assume that tension is constant in the span and at large time scales his results and those of the more common assumption agree, but at that point, he lost the effects of tension transferring.

### 1.6.2 Nonlinear Models

The previous section described works which used a linear analysis or a linearized analysis about an operating point. Real, physical web lines and materials rarely fit in the idealized realm of linear analysis. Next, some of the nonlinear models from the literature are discussed.

#### Length Changing Spans

Pagilla et al. derived a nonlinear span model for web spans in an accumulator in [9] and included time dependent roll radii and inertia for the rewinding and unwinding rolls. The nonlinear equation for span tension will be used in this research for dancers, out-of-round rolls, and accumulators and is derived in Appendix D. Branca et al. modeled out-of-round unwind rolls to account for the shape variation of the roll increasing and decreasing the mass flow of web into the initial span of the web transport system in [33] using the same equation with arbitrarily shaped rolls. This research uses a shape-of-the-roll definition (shown in Appendix I.3) and geometry to account for the time rate of change of the length of the initial span due to an out-of-round roll. An experiment and simulation of a bump on the unwind are presented in chapter 2, section 2.2.5. This is a simpler approach and arbitrarily shaped rolls are not possible with it.

#### Material Property Variations

Material property changes can be nonlinear. Boulter estimated loss torque, tension, and Young's modulus with extended Kalman filters for a web line in [34]. The described Kalman filter may be a useful mechanism to provide some validation of a variable Young's Modulus span model, but estimating is not the same as modeling where the model is the source of the variation. Kuhm and Knittel modeled an accumulator and the effects of varying mechanical properties like Young's modulus in [35]. They started with a nonlinear model and linearized about an operating point and then ran simulations with elastic modulus at the nominal value, at $5$ times the nominal, and at $1/5$ the nominal value. Gassmann and Knittel created an observer for the elastic modulus of the web in [36] which requires a plant to create the states to be observed in order to calculate the modulus. This is different than modeling for simulation which includes the variation of parameters.

### Traction/Slip

Web-roller interaction may also be nonlinear. Rice et al. collected experimental traction data from several webs on a nonvented roller to compare with theoretical values in [37]. They presented a new model for air gap height using non-Gaussian surfaces. Ducotey and Good produced a predictive traction model based on air entrainment in the steady-state in [38] which is part of the research on slip models in chapter 5. Their model assumed Guassian surface profiles and used the results from [39] to select air film height, roughness, and coefficient of static friction as key parameters for the traction model. They studied the effects of permeability and side leakage on air film height which impacts traction in [40]. Permeable webs would have applicability with the nonwovens used on the HSLT line. They applied squeeze film theory to study side leakage like Blevins shows in [41]. Ducotey and Good assumed infinitely wide webs initially while relatively narrow webs were necessarily used for the side leakage treatment. They devised a numerical method for calculating traction between a web and a grooved roller since a grooved roller is a common mitigation technique for air entrainment in [42]. Schüler et al. created a model for torque transfer that allowed them to perform sensitivity analyses to find significant parameters in [43]. They were able to combine effects of fluid pressure and solid contact together in their model which had coupled pressure and gap profile components but they were using a constant value for traction between web and roller.

Total slip is the loss of traction where there are different speeds between web and roller. Partial slip is an interface phenomena which is beyond the scope of this work. The factors involved in the initiation of total slip between a web and a roller have been studied in the past by Brandenburg [17, 44] who set up the notation and theory using continuum mechanics. The region of contact is assumed to be divided into two regions–a region of adhesion, and an exit region of slip.

Like Brandenburg, Whitworth [45, 46] also divides the region of contact on the roller into two regions, initially. He then adds a third region, the entry slip region, which can vanish. The roller where slip occurs is designated roller $n$, the incoming span as span $n-1$, and the outgoing span as span $n$. Each continuity equation for span $n-1$ and $n$ is combined with Hooke's law, to develop a pair of coupled equations relating the tensions $T_{n-1}$ and $T_n$ and velocities $v_n$ and $v_{n+1}$. Using

the relationships, equations are developed for the region of adhesion and the region of slip. Whitworth assumed that a temporary loss of adhesion between a web and a roller $n$ occurs, and that adhesion is maintained on rollers $n-1$ and $n+1$. When slipping occurs, the derivatives $\frac{\mathrm{d}T_{n-1}}{\mathrm{d}t}$ and $\frac{\mathrm{d}T_n}{\mathrm{d}t}$ are not defined so integrating in the presence of slip will not work. However, from dynamic considerations, $T_{n-1}$ and $T_n$ must be continuous functions of time. Whitworth then accounted for the strain in the web from the exit slip boundary on roller $n-1$, through the free span $n-1$, across the roller $n$, through the free span $n$, and up to the entry slip region boundary on roller $n+1$. This produces an equation in terms of strain. The strain in the web assuming adhesion is known which causes a certain effective length. A similar equation can be found for the amount of strain assuming slip over roller $n$. By equating the two equations for the amounts of strain, the following equations for tension can be derived.

$$T_{n-1}^s = \frac{L_{n-1}^e T_{n-1}^a + L_n^e T_n^a}{L_{n-1}^e + L_n^e e^{\mu_n \theta_{wn}}} \tag{1.10}$$

$$T_n^s = T_{n-1}^s e^{\mu_n \theta_{wn}} \tag{1.11}$$

The variables $L_{n-1}^e$ and $L_n^e$ are the effective lengths of the incoming and outgoing spans considering the roller is slipping, $T_{n-1}^a$ and $T_n^a$ are the incoming and outgoing span tensions assuming adhesion on roller $n$, $\mu_n$ is the coefficient of friction between web and roller, and $\theta_{wn}$ is the wrap angle of the web on roller $n$. Equations (1.10) and (1.11) will be used in a slightly modified form later in this research.

Ducotey and Good studied the effects of air entrainment on traction as noted above. A no-slip condition would be total traction while total slip would be no traction. Knowing characteristics of the web and roller, they were able to predict when total slip would occur. They demonstrated that air film thickness is key to the prediction of total slip or loss of traction.

Dwivedula and Pagilla [28] proposed an alternate method for modeling slip which they argued would be better than that those previously developed when developing tension control schemes for web processing systems. Expressions were developed for an effective friction force and an effective normal force when the web is slipping over the entire region of contact between the web and roller. It was necessary to define and employ a traction model. A classical model of traction

was selected, which included stiction, Coulomb friction, and viscous friction. The result is:

$$v_{r,i} = v_i + \frac{t_i}{b}\left(1 - p(1 - e^{-\mu_i \theta_{wi}}) - \frac{t_{i+1}}{t_i}\right) \tag{1.12}$$

where $v_{r,i}$ is the velocity of the roller, $v_i$ is the velocity of the web, $t_i$ is the tension in the incoming span, $t_{i+1}$ is the tension in the outgoing span, $\theta_{wi}$ is the wrap angle, $\mu_i$ is the coefficient of friction of the web on the roller, and $b$ and $p$ are constants.

If the web is not slipping on the roller, $v_{r,i} = v_i$. But if the web is slipping on the roller, the velocity of the roller is generally less than that of web. The reduction in velocity of the roller is dependent on the incoming and outgoing tensions, the wrap angle and the coefficient of friction. Equation (1.12) was combined with the dynamic equation for the incoming web span to show the effects of slip on tension. Two conclusions were that when slipping occurs (i) there may be tension oscillations in the incoming and outgoing spans, and (ii) disturbances in the downstream span may propagate upstream.

### 1.6.3 Modeling for Control

Sometimes, a web line is modeled to display a certain kind of control. Reid et al. modeled a idle roller-span-rewind subsystem to highlight the need for variable gains on a winding roll because of the changing radius in [8]. Variable radius models and the method for speed control will be incorporated in this research. Mathur and Messner describe the difficulty of designing a drive controller for tape drives from a time when tape backup was the only way to cost-effectively copy computer harddrives for failure mitigation in [47]. Their system is far smaller in scale than industrial web processing lines, but for that, they were much closer to the hard nonlinearity of zero measurable tension. In fact, they had a 1 oz. running tension. They build up a nonlinear model and use experimentation to validate reducing it to a linear model at an operating point. Mathur and Messner created a controller based on a sequential loop closing scheme from [48] and repeated testing. They had to increase gains during some periods of operation and decrease some gains at other times to maintain stable operation. They gave a method for estimating the roll radii without knowing the web thickness that works as long as there is no slack in the system. They brought up the importance of having known problem mitigation methods built in to the control mechanism.

Pagilla et al. reported on the dynamics of an accumulator in [10]. Pagilla et al. studied an accumulator subsystem to show the usefulness of tension observers and the control scheme they came up with in [13]. They used an average tension for accumulator spans instead of individual tensions in each span to formulate the control as will be used in this research. Pagilla simulated an accumulator during a zero-speed splice operation in [11] but does not model festoon rotation nor is span weight or festoon actuator dynamics included in the model as Kuhm and Knittel did in [35]. Nor does the model include web slip during a splice. Kuhm et al. compared control strategies in [49] using PI and $H_\infty$ controllers on a nonlinear model of a motor driven accumulator.

Oedl shows how elongation may be measured in a web process where encoders are used to indicate the position of all the rollers and thereby the position of a point on the web in [50]. This method is good for directly-driven rollers, but will work with idlers. The case where the path length changes is not covered so using this method with a dancer is problematic. The method can be used to characterize the out-of-roundness of a roller by examining the differences in the projected path and the real distance from the encoder. Obviously, slip between the web and the roller would nullify the method.

Gassman and Knittel added a friction observer to model web tension in a continuous loop system and used it to estimate tension in an unmeasured span in [51]. Constant bearing friction values are used in this research. Gassman et al. developed a nonlinear model for a pendulum dancer using a summation of angles, and linearized the model for use in developing an $H_\infty$ control scheme in [52]. This research does not use the small angle approximation and uses geometry to calculate the span length changes associated with pendulum movement to incorporate span length change effects into tension.

Branca et al. use the model developed in [33] to evaluate a method of controlling web tension in the presence of an unwinding roll with a flat on it in [53], but, again, did not include a whole subsystem in their analysis. Abjadi et al. created a sliding mode control for a $5$ roller, $4$ span web line to be robust to changes in Young's Modulus and other non-ideal effects in [54]. They use nonlinear control to linearize a coupled system. Their model is similar to what is used in this research, but to different ends; no attempt is made to verify their model against their web line.

Muthukumar et al. describe adaptive model predictive control applied to a web line with time varying parameters in [55]. Raul and Pagilla developed a Proportional-Integral model reference adaptive control in [56] on the EWL. They showed easily implementable options to plain PI control that harnessed model reference control and indirect adaptive control. Chang and Jung showed a robust method for selecting PID gains in discrete time applications using a robotic arm as a test case in [57] with good tracking results, but their whole system was discrete.

### 1.6.4 Gain Selection Method Resident in the Euclid and High-Speed Web Lines

Boulter describes how to use motor rated parameters to calculate speed controller gains in [58]. Motors are purchased based on rated parameters, so the values are well known and well tested. Boulter continued his theme of using motor or load parameters to calculate torque/current drive gains in [59] and position control gains in [60]. He quickly covers tuning an outer tension control loop in the appendix of [61]. The concepts of these papers were used in the creation of the gain calculation methods on the EWL and the HSWL. It is not known whether the control methods used in the HSLT line are following Boulter's concepts or not. Even so, the Rockwell Automation implementation of the Boulter's method leaves out several steps that he outlines in [58]. For example, the Rockwell implementation is not iterative while Boulter's paper does describe iteration as being a necessary part of the work. Also, the integral gain calculation is much simplified as there is no mention of the shock recovery that Boulter uses.

The diameter of the unwind and rewind rolls is discretely measured and updated, every two revolutions, during operation of the EWL or the HSWL. It is the responsibility of the operator to measure the unwind and rewind rolls and update the starting values on the HMI prior to the start-up operation (Steps V, XVI, and XVII in the test plan in Appendix G.2). The measurement of the rewind (or unwind) diameter could be accomplished by assuming the same tangential speed at the rewind as at the pull roll. Then the rotation rates of those two rolls could be compared because they both have encoders and the unwind, $n = 1$, or rewind, $n = 25$ ($n = 29$ on the HSWL), radius could be determined and the diameter is two times the radius.

$$R_n = \frac{R_{pr}\omega_{pr}}{\omega_n} \tag{1.13}$$

The pull roll can be any constant radius driven roll, but is generally selected to be the motor that sets the line speed. The assumption of equivalent tangential speeds can be a bad assumption during the start-up sequence. Instead, the software integrates the roll's (unwind or rewind) speed and the pull roll's speed. Once a set quantity of arc length has been accrued, the a similar equation is used to calculate the radius of the roll:

$$R_n = \frac{R_{pr}\theta_{pr}}{\theta_n} \tag{1.14}$$

and both integrators are reset to zero for the next arc length accrual period.

With the diameter (radius) in hand, the inertia of the roll can be estimated using the density of the web material and the volume of the roll. The RSLogix 5000® software utilizes the following equation to calculate the proportional speed gain.

$$K_{ps} = \frac{J_n(S_b)BW}{307.487(\tau_{rated})} \tag{1.15}$$

The $J_n$ is the total calculated inertia in lbm-ft$^2$ calculated with (2.5), $S_b$ is the rated speed ($1772$ for the $15$ hp motor) in RPM, $\tau_{rated}$ is the rated torque of the motor ($44.4$ for the $15$ hp motor) in lbf-ft, and the $BW$ is the system bandwidth in rad/s ($15$ on the EWL) according to [58]. The $307.487$ is commonly rounded to $308$ for simplicity[2]. The EWL and the HSWL both use a logic function to ensure that $K_{ps}$ does not become too large with the periodic recalculation of the roll inertia. Boulter describes this process in [58] where he shows a method to use the motor parameters and load characteristics to set the speed gains in terms of system bandwidth. Other than the bandwidth, the rest of (1.15) is what Boulter calls the 'Per Normal Inertia' calculation. Boulter includes a test to determine a system's specific per normal inertia experimentally instead of by calculation.

The integral speed gain is calculated from the proportional gain. The Euclid line drives use the following equation to calculate the Integral speed gain:

$$K_{is} = \frac{BWK_{ps}}{4(1.1)^2} \tag{1.16}$$

where the bandwidth, $BW = 15$, is divided by $4$ (Boulter gives are range of 3–5 for the divisor for phase margin stability by forcing the PI lead frequency to be smaller than the closed loop

[2] Observing the unit conversions being combined to form the number, it does not round to $308$ (maybe pre-2.54 cm per inch it did).

bandwidth) and $1.1^2$ is a safety factor per [62]. All together, the $15/4/1.1^2$ makes an open-loop crossover frequency of $3.1$ rad/s for the EWL. The HSWL speed loop gains are calculated with (1.15) for $K_{ps}$ and for the integral gain,

$$K_{is} = 0.4K_{ps} \tag{1.17}$$

where the $0.4$ is the open-loop crossover frequency in rad/s when the HSWL is in load cell feedback, speed control mode.

Equations (1.15) and (1.16) can be linked to performance goals like damping ratio and rise time by first conducting a simulation of the system for the case of a step input in the tension reference, and then estimating a damping ratio and natural frequency from the step response. This method will work reasonably well if the step response behavior is sufficiently like that of a second-order system.

The tension loop gains on the EWL were constants set by a previous researcher. The tension loop gains on the HSWL are not constant. The proportional gain begins with a target value and it is scaled by a "build-up ratio" and divided by the speed loop proportional gain.

$$K_{pt} = \frac{K_{pt\,target}\left(\frac{R_n}{R_{ec}}\right)^2}{K_{ps}} \tag{1.18}$$

The $K_{pt\,target}$ was $60$ on the HSWL and $R_{ec}$ was $1.5$ inches. The integral gain begins as a target value, the open-loop crossover frequency, that is multiplied by the tension loop proportional gain.

$$K_{it} = K_{it\,target}K_{pt} \tag{1.19}$$

The target value, $K_{it\,target}$ is $10$ rad/s in the HSWL software. The Rockwell method of gain calculation for load cell tension feedback is defined using equations (1.15), (1.17), (1.18), and (1.19). Since the proportional gain varies with roll radius, both proportional and integral gains vary with roll radius when using the Rockwell method.

### Laplace Domain Model of Speed-Based Tension Controller

Since both the EWL and the HSWL are controlled using RSLogix 5000® software and Allen-Bradley drives, the resident (Rockwell) control takes the form of proportional and integral (PI) control. The

unwind and rewind of both lines are speed-based tension control systems similar to Figure 1.14. The speed control gains are varied with respect to the roll diameter, but the tension loop is implemented with constant gains in $G_1$ for the EWL. The tension loop creates an adjustment to the linear speed of the web, not the rotating rate of the motor. The $K_c$ in Figure 1.14 serves the purpose of scaling the linear speed of the web to a rotating speed expected by the motor drive using the calculated roll radius and gear ratio. It is assumed that the field-oriented control (FOC) of the motor happens fast enough to allow the torque control aspects of the Rockwell drives to be considered instantaneous when compared to the time scales for speed control and tension dynamics. Chan says the torque delivery time is on the order of microseconds [63]. The HSWL control for the unwind and rewind motors is pictured in block diagram form in Figure 1.4 on page 6. The two figures are similar other than the equilibrium torque that is in Figure 1.4 which is the Motor Drive block in Figure 1.14. The RSLogix 5000® program for the HSWL calculates tension gains that vary with roll diameter unlike the tension gains for the EWL which were constants. Figure 1.14 can be considered as the block diagram of the rewind section in the EWL. Assuming one span precedes the rewind roll, it can be shown that the transfer function for the system is (1.20).

$$\frac{T_n(s)}{T_{ref}(s)} = \frac{\frac{K_{mn}AE}{J_nL_n}\left(\begin{array}{c} K_{ps}K_{dt}s^2 + (K_{pt}K_{ps} + K_{dt}K_{is})s^2 \\ +(K_{it}K_{ps} + K_{pt}K_{is})s + K_{it}K_{is} \end{array}\right)}{\begin{array}{c} s^4 + \left(\frac{K_{mn}AE}{J_nL_n}K_{ps}K_{dt} + \frac{B_{fn}+C_{mn}}{J_n} + K_{ps}\frac{K_{mn}}{J_n} + \frac{V_{n+1,0}}{L_n}\right)s^3 \\ + \left[\begin{array}{c} \frac{K_{mn}AE}{J_nL_n}(K_{ps}K_{pt} + K_{dt}K_{is}) \\ +K_{is}\frac{K_{mn}}{J_n} + \frac{V_{n+1,0}}{L_n}\left(\frac{B_{fn}+C_{mn}}{J_n} + K_{ps}\frac{K_{mn}}{J_n}\right) \end{array}\right]s^2 \\ + \left(\frac{K_{mn}AE}{J_nL_n}(K_{ps}K_{it} + K_{pt}K_{is}) + \frac{V_{n+1,0}}{L_n}K_{is}\frac{K_{mn}}{J_n}\right)s \\ + \frac{K_{mn}AE}{J_nL_n}K_{it}K_{is} \end{array}} \tag{1.20}$$

The index $n$ is 25 for all the parameters in (1.20) except for $L_n$, which is assumed to be equal to $L_{19} + \cdots + L_{24}$ (the total span length from the feedback device to the rewind). The characteristic equation is fourth-order, which leads to difficulty in finding numerical values for the gains $K_{ps}$, $K_{is}$, $K_{pt}$, $K_{it}$, and $K_{dt}$. Chapter 3 describes a method for calculating the gains in a systematic manner.

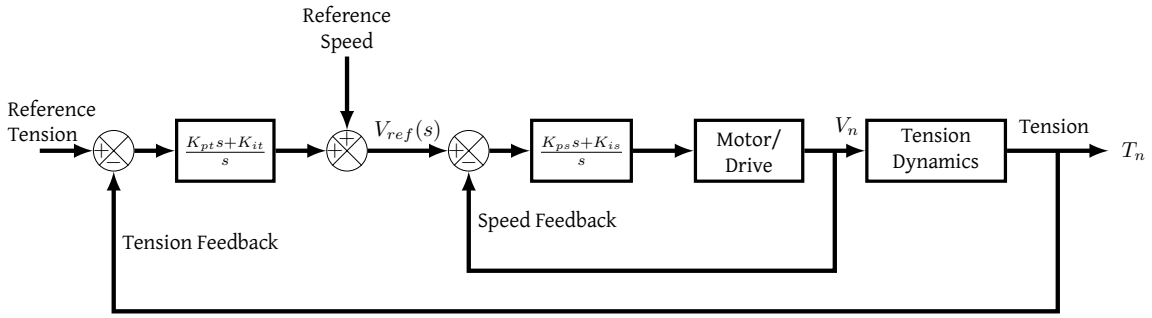



Figure 1.14: Euclid Line controller block diagram for the unwind and rewind rolls. The Plant has one output, the web tension. The tension is fed back to correct the speed reference.

## 1.7 Contributions

The contributions of this research are described in the following:

- **Modeling Primitive Elements**
  - Developed a model of the pendulum dancer that includes the nonlinear span length change effects due to the arc of swing of the pendulum
  - Developed the S-Wrap dancer model that includes the nonlinear span length change effects due the to rotation of the dancer about its pivot
  - Developed an online model of the non-circular parent roll that includes the nonlinear span length change effects due to the radius change as the bump rotates with the roll
  - Demonstrated the importance of nonlinear equations for simulating disturbances like a bump on the unwind
- **Determining the Efficacy of the Routh Approximation Method of Determining Controller Gains**
  - A systematic method for determining controller gains in a web handling system has been applied to two web lines at Oklahoma State University
  - Experimental verification of the gains calculated by the Routh Approximation method for ability to control tension within 0.5% error and a standard deviation of 0.883 lbf and speed within 0.25% error and standard deviation of 1.00 FPM on the average for steady-state operation on the High-Speed Web Line
- **Using Simulation**
  - Described how the model of the High-Speed Web Line was built up from primitive elements and factors for consideration when developing a web line simulation
  - Showed comparisons of experimental results and simulation results from the High-Speed Web Line for two materials performing a 0–800 FPM start-up

  - Showed examples of how a simulation could be used by a web line designer to test new conditions in an existing web line or build a model of a new web line

- **Modeling of Slip Between a Web and a Roller and Simulation of the System Model of the Euclid Line to Determine the Efficacy of Three Slip Models**
  - Determined that the Whitworth slip model underestimates the change in roller speed due to the impact of slip on tensions across a roller
  - Developed a model for slip utilizing sliding-friction between the web and the roller that uses the Whitworth criteria and model, but with modification of the dynamic equation for the roller where slip occurs
  - Implemented the Ducotey-Good Traction Model in a simulation of the Euclid Web Line and indicated rollers where slipping between the web and roller could occur
  - Simulated an idle roller on the Euclid Web Line in a slip condition with Ducotey-Good Traction and with the Sliding-Friction Driven Roller model

- **Experimental Studies with the Euclid Line to Determine the Efficacy of Slip Models**
  - Determined that the Sliding-Friction Driven Roller model is better than the Whitworth model for certain cases
  - Verified rollers indicated by the simulation with the Ducotey-Good traction model had slip occurring via roller speed measurement
  - Measured roller speed experimentally for the roller simulated with both Ducotey-Good and Sliding-Friction Driven Roller models showing the experiment switched between the two simulation methods

# CHAPTER II

# MODELING WEB LINES

## 2.1 Modeling Web Lines with Primitive Elements

Web processing lines contain sub-systems which are themselves made up of individual components like motors, rollers, and spans. The individual components were termed "primitive elements" in [8, 19] by Shin who defined several mathematical primitive elements such that the output of one was the input to the next. For example, a dancer sub-system, either translational or pendulum, is made up of an incoming span, a roller attached to a moving support, and an outgoing span. The dancer may be passive or active. The Euclid Web Line (EWL) has a passive pendulum dancer to reduce tension disturbances by being a part of a position regulating controller [25]. The torque imbalance of the tension in the spans wrapped around the dancer roller and the input torque on the dancer arm causes motion in the dancer arm. The motion allows the dancer to smooth out tension variations in the line which is its main function.

Each primitive element, including the web span, driven roller, idle roller, unwind roll and rewind roll, has specific 'parameters' that are sometimes shared among many elements and sometimes specific to one element. These elements are first-principles models of individual components in the web line. These primitive elements can be linked together in such a way that the output of one element is the input to the next one. In this way a model of the web traveling through the web line can be created. Using various sensors and motors (each of which is a primitive element), control of both speed and tension in the web may be attained. Models for some of the primitive elements used in a typical web processing line are presented below. In following sec-

tions, subscripts describe the roller or span index number. The velocity of the $n^{\text{th}}$ roller is $v_n$ and the tension of the previous span to the $n^{\text{th}}$ span is $t_{n-1}$.

### 2.1.1 The Idle Roller Primitive Element

The primitive element idle roller, whose physical model is shown in Figure 2.1 on the left, is modeled with a torque balance about the roller's axis. The difference in tension of the incoming and outgoing spans (and the assumption of no slip) and the bearing friction form the main drivers for the primitive element model shown below. The last term accounts for eccentricity in the idle roller as shown on the right of Figure 2.1. The offset, $e_n$ is the radial distance between the center of mass and the center of rotation and $M_n$ is the mass of the rotating roller.

$$\frac{\mathrm{d}v_n}{\mathrm{d}t} = \frac{-B_{fn}}{J_n}v_n + \frac{R_n^2}{J_n}(t_n - t_{n-1}) + \frac{R_n M_n}{J_n} g e_n \sin(\omega_n t + \phi_{n,0}) \tag{2.1}$$

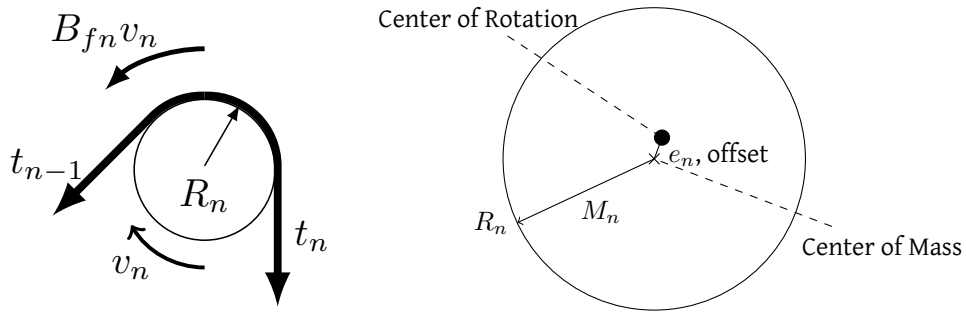



Figure 2.1: Physical Model of the Idle Roller (left) and a Model of an Idle Roller Eccentricity (right)

### 2.1.2 The Free Span Primitive Element

Any section of the web not contacting a support is considered a free span. Tension and material properties are initially assumed to be constant throughout the span. The primitive element model is derived in [9, 19, 26, 64] from the principle of conservation of mass in the span which defines a control volume as shown in Figure 2.2. The equation was initially in terms of material strain, but through assumption and application of Hooke's Law, a description of the change in tension of the span is developed. The equation for the change in tension in a span of material is

$$L_n \frac{\mathrm{d}}{\mathrm{d}t} t_n(t) = v_{n+1}\left(E_n A_n - t_n(t)\right) - v_n\left(E_n A_n - \frac{E_n A_n}{E_{n-1} A_{n-1}} t_{n-1}(t)\right) \tag{2.2}$$

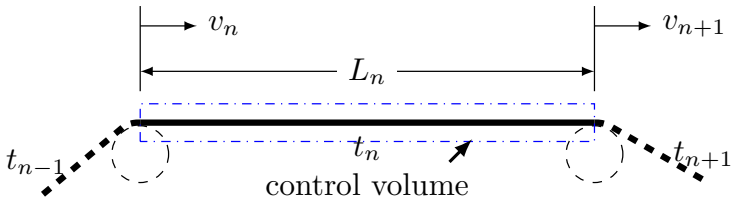



Figure 2.2: Free Span Supported by Two Rollers. The material properties of cross-sectional area and Young's Modulus are assumed constant in the span.

where the parameters are length, $L_n$, tension, $t_n$, roller velocity at the span end, $v_{n+1}$, Young's modulus, $E_n$, and cross-sectional area for the web material, $A_n$, roller velocity at the span beginning, $v_n$, and the tension in the previous span, $t_{n-1}$. The ratio of Young's modulus and area from the current span to the previous one is present to allow properties of a span to change across a roller. The form of the previous equation when linearized about an operating point is

$$L_n(t)\frac{\mathrm{d}}{\mathrm{d}t}t_n(t) = E_nA_n(v_{n+1} - v_n) - V_{n+1}t_n(t) + V_n\frac{E_nA_n}{E_{n-1}A_{n-1}}t_{n-1}(t) \tag{2.3}$$

where $V_n$ and $V_{n+1}$ are the steady-state velocities at an operating point, $v_n$ and $v_{n+1}$ are the small variations around the operating point, and $t_n(t)$ and $t_{n-1}(t)$ are the small variations in tension about the operating point.

### 2.1.3 The Unwind Roll and Motor Primitive Element

The unwind roll is a motor-driven spindle that can be operated in either a motoring or braking mode. Usually, if the unwind roll motor is controlled by tension feedback, the tension control element of the motor is acting as a brake. The describing equation using $n = 1$ is (from [19]),

$$J_1\frac{\mathrm{d}v_1}{\mathrm{d}t} = -\left(B_{f1} + \frac{C_{m1}}{\mathsf{g}_{r_n}}\right)v_1 + R_1^2t_1 + \frac{R_1K_{m1}}{\mathsf{g}_{r_n}}u_1 + R_1M_1ge_1\sin(\omega_1t + \phi_{10}) \tag{2.4}$$

where the parameters are the rotational inertia, $J_1$, bearing friction, $B_{f1}$, the motor-brake damping, $C_{m1}$, the roll radius, $R_1$, the exiting span tension, $t_1$, the motor torque constant[1] (torque per amp), $K_{m1}$, the motor input from the control, $u_1$, and the eccentricity term where $M_n$ is the mass of the roll, $g$ is the acceleration of gravity, $e_n$ is the offset of the center of rotation from the center of gravity of the roll as shown in Figure 2.1, and the $\sin(\ldots)$ term is the component of the torque

[1] Boulter uses $\frac{(rated\ max\ torque)}{(rated\ max\ amps)}$ to define $K_{m1}$ in [61].

due to the offset from the rotation of the roll itself. The term $(\omega_1 t + \phi_{10})$ can be more accurately modeled with a separate state for the angular position of the eccentric roll. Equation (2.4) is stated from the roll's perspective with the state variable being the tangential speed of the roll.

Figure 2.3 shows the parameters $J_{web}$, $J_{sn}$, and $J_{mn}$ instead of $J_1$. The total inertia, $J_1$, from the roll's perspective is the sum of the roll of web inertia, the core (that the web is wound on) inertia, the spindle inertia, and the motor inertia which is reflected through the gear ratio. The core inertia, $J_{core}$, is sometimes neglected as though the web is being wound directly on the spindle. The last three terms do not change with time, only $J_{web}$ does. Equation (2.5) shows the equivalent inertia from the roll's perspective (or at the roll's rotational speed).

$$J_1 = J_{web} + J_{core} + J_{s1} + \frac{J_{m1}}{\mathrm{gr}_1^2} \tag{2.5}$$

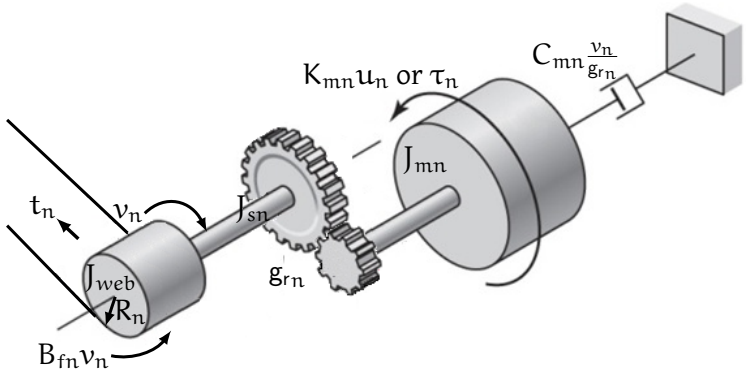



Figure 2.3: Physical Model of the Unwind Motor and Roll (adapted from [65]). The web wrap direction could be changed to over-wrap instead of under-wrapping as shown for the rewind motor configuration.

Modeling the AC or DC motor as a constant gain, $K_{mn}$, is not ideal, but the internal dynamics of the motor are very fast (on the order of microseconds) compared to dynamics in rest of the web line (on the order of milliseconds). The modeling of the currents of in the 3-phase winding of an AC motor is beyond the scope of this study. "…[T]he induction motor can be referred to as a dynamic, recurrent, and nonlinear system" [63]. Linearization of such a system is difficult and fraught with danger, but Field Oriented Control (FOC) as is used in the Rockwell drives attempts to do just that. Equation (2.6) shows the motor model in rotational speed terms and from the motor's

perspective.

$$J_1 \frac{\mathrm{d}\omega_1}{\mathrm{d}t} = -\left(B_{f1}\mathsf{g}_{\mathsf{r}1} + C_{m1}\right)\omega_1 + R_1 t_1 \mathsf{g}_{\mathsf{r}1} + \tau_1 + M_1 g e_1 \mathsf{g}_{\mathsf{r}1} \sin(\omega_1 t + \phi_{10}) \tag{2.6}$$

$$J_1 = (J_{web} + J_{core} + J_{s1})\mathsf{g}_{\mathsf{r}1}^2 + J_{m1} \tag{2.7}$$

The inertia, $J_1$, the equivalent inertia from the motor's perspective. The motor torque, $\tau_1$, is the torque applied by the motor. To estimate the motor torque, Pagilla et. al showed the following equation, called forced equilibrium, in [4]. The motor speed reference and acceleration, $\omega_{1,ref}$ and $\dot{\omega}_{1,ref}$, are from the Industrial S-Curve speed reference (see Appendix I.2).

$$\tau_1 = J_1 \dot{\omega}_{1,ref} + \left(B_{f1}\mathsf{g}_{\mathsf{r}1} + C_{m1}\right)\omega_{1,ref} - R_1 t_1 \mathsf{g}_{\mathsf{r}1} \tag{2.8}$$

Assuming no-slip in the outer layer of the roll and the thickness of the web is negligible in comparison with the roll radius, the web velocity, $v_n$, is the tangential velocity of the roll. Equation (2.4) is an idle roller primitive element, (2.1), with one less span and an input torque added. A driven roller in the middle of a web line like the EWL's S-Wrap Lead motor is modeled with (2.9). A rewind motor is modeled with (2.10).

$$\frac{\mathrm{d}v_n}{\mathrm{d}t} = \frac{1}{J_n}\left[-\left(B_{fn} + \frac{C_{mn}}{\mathsf{g}_{\mathsf{r}n}}\right)v_n + \frac{R_n K_{mn}}{\mathsf{g}_{\mathsf{r}n}} u_n + R_n^2 (t_n - t_{n-1})\right] \tag{2.9}$$

$$\frac{\mathrm{d}v_n}{\mathrm{d}t} = \frac{1}{J_n}\left[-\left(B_{fn} + \frac{C_{mn}}{\mathsf{g}_{\mathsf{r}n}}\right)v_n + \frac{R_n K_{mn}}{\mathsf{g}_{\mathsf{r}n}} u_n - R_n^2 t_{n-1}\right] \tag{2.10}$$

### 2.1.4 The Nip Roll Primitive Element

Adding a nip roll to a driven roller ensures that web does not slip. The derivation found in Appendix H.7 comes from the Web Transport System - V2.1 Help, [66]. It is useful for finding the impact of a nip on the longitudinal dynamics of the web, but not for things like web compression or lateral (cross-machine) strain. A driven roller with a nip would use the following governing equation. The parameters $Rn_n$, $Jn_n$, and $Bn_{fn}$ refer to properties of the nip roller which are analogous to a normal roller. On both the EWL and the High-Speed Web Line (HSWL), the nips are the same size and construction as the normal idle rollers.

$$\frac{\mathrm{d}}{\mathrm{d}t}v_n = \frac{1}{(R_n^2 Jn_n + Rn_n^2 J_n)} \left[ -\left( Rn_n^2 (B_{fn} + \frac{C_{mn}}{\mathrm{g}_{\mathrm{r}n}}) + R_n^2 Bn_{fn} \right) v_n \right.$$

$$\left. + Rn_n^2 R_n^2 (T_n - T_{n-1}) + Rn_n^2 R_n \frac{K_{mn}}{\mathrm{g}_{\mathrm{r}n}} u_n \right] \tag{2.11}$$

### 2.1.5 The Lead-Lag Filter

The lead-lag filter is used extensively in industry and a derivative model of the device was found in [67] which reports the differential equations from [68]. The filter requires a state of its own, $x$, and the input, $u$, from which it calculates the filtered output, $y$, based on the lead and lag time constants given to it as parameters, $T_{ld}$ and $T_{lg}$.

$$\dot{x}(t) = \frac{1}{T_{lg}}(u(t) - x(t)) \tag{2.12}$$

$$y(t) = \frac{T_{ld}}{T_{lg}} u(t) + \left(1 - \frac{T_{ld}}{T_{lg}}\right) x(t) \tag{2.13}$$

The initial condition is $x(0) = y(0) = u(0)$.

### 2.1.6 The Pendulum Dancer Primitive Element

The pendulum dancer (see Figure 2.4) is an idle roller combined with a pivoting arm. There is an input torque, $t_{qn}$, that counters the moments resulting from the tensions in the incoming and outgoing web spans in the steady-state. If it is a constant torque, the dancer is a passive one, but if the torque is controlled, it is an active dancer [27]. The pendulum dancer can have a rotational damping, $C_{pn}$, a rotational spring, $K_{pn}$, and does have a rotational inertia, $J_{pn}$. The pendulum arm weight component of the torque balance is assumed to be negligible for derivation, but in simulation, the weight of the pendulum arm was calculated and used. Figure 2.4 shows a diagram for the pendulum dancer. The model for the dynamic behavior of the pendulum dancer is made

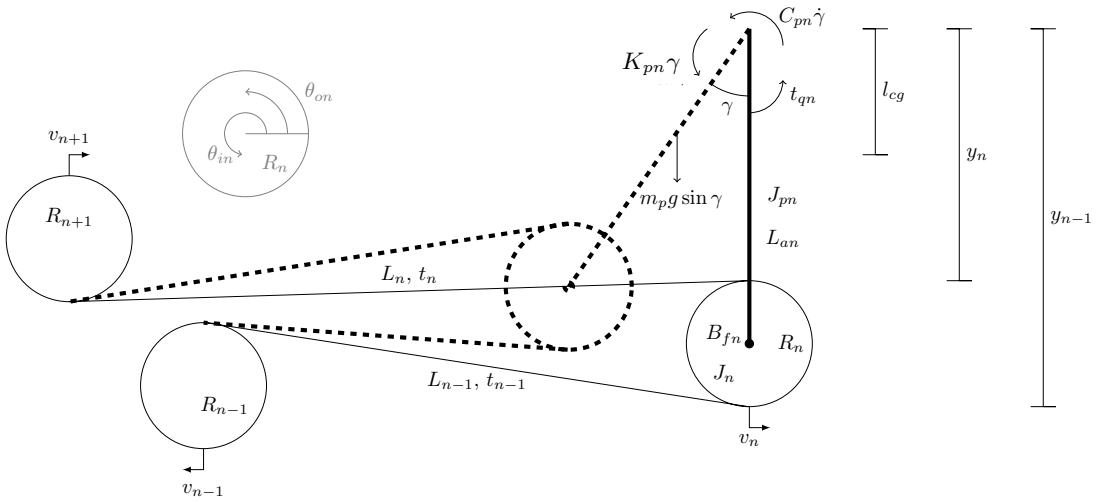



Figure 2.4: Pendulum Dancer Physical Subsystem Model. The gray circle inset is shows the sign convention for $\theta_{in}$ and $\theta_{on}$. The dashed outline shows the configuration of pendulum and spans at the short-span end of the pendulum swing.

up of a set of equations (from [19]). The first pair relates the pendulum arm rotational dynamics:

$$\frac{\mathrm{d}}{\mathrm{d}t}\gamma_n = \dot{\gamma} \tag{2.14}$$

$$J_{pn}\frac{\mathrm{d}}{\mathrm{d}t}\dot{\gamma} = -C_{pn}\dot{\gamma} - K_{pn}\gamma + t_{qn} - t_{n-1}y_{n-1} - t_n y_n - m_p g \sin\gamma \tag{2.15}$$

where $J_{pn}$ is the inertia of the pendulum, $\gamma$ is the angle of swing of pendulum, $C_{pn}$ is the the pivot damping, $K_{pn}$ is the rotary spring constant, $t_{qn}$ is the input torque, and $y_n$ and $y_{n-1}$ are the moment arms for the tensions from the web spans (see Figure 2.4). The velocity dynamics of the roller are:

$$J_n\frac{\mathrm{d}}{\mathrm{d}t}v_n = -B_{fn}v_n + R_n^2(t_n - t_{n-1}) \tag{2.16}$$

where $J_n$ is the rotational inertia of the roller itself and $B_{fn}$ is the bearing friction of the roller. The web tension linearized dynamic equations for the incoming and outgoing spans are

$$\begin{aligned} L_{n-1}\frac{\mathrm{d}}{\mathrm{d}t}t_{n-1}(t) &= E_{n-1}A_{n-1}(v_n(t) - v_{n-1}(t)) + \left(V_{n-1}\frac{E_{n-1}A_{n-1}}{E_{n-2}A_{n-2}}t_{n-2}(t) - V_n t_{n-1}(t)\right) \\ &\quad + (E_{n-1}A_{n-1} - t_{n-1})L_{a\,n+1}\dot{\gamma}_{n+1}\sin(\theta_{in+1}) \end{aligned} \tag{2.17}$$

$$\begin{aligned} L_n\frac{\mathrm{d}}{\mathrm{d}t}t_n(t) &= E_nA_n(v_{n+1}(t) - v_n(t)) + \left(V_n\frac{E_nA_n}{E_{n-1}A_{n-1}}t_{n-1}(t) - V_{n+1}t_n(t)\right) \\ &\quad + (E_nA_n - t_n)L_{an}\dot{\gamma}_n\sin(\theta_{in}) \end{aligned} \tag{2.18}$$

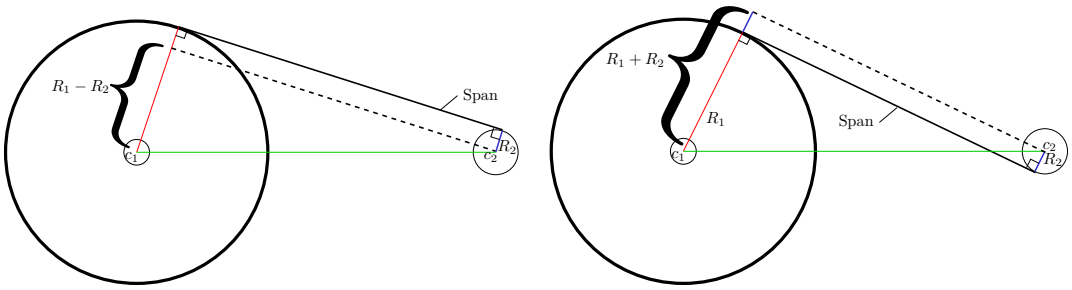
where $\dot{\gamma}_n$ is the angular velocity, $L_{an}$ is the pendulum arm length as shown in the figure, and $\theta_{on}$ is the outgoing web span angle just as $\theta_{in+1}$ is the incoming web span angle. Unlike the models for the span, unwind roll, and idlers, the last two equations are nonlinear. The last term in each of the above two equations is needed in order to model the effect of the dancer movement on span tension.

## 2.2 Nonlinear Primitive Elements

The pendulum dancer discussion noted that the span tension equations were nonlinear. Even so, the equations were linearized somewhat by applying the small angle approximation to estimate the change in length of the span with the pendulum swing angle change: i.e., $\dot{\gamma} L_{an} \sin(\theta_{in})$ which assumes the pendulum does not swing very far. The approximation limits the usefulness of the model. The nonlinear model would be valid over the whole swing of the pendulum. Numerical solvers like MATLAB allow the use of nonlinear equations with a high degree of accuracy. Nonlinear primitive elements that include the length changing span are the non-circular roll, the pendulum dancer, and the S-wrap dancer. The derivation of the length changing span tension equation can be found in Appendix D.

### 2.2.1 The Non-Circular Unwind Roll

The Euclid Web Line (EWL) unwind section was modeled using a nonlinear span equation in the first span connecting the unwind roll to the web line. The unwinding roll shape had to be characterized to be able to calculate the span length change due to the shape of the roll passing by the point where the span is tangent to the roll. The length of the span from the unwind roll to the first idler is found by calculating the line tangent to those two cylinders as shown in Figure 2.5. The ideal web path follows a line that is tangent to both outer circumferences at its end points. The directions of rotation of the roll and roller involved have to be known and in the case of the EWL, the orientations are like (b) in the figure. The center-to-center distance of the rollers involved is a known value and shown in green. Thus, a right triangle can be formed whose hypotenuse is the center-to-center distance, one side is the sum of the two roller's radii, and the last side is the




(a) (b)

Figure 2.5: Unwind Span Geometry for (a) Both Rollers Rotating Clockwise and (b) One Clockwise, One Counter-Clockwise. The Euclid Line was a situation like (b).

span length (the dashed line). When two sides are known, the Pythagorean theorem may be used to find the third side which defines $S(\theta)$, the span length function. To find the rate of change of the span length, $S(\theta)$ has to be differentiated with respect to the unwind roll's radius. The roll's radius change is a function of time and also of position of the unwind roll, $r(\theta, t)$ (the bump is not moving around the roll; it has a fixed position relative to the roll; the roll is rotating). Assuming all other components of span length change are negligible, the derivative of the span length for span 1 can be found with the equations below. Equation (2.20) uses angles from measuring a bump on the EWL unwind.

$$S(\theta) = \sqrt{\left(d(c_1, c_2)\right)^2 - \left(r_1(\theta(t)) + r_2\right)^2} \tag{2.19}$$

$$r_1(\theta) = r_{1,0} + \left(\Phi(\theta - 30^\circ) - \Phi(\theta - 84^\circ)\right) \cdot \left(a_1 \cdot \left[1 - \cos\left((\theta - 30^\circ) \cdot \frac{180^\circ}{54^\circ}\right)\right]\right) \tag{2.20}$$

$$\frac{\mathrm{d}S}{\mathrm{d}t} = \frac{\mathrm{d}S}{\mathrm{d}r_1} \cdot \frac{\mathrm{d}r_1}{\mathrm{d}\theta} \cdot \frac{\mathrm{d}\theta}{\mathrm{d}t} \tag{2.21}$$

The function $d(c_1, c_2)$ is Euclidean distance between 2 X-Y coordinate points that are the centers of the unwind roll, $c_1$, and the next roller, $c_2$. To find the radius of the unwind roll with respect to $\theta$, a coordinate system is arbitrarily drawn on the roll with its origin concentric with the roll and the position and extent of the bump is found by inspection (see Figure 2.6). Then the radius is modeled as a constant plus a $a_1(1 - \cos[\alpha\theta + \phi])$ curve with limits at the extents of the bump. The bump begins and ends with zero velocity and begins and ends with zero displacement. Heavyside step functions were used to limit the cosine wave to the angular position of the bump. The MATLAB codes used are in Appendix I.3. The mapping, $\alpha$, is used to allow the cosine to go through $\pi$ radians

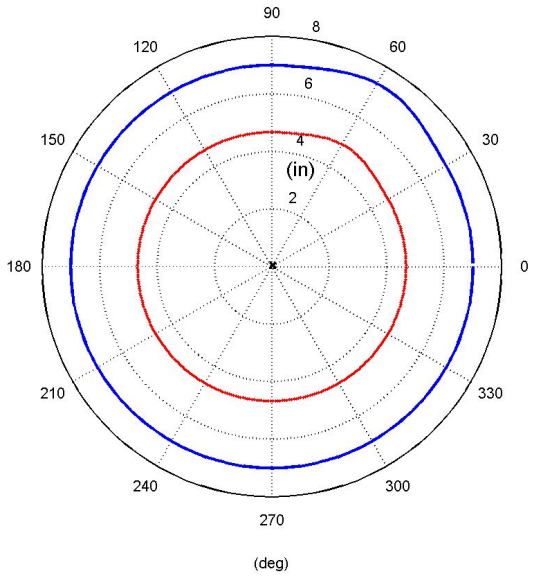



Figure 2.6: Unwind Radius (blue) and Associated Centroids of Triangles (red) used for Calculating the Eccentricity Due to the Bump. See Appendix I.3 for more detail.

during the arc length of the bump and $\phi$ is the offset angle from the arbitrary coordinate system to the bump. So the first span length is dependent on radius, which is dependent on $\theta$, and $\theta$ is dependent on time. The unwind roll is rotating so the derivative of $\theta$ with respect to time is the angular velocity of the roll which is measurable. In this manner, the span length and rate of change of the first span's length may be calculated given the shape of the roll.

The model for the span tension is dependent on several things as described in a previous section, but one of them is the entering velocity, $v_1$ (see Figure 2.2). In the case of the non-circular unwind roll, that velocity is the tangential velocity of the roll. Assuming the amount of web unwound from the roll in one revolution is

$$c = \int_0^{2\pi} r(\theta)\mathrm{d}\theta = \int r(\theta)\mathrm{d}\theta \tag{2.22}$$

and taking the derivative of the above equation with respect to time applies to $\theta$ because it can stand the differentiation:

$$\frac{\mathrm{d}c}{\mathrm{d}t} = \frac{\mathrm{d}c}{\mathrm{d}\theta}\frac{\mathrm{d}\theta}{\mathrm{d}t} = \frac{\mathrm{d}}{\mathrm{d}\theta}\left(r(\theta)\theta\right)\frac{\mathrm{d}\theta}{\mathrm{d}t} = \left(r(\theta) + \theta\frac{\mathrm{d}r(\theta)}{\mathrm{d}\theta}\right)\frac{\mathrm{d}\theta}{\mathrm{d}t} \tag{2.23}$$

where $r(\theta)$ is the radius of the unwind roll as a function of $\theta$. Thus the tangential speed of the roll can be calculated. With the bump comes a small eccentricity due to the web not being evenly

distributed (as well as the mass of the bar in the roll used to form the bump). Since the roll shape is known, the eccentricity can be calculated using the density of the web and applied using the equation described in a previous section.

In summary, for the non-circular roll primitive element, the span length was found along with the rate of change of the span length. The roll tangential speed was found as a function of radius, angular position, and angular velocity. And finally, the eccentricity of the roll due to its non-circular shape was found. So, the model for a non-circular roll is a torque balance of the roll, the roll speed is calculated separately using $\mathrm{d}c/\mathrm{d}t$, and the span tension is calculated with changing length included which is a function of roll shape.

### 2.2.2 Time-Varying Radius Unwind Roll Primitive Element

Even if a bump is not modeled on a parent roll, the diameter of the roll intrinsically changes as the web line operates and web is paid off into the system. The tangential speed of the roll is still necessary for the span tension calculations. The model for an unwind roll where the diameter reduces by two thicknesses of the web per rotation (or one web thickness in radius per rotation) has variable inertia as well. The rate of radial decrement is found by multiplying change in radius per revolution by the rotation rate of the roll.

$$\frac{\mathrm{d}r_1}{\mathrm{d}t} = \frac{-t_{web}}{2\pi}\omega_1 \tag{2.24}$$

The radius of the unwind is $r_1$, $t_{web}$ is web thickness, and $\omega_1$ is the rotation rate of the unwind roll in radians per second. The tangential roll speed, $v_1$, is needed for span tension calculations and has to be separately calculated.

$$\frac{\mathrm{d}v_1}{\mathrm{d}t} = \frac{\mathrm{d}}{\mathrm{d}t}(r_1\omega_1) = \omega_1\frac{\mathrm{d}r_1}{\mathrm{d}t} + r_1\frac{\mathrm{d}\omega_1}{\mathrm{d}t} \tag{2.25}$$

Tracking the roll radius adds two states to the simulation: one for the angular speed of the roll which can also be the shaft speed of the motor with a little manipulation, and the radius.

The motor model for use with time-varying radius parent roll describes the motor angular acceleration. The summation of torques about the motor shaft is used to define the differential equation, but the inertia is calculated from the motor's perspective and the gear ratio, $g_r$, is applied

to the web tension, the bearing friction of the spindle bearings, and the moment caused by the eccentricity.

$$J_1 \frac{\mathrm{d}\omega_1}{\mathrm{d}t} = -\left(B_{f1}\mathsf{g}_{\mathsf{r}1} + C_{m1}\right)\omega_1 + r_1 t_1 \mathsf{g}_{\mathsf{r}1} + \tau_1 + \mathsf{g}_{\mathsf{r}} M_1 g e_1 \sin(\theta_1) - 2\rho_{web}\mathsf{w}_{web}\pi r_1^3 \dot{r}_1 \omega_1 \quad (2.26)$$

The $-2\rho_{web}\mathsf{w}_{web}\pi r_1^3 \dot{r}_1 \omega_1$ term is the derivative of the roll's inertia. The $\dot{r}_1$ is from (2.24).

### 2.2.3 The Nonlinear Pendulum Dancer Primitive Element

Through a similar derivation, the span length of spans around a pendulum dancer may be calculated. The geometry of the hardware and the angle of the dancer arm are needed to calculate the dancer spans' lengths. Knowing the positions of the roller on the incoming span and where the dancer roller's center is when the arm is at an angle, $\gamma = 0$, a right triangle may be formed whose hypotenuse is the span length. Figure 2.7 shows this physical model. This equation may be differ-

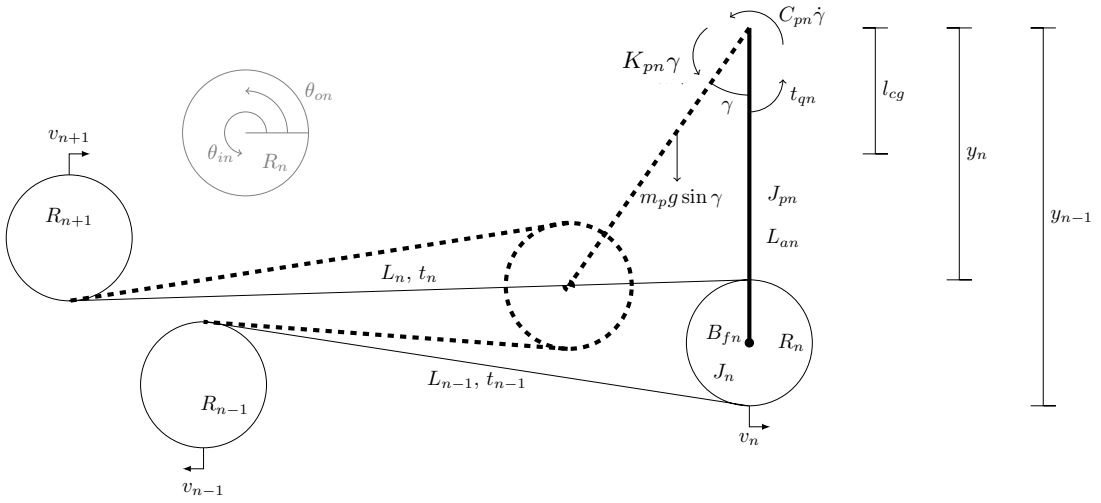



Figure 2.7: Pendulum Dancer Physical Model. The gray circle inset is shows the sign convention for $\theta_{in}$ and $\theta_{on}$.

entiated with respect to time to obtain the rate of change of the span length based on pendulum position. The angle $\gamma$ is defined to be 0 at the center of its swing. The span length can be defined in terms of initial, unstretched length, constant distances and the angle $\gamma$ as

$$L_n = \sqrt{\left(L_{nc} - L_{an}\sin(\gamma)\right)^2 + \left(h_n - R_n - R_{n-1} + L_{an}(1 - \cos(\gamma))\right)^2} \quad (2.27)$$

where $L_{nc}$ is the initial, unstretched length of the span when the dancer is at $0^\circ$ angle, $h_n$ is the difference in heights of the centers of the rollers $R_n$ and $R_{n-1}$. Differentiating the previous equation

with respect to time gives

$$\frac{\mathrm{d}L_n}{\mathrm{d}t} = \frac{1}{2}\frac{(-2L_{an}(L_{nc} - L_{an}\sin\gamma)\cos\gamma + 2L_{an}(h_n - R_n - R_{n-1} + L_{an}(1-\cos\gamma)\sin\gamma))\dot{\gamma}}{L_n} \tag{2.28}$$

which is the rate of change of the span length with respect to $\gamma$. The span tension equation then becomes

$$L_n\frac{dt_n(t)}{dt} = v_{n+1}(E_nA_n - t_n(t)) - v_n\left(E_nA_n - \frac{E_nA_n}{E_{n-1}A_{n-1}}t_{n-1}(t)\right) + (E_nA_n - t_n(t))\frac{\mathrm{d}L_n}{\mathrm{d}t} \tag{2.29}$$

where $\frac{\mathrm{d}L_n}{\mathrm{d}t}$ is included to account for the pendulum motion causing span length changes.

Also, the input torque in the dancer system is caused by an air cylinder pushing on the pendulum arm in opposition to the spans' tensions. The air cylinder is allowed to pivot on both ends so that the line of force application to the pendulum is not always perpendicular. This has the effect of causing the applied torque to decrease as the pendulum swings away from the angle $\gamma$ where the air cylinder is perpendicular to the pendulum arm. This is modeled with a cosine function multiplying the force created by the air cylinder and the moment arm. The difficulty is describing the angle for the cosine function to use in terms of $\gamma$ and other constants as shown in Figure 2.8. The angle is

$$\phi = \gamma + \tan^{-1}\left(\frac{d(1-\cos\gamma)}{c - d\sin\gamma}\right) \tag{2.30}$$

where $\gamma$ is the pendulum angle, $c$ is the pendulum arm length, and $d$ is the perpendicular length of the air cylinder to its point of rotation (from the pivot of the air cylinder to the pendulum arm).

In summary, the pendulum dancer primitive element (system) includes a function for span length and its derivative, two span tension equations, each including a term for changing length, a roller torque balance, and a torque balance on the pendulum arm including the tensions from the spans, the input torque applied to the dancer with its reduction due to angle of swing, and the restorative moment caused by the mass of the pendulum.

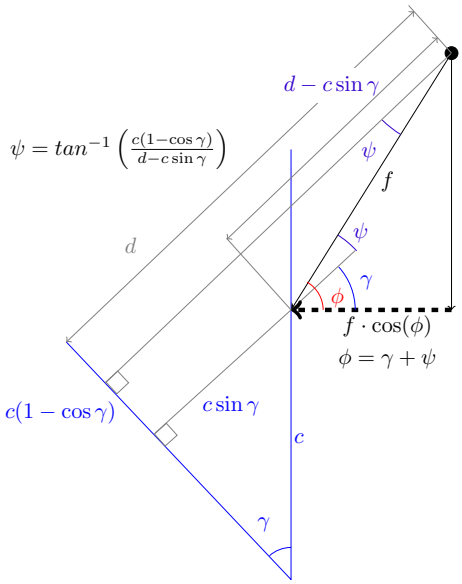



Figure 2.8: Defining the Angle for the Torque Component of the Force from the Air Cylinder, $\phi$. In the above figure, the angle $\gamma$ is the pendulum arm angle so the figure is up-side-down as it would appear in the actual web line.

### 2.2.4 The S-Wrap Dancer Primitive Element

The S-wrap dancer primitive element is a derivative of the pendulum dancer and models a device used to attenuate tension disturbances in the rewind section of the HSWL. The end view shown in Figure 2.10 with the fixture, point of rotation (at center), and applied external force shown. The S-wrap dancer is a subsystem with a rotating fixture, two idle rollers, and three spans. The subsystem is governed by the following set of equations.

$$J_{pn}\frac{\mathrm{d}^2\gamma}{\mathrm{d}t^2} = 2f_{qn}L_{qn}\cos(\gamma) - (t_{n-1}+t_{n+1})\,L_{sn}(\gamma)\cos(\gamma) - C_{bn}\dot{\gamma} - K_{sn}\gamma \tag{2.31}$$

$$\frac{\mathrm{d}\gamma}{\mathrm{d}t} = \dot{\gamma} \tag{2.32}$$

$$\begin{aligned} L_{n-1}\frac{\mathrm{d}t_{n-1}}{\mathrm{d}t} &= v_n(E_nA_n - t_{n-1}(t)) - v_{n-1}\left(E_{n-1}A_{n-1} - \frac{E_{n-1}A_{n-1}}{E_{n-2}A_{n-2}}t_{n-2}(t)\right) \\ &\quad + (E_{n-1}A_{n-1} - t_{n-1}(t))\frac{\mathrm{d}L_{n-1}}{\mathrm{d}t} \end{aligned} \tag{2.33}$$

$$L_n\frac{\mathrm{d}t_n}{\mathrm{d}t} = v_{n+1}(E_nA_n - t_n(t)) - v_n\left(E_nA_n - \frac{E_nA_n}{E_{n-1}A_{n-1}}t_{n-1}(t)\right) \tag{2.34}$$

$$\begin{aligned} L_{n+1}\frac{\mathrm{d}t_{n+1}}{\mathrm{d}t} &= v_{n+2}(E_{n+1}A_{n+1} - t_{n+1}(t)) - v_{n+1}\left(E_{n+1}A_{n+1} - \frac{E_{n+1}A_{n+1}}{E_nA_n}t_n(t)\right) \\ &\quad + (E_{n+1}A_{n+1} - t_{n+1}(t))\frac{\mathrm{d}L_{n+1}}{\mathrm{d}t} \end{aligned} \tag{2.35}$$

$$J_{pn} = J_{fix} + 2\left(J_{endcap} + m_{endcap}(L_{en} + L_{ecwidth})^2\right) + 2\left(J_n + m_n L_{en}^2\right) \tag{2.36}$$

$$L_{sn}(\gamma) = L_{en} + R_n \cos(\gamma) \tag{2.37}$$

As with other dancer models, the angle (movement) from the vertical is measured as positive if it is in the direction away from the spans. Figure 2.11 shows $\gamma$ measured positively. The rotation of the fixture causes either lengthening or shortening of spans $n-1$ and $n+1$. Span $n$ is a short span, but its length does not change due to rotation of the fixture. Constants $D_{n-1}$ and $D_{n+1}$ will be defined first. The constants are depicted in Figure 2.9. A similar picture could be drawn for the situation where the rollers in question both rotated the same direction.

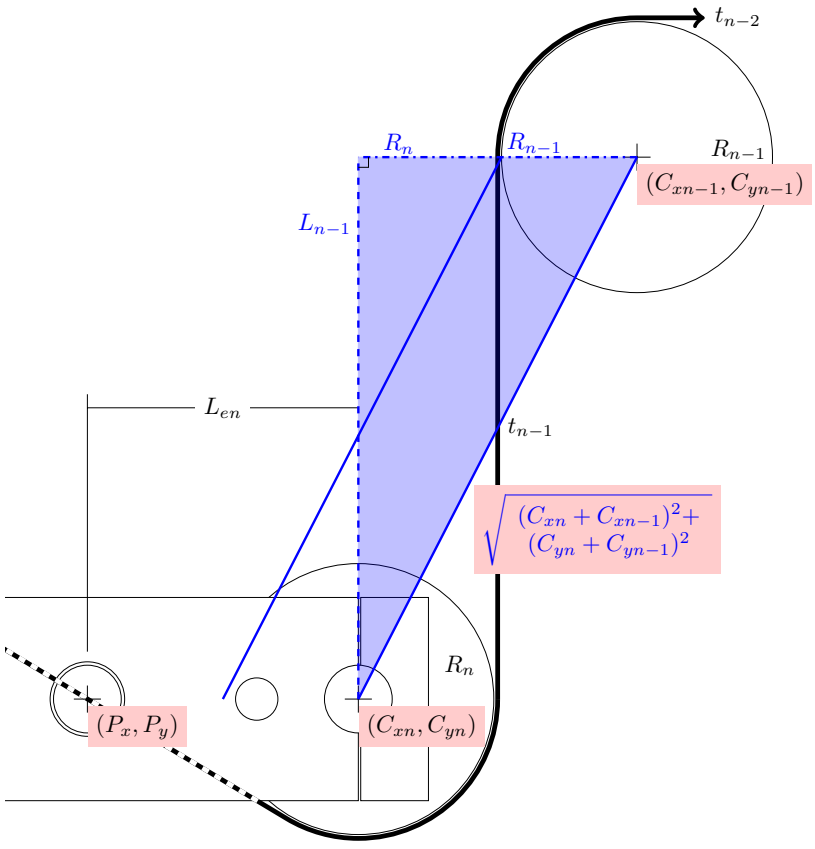



Figure 2.9: The Length of span $L_{n-1}$ is Calculated Using Lengths Defined in the Figure. The values of $C_{xn}$ and $C_{yn}$ are dependent on $\gamma$. $C_{xn} = P_x + L_{en}\cos(\gamma)$ and $C_{yn} = P_y - L_{en}\sin(\gamma)$

$$D_{n-1} = C_{xn-1}^2 - 2C_{xn-1}P_x + P_x^2 + C_{yn-1}^2 + \\ - 2C_{yn-1}P_y + P_y^2 + L_{en}^2 - (R_{n-1} + R_n)^2 \tag{2.38}$$

$$D_{n+1} = C_{xn+2}^2 - 2C_{xn+2}P_x + P_x^2 + C_{yn+2}^2 + \\ - 2C_{yn+2}P_y + P_y^2 + L_{en}^2 - (R_{n+2} + R_{n+1})^2 \tag{2.39}$$

$$L_{n-1} = \sqrt{D_{n-1} + 2\left(C_{xn-1} - P_x\right) L_{en}\cos(\gamma) + 2\left(C_{yn-1} - P_y\right) L_{en}\sin(\gamma)} \tag{2.40}$$

$$L_{n+1} = \sqrt{D_{n+1} + 2\left(-C_{xn+2} + P_x\right) L_{en}\cos(\gamma) + 2\left(-C_{yn+2} + P_y\right) L_{en}\sin(\gamma)} \tag{2.41}$$

$$\frac{\mathrm{d}L_{n-1}}{\mathrm{d}t} = \frac{1}{2}\frac{\left[2\left(C_{xn-1} - P_x\right) L_{en}\sin(\gamma) - 2\left(C_{yn-1} - P_y\right) L_{en}\cos(\gamma)\right]\dot{\gamma}}{\sqrt{D_{n-1} + 2\left(C_{xn-1} - P_x\right) L_{en}\cos(\gamma) + 2\left(C_{yn-1} - P_y\right) L_{en}\sin(\gamma)}} \tag{2.42}$$

$$\frac{\mathrm{d}L_{n+1}}{\mathrm{d}t} = \frac{1}{2}\frac{\left[2\left(-C_{xn+2} + P_x\right) L_{en}\sin(\gamma) - 2\left(-C_{yn+2} + P_y\right) L_{en}\cos(\gamma)\right]\dot{\gamma}}{\sqrt{D_{n+1} + 2\left(-C_{xn+2} + P_x\right) L_{en}\cos(\gamma) + 2\left(-C_{yn+2} + P_y\right) L_{en}\sin(\gamma)}} \tag{2.43}$$

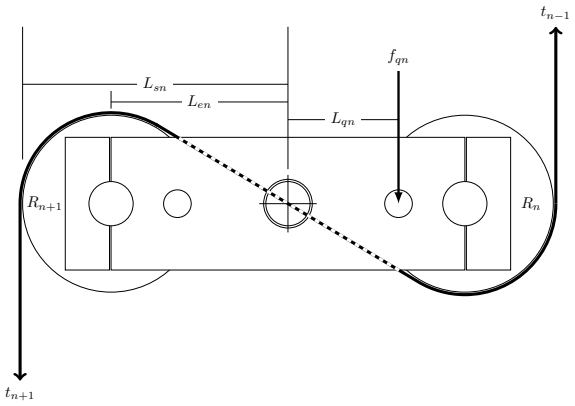



Figure 2.10: End View of the S-Wrap Dancer with Web Path and Applied External Force.

The situation can be indexed over to the span following the S-Wrap Dancer. The tension in the spans leading and following the S-Wrap Dancer can be described with equations (2.33) and (2.35).

### 2.2.5 Impact of Nonlinear Equations in Simulating Web Lines

As an example, a bump was created on the EWL unwind by inserting a steel bar into the roll and wrapping it in web. The extents of the bump were measured and experimental data was recorded.

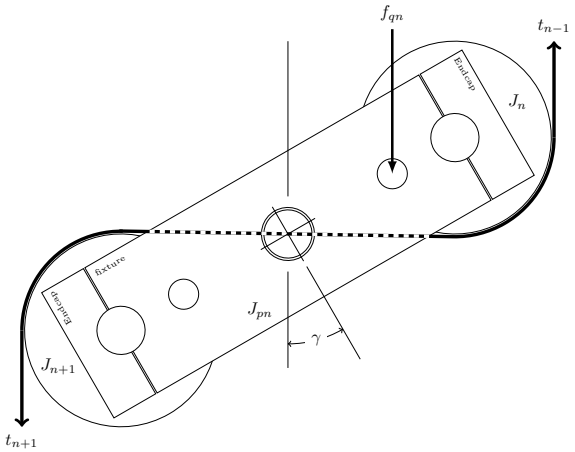



Figure 2.11: The Rotation Angle of the S-Wrap Dancer is $\gamma$. In the figure, $\gamma = -30°$, indicating that positive rotation is clockwise.

Then the bump was simulated on a model of the EWL that had linearized primitive elements and on a model that had nonlinear primitive elements. The recorded simulations were then plotted for comparison against the data recorded from the Euclid web line. The dancer position is shown in Figure 2.12 comparing the linear, nonlinear, and recorded data. The linear simulation barely moves in response to the unwind bump while the nonlinear simulation has an offset from zero, but does show the same frequency as the recorded data, but not the magnitude. This could indicate too much damping in the simulation. The speed of the web entering the first span has far more drastic changes in the nonlinear simulation than in the linear one as shown in Figure 2.13. Figure 2.14 shows the comparison of nonlinear and linear simulations and recorded data for a load cell tension. The simulated load cell is calculated by averaging span tensions 8 and 9 together on the EWL. Here again, the nonlinear simulation is agreeing with the recorded data far better than the linear model is. This same information is shown in Figure 2.15 but zoomed in on 1 second worth of data to show detail. The linear simulation really shows that it is not keeping up with the recorded data. The nonlinear simulation does a better job, but it is obvious that even with the nonlinear equations, the recorded data has more frequency content than the simulation does. The FFT of the data sets is shown in Figure 2.16 and does indicate that the recorded data has a couple of peaks that are not accounted for in the simulations at just over 3 Hz, 12 Hz, and the two large peaks at 39 Hz and 60 Hz. The last two peaks are most likely due to the tension transducer for the load cell. The data suggests it is not well isolated from the 60-cycle power and its documentation states it has a built

in filter which is nominally set at 40 Hz.

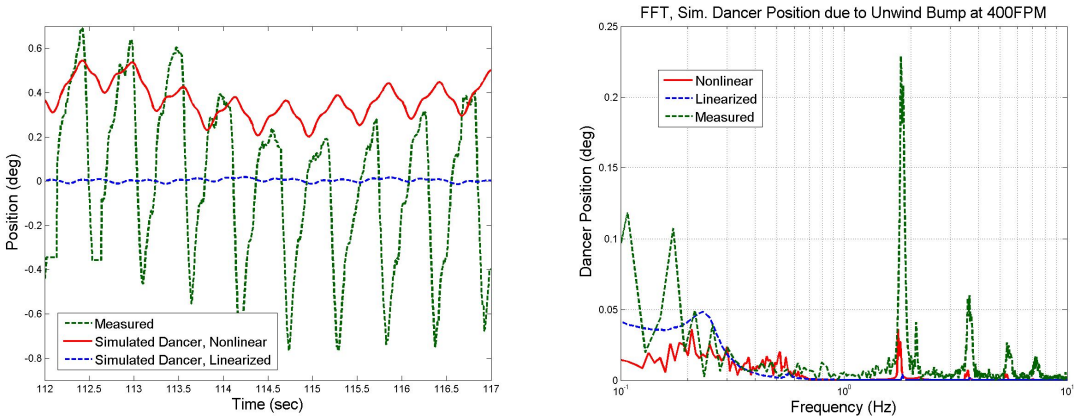



Figure 2.12: Comparing Dancer Position from Linear, Nonlinear, and Recorded Data. The time domain is left and the FFT is right. The linear simulation barely notes the 1-per-rev frequency while the nonlinear simulation picks up the 1-per-rev and a couple of the harmonics.

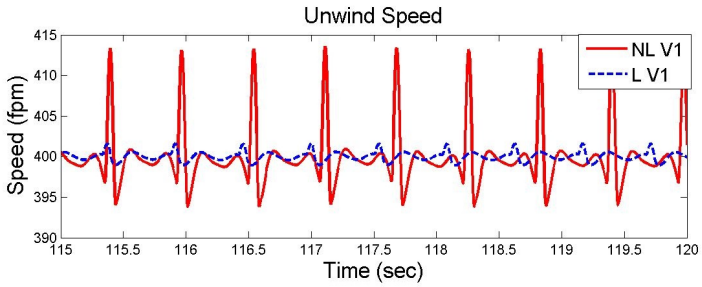



Figure 2.13: The Speed of the Web Entering the First Span, Nonlinear (red) and Linear (blue), Are Quite Different at the Bump But Similar in the Other Part of the Roll (top).

In this section, the EWL was simulated with nonlinear equations and compared to the simulation using linearized equations and to recorded data. The dancer showed a lower magnitude of oscillation than the recorded data, but a far larger oscillation than the linearized equation simulation did. The nonlinear simulation did capture the driving frequency and a couple of the harmonics while the linearized simulation barely showed a peak at the driving frequency. The differences between the dancer recorded position and the simulations suggest a sensitivity study on the dancer's arm damping and mass. The unwind speed showed marked differences between linearized and nonlinear simulations. Since span tension is dependent on the speed of the web entering the span, the difference in unwind speeds between the two simulations would indicate that the linearized simulation will have smaller tension excursions than the nonlinear one will

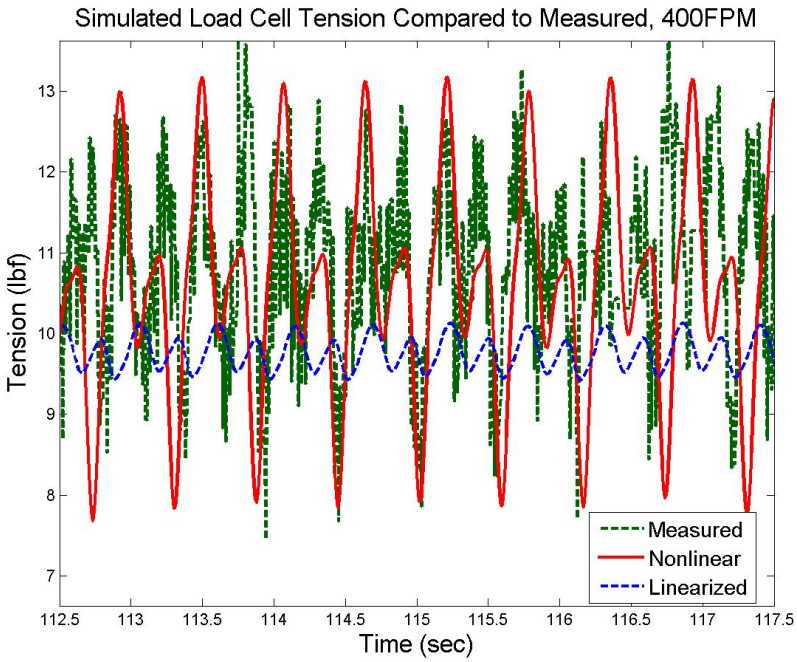



Figure 2.14: Load Cell Tension from Recorded (measured) Data, Nonlinear, and Linear Simulations. Note the way the nonlinear simulation generally agrees with the recorded data while the linear simulation does not.

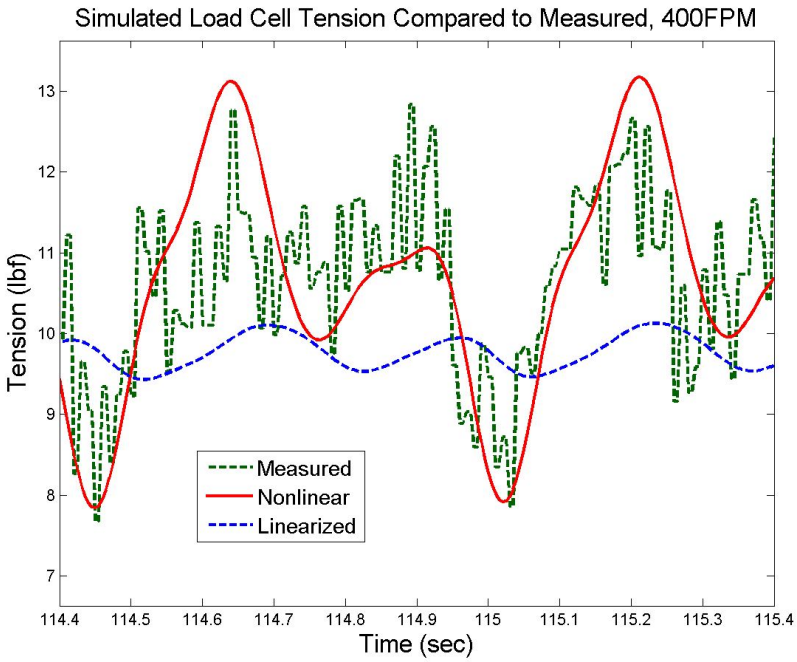



Figure 2.15: Load Cell Tension Comparison Zoomed in on 1 Second. The nonlinear simulation captures more of the tension dynamic than the linearized simulation does.

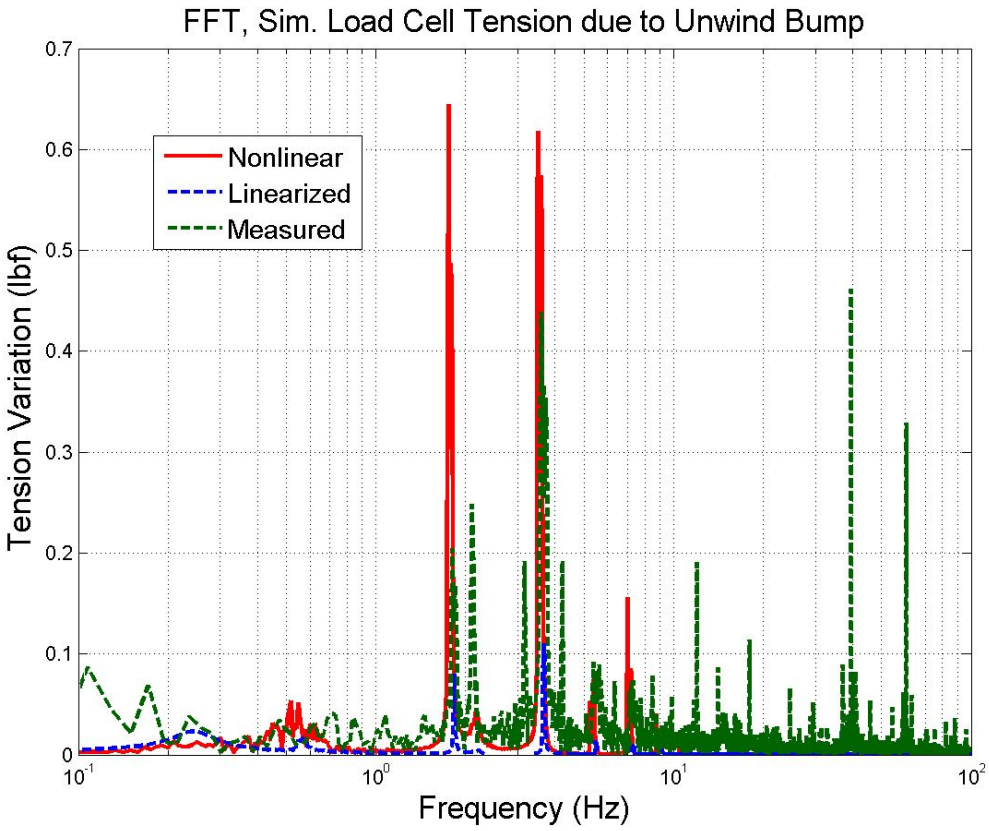



Figure 2.16: Fast Fourier Transform of Load Cell Tension from Nonlinear and Linear Simulations and Recorded Data.

have. The load cell tension simulation from the nonlinear model shows the same general trends as the recorded data while the linearized simulation load cell falls short. Looking at the FFT, the nonlinear simulation does not catch all the frequency content that the recorded data has and it over emphasizes certain frequencies more than the recorded data does.

# CHAPTER III

# A SYSTEMATIC METHOD FOR DETERMINING CONTROLLER GAINS IN A MULTI-SPAN WEB LINE

Web material has to travel through multiple tension zones in order to be unwound from the parent roll, processed once or more times, and finished. Finishing could include cutting to length or winding onto another roll for storage until it is used in another process. In each section the web travels through, speed and tension are monitored because excessive tension changes may cause blemishes in the web material or rupture and loss. The defense against loss is controlling tension in the web. In industrial settings, Proportional plus Integral (PI) controllers are used extensively. There are two gains per control loop that have to be determined.

A systematic method for determining controller gains has a process and equations provided such that the need for an individual user to have much experience is minimized. This method differs from the method proposed by Boulter in [58–60] by focusing on the time domain characteristics of the controlled response and not the frequency characteristics. Time domain characteristics are preferable because the impact of the parameter can be seen in real time. Also, linear and nonlinear models can be treated in the time domain. The time needed for the response to rise to the reference after a step input is called the "rise time," ($t_r$) and the damping ratio ($\zeta$) is an indicator of how quickly oscillations in the response will die out. These two parameters of second-order systems will govern the determination of gains in described method and are used as performance goals. Web lines are not simple, second-order systems in reality, but in order to determine gains,

assumptions and simplifications have to be made.

The concept of the Routh Approximation Method for gain calculation is described in the following steps:

I. Create transfer functions from primitive elements for individual components of the web line beginning with where force is applied, i.e. at the motor

II. Determine a transfer function for the open-loop path of the controller by compiling primitive elements together

III. Use Hutton's Routh Approximation Method to reduce the open-loop transfer function if it is of higher order than second-order

IV. Use second-order dynamic system performance goals to determine gains for the closed-loop system

The steps will be iterated through as needed, depending on how many loops a specific motor has. As a reminder, a speed controlled motor only has one control loop while a speed-base tension controlled motor has two loops.

## 3.1 Performance Goals

Parameters of a second-order response whose effects are immediately apparent to the user are damping ratio, $\zeta$, rise time, $t_r$, and steady-state error. The damping ratio defines if a perturbed system does not oscillate ($\zeta \geq 1.0$) or oscillates ($0 \leq \zeta < 1.0$) and how quickly the oscillation dies out. A family of time domain responses of a second-order system is shown in Figure 3.1 (a). The damping coefficient can be calculated from the percent overshoot ($PO$), if that is more meaningful, by

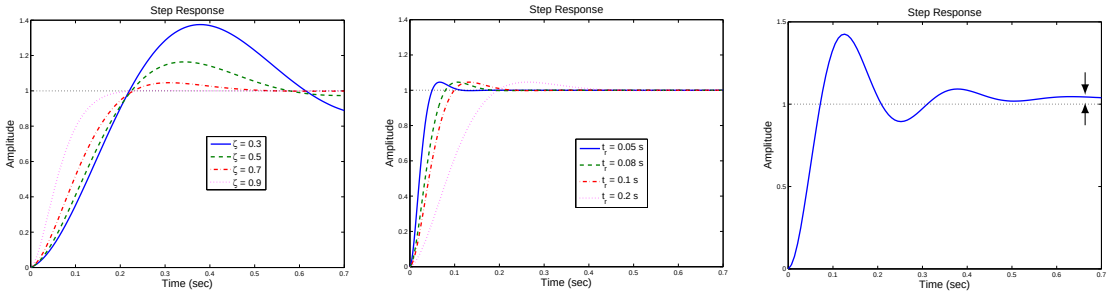
$$\zeta = \sqrt{\frac{\left[\ln\left(\frac{PO}{100}\right)\right]^2}{\pi^2 + \left[\ln\left(\frac{PO}{100}\right)\right]^2}} \tag{3.1}$$

where $\zeta$ is the damping ratio and $PO$ is the percent overshoot [65]. Rise time is defined as the amount of time required for the output of the system to rise from 0% to 100% of its final value

(Palm also notes that some writers define rise time as the time for the step response to go from 10% to 90% of the final value) [65]. A family of second-order system responses with varying rise times is shown in Figure 3.1 (b). For values of $\zeta$ such that $0.1 \leq \zeta \leq 0.9$, the natural frequency, $\omega_{nat}$, of a second-order system is related to rise time and damping ratio by

$$\omega_{nat} = \frac{\pi - \tan^{-1}\left(\sqrt{\frac{(1-\zeta^2)}{\zeta}}\right)}{t_r\sqrt{(1-\zeta^2)}} \tag{3.2}$$

where $\omega_{nat}$ is the natural frequency of the system in rad/sec, and $t_r$ is the rise time in seconds. For values of $\zeta > 0.9$, the inverse tangent becomes asymptotic. All values from experiments on the Euclid Web Line (EWL) and High-Speed Web Line (HSWL) are listed in Table 3.1 using (3.2) to calculate the natural frequency. The last five rows are for the HSWL. The natural frequencies are lower than for a comparable rise time with a higher damping coefficient.




(a) (b) (c)

Figure 3.1: Plots Depicting the Effects of Varying Second-Order Performance Goals and a Definition of Steady-State Error. Damping ratio (a) is varied with constant rise time, $t_r = 0.235$, rise time (b) is varied with constant damping ratio, $\zeta = 0.7$, and a steady-state error (c) is shown as the gap between the arrows.

Table 3.1: Damping Coefficients and Calculated Natural Frequencies Used in Experiments on the Euclid and High-Speed Web Lines

| Damping, $\zeta$ | Rise Time, $t_r$ (sec) | Natural Frequency, $\omega_n$ ($\frac{rad}{s}$) |
|---|---|---|
| 0.9 | 0.3 | 20.7 |
| 0.9 | 0.15 | 41.5 |
| 0.7797 | 0.3 | 13.9 |
| 0.7797 | 0.15 | 26.2 |
| 0.7797 | 0.1 | 39.4 |
| 0.9 | 0.239 | 26.0 |
| 0.7797 | 0.259 | 16.284 |

A third performance goal is steady-state tension error. Figure 3.1 (c) shows a second-order

system with a steady-state error. The steady-state tension error applies to tension feedback controllers and not speed controllers. The error can be introduced because of the assumptions and simplifications used to develop controller gains.

## 3.2 Modeling Needed for Gain Calculations

Web lines being modeled in this section have an unwind, free spans, idlers, maybe a dancer, and a rewind. An unwind is a driven roller and therefore has a motor. A motor is modeled as a first-order linear differential equation with constant parameters as shown in (3.3). This differential equation is from the roll's perspective and assumes it is a DC motor.

$$J_n\dot{v}_n = -\left(B_{fn} + \frac{C_{mn}}{\mathsf{g}_{\mathsf{r}n}}\right)v_n + \frac{R_n K_{mn}}{\mathsf{g}_{\mathsf{r}n}}u_n \pm R_n^2 T_n \tag{3.3}$$

The $\pm$ sign is decided by whether or not the motor is unwinding or rewinding. If the motor is rewinding, then negative; otherwise it is positive. The $\mathsf{g}_{\mathsf{r}n}$ is the gear ratio defined as the number of spindle revolutions per motor revolution. The Laplace transform of (3.3) leads to the open-loop transfer function relating input current, $u_n$, to tangential roll speed, $v_n$, given in (3.4).

$$G_1(s) = \frac{V_n(s)}{U_n(s)} = \frac{\frac{R_n K_{mn}}{J_n \mathsf{g}_{\mathsf{r}n}}}{s + \frac{(B_{fn}\mathsf{g}_{\mathsf{r}n} + C_{mn})}{J_n \mathsf{g}_{\mathsf{r}n}}} \tag{3.4}$$

The AC motor is modeled with a similar differential equation:

$$J_n\dot{\omega}_n = -\left(B_{fn}\mathsf{g}_{\mathsf{r}n} + C_{mn}\right)\omega_n + \tau_n \pm R_n T_n \mathsf{g}_{\mathsf{r}n} \tag{3.5}$$

where $\tau_n$ is the torque supplied by the motor. More detail is given in section 4.2 about modeling AC motors. The Laplace Transform relating motor torque to motor speed is

$$G_1(s) = \frac{\Omega_n(s)}{\mathsf{T}_n(s)} = \frac{\frac{1}{J_n}}{s + \frac{(B_{fn}\mathsf{g}_{\mathsf{r}n} + C_{mn})}{J_n}} \tag{3.6}$$

where $\mathsf{T}_n(s)$ is the Laplace Transform of $\tau_n$.

A physical model of the free span of the web is shown in Figure 3.2. It is assumed that the web is linear-elastic, has a constant cross sectional area, undergoes small strains, and does not slip on the idle rollers. With these assumptions from [20], the following nonlinear differential equation

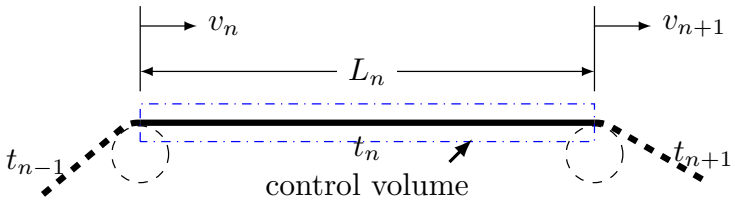



Figure 3.2: The Free Span is the Material Between the Points of Contact with the End Rollers that Experiences Tension and Velocity.

results from a combination of the law of conservation of mass for the control volume and the application of Hooke's Law:

$$L_n \frac{\mathrm{d}t_n}{\mathrm{d}t} = EA\left(v_{n+1} - v_n\right) + \left(v_n t_{n-1} - v_{n+1} t_n\right) \tag{3.7}$$

where $t_n$ is the tension in the span of interest, $v_n$ is the roller speed of the roller at the entering end of the span, $v_{n+1}$ is the speed at the outgoing end of the span, and $t_{n-1}$ is the tension in the previous span. Equation (3.7) is nonlinear because of the multiplication of roller speed and span tension. This equation can be linearized if it is assumed that all variables ($v_n$ and $v_{n+1}$ ) undergo small changes about initial steady-state values. The result is

$$L_n \frac{\mathrm{d}t_n}{\mathrm{d}t} = EA\left(v_{n+1} - v_n\right) + \left(V_{n,0} t_{n-1} - V_{n+1,0} t_n\right) \tag{3.8}$$

where $V_{n,0}$ and $V_{n+1,0}$ are the steady-state speeds of the incoming and outgoing rollers. Equation (3.8) may be used to demonstrate behavior of the span tension in the steady-state operation (i.e., ($\frac{\mathrm{d}T_n}{\mathrm{d}t} = 0$). The tension in a span is dependent on the difference in the velocities at the ends of the web, and that tension is transferred from an upstream span to a downstream span.

The Laplace transform of (3.8) leads to the open-loop transfer function relating the web speed at the incoming roller to web span tension given in (3.9).

$$\frac{T_n(s)}{V_n(s)} = \frac{\pm\frac{EA}{L_n}}{s + \frac{V_{n+1,0}}{L_n}} \tag{3.9}$$

The negative sign is for the situation where the driven roll is upstream of span $n$. In the case of a rewind motor where the driven roll is downstream of the span, there is no negative sign. Driven rolls in the middle of the web line can be either positive or negative because the tension feedback could come from either upstream or downstream of the driven roll. In fact, it is advantageous

to treat all driven rolls as rewinds when linking a tension feedback to them because there are no negative signs in the Laplace Transform model being built. The effect of the negative sign can be properly taken care of in the application of the calculated trim value in the control loop.

A dancer (see Figure 3.3) is modeled with summation of moments about the dancer pivot with the dancer angle, $\gamma$, increasing away from the spans. The differential equation is

$$\frac{\mathrm{d}\dot{\gamma}}{\mathrm{d}t} = \frac{f_{qn}}{J_{pn}} - t_n \frac{L_{an} - R_n}{J_{pn}} - t_{n-1} \frac{L_{an} + R_n}{J_{pn}} - \frac{C_{pn}}{J_{pn}}\dot{\gamma} - \frac{K_{pn}}{J_{pn}}\gamma - \frac{m_p g l_{cg}}{J_{pn}} \sin\gamma \tag{3.10}$$

where $K_{pn}$ is the spring constant of the dancer and $C_{pn}$ is the damping in the pivot of the dancer arm and torque application system, $m_p$ is the mass of the pendulum dancer, $l_{cg}$ is the distance from the pivot to the center of gravity, and $g$ is the gravitational constant. This equation can be linearized if it is assumed that the input torque ($f_{qn}$), outgoing span tension ($t_n$), and the initial conditions for the dancer position are all zero, and that the small angle approximation applies ($\sin\gamma \approx \gamma$). With these assumptions, the Laplace Transform of the linearized version of (3.10) leads to an open-loop transfer function relating the tension in the incoming span ($T_{n-1}(s)$) to the dancer position ($\Gamma(s)$).

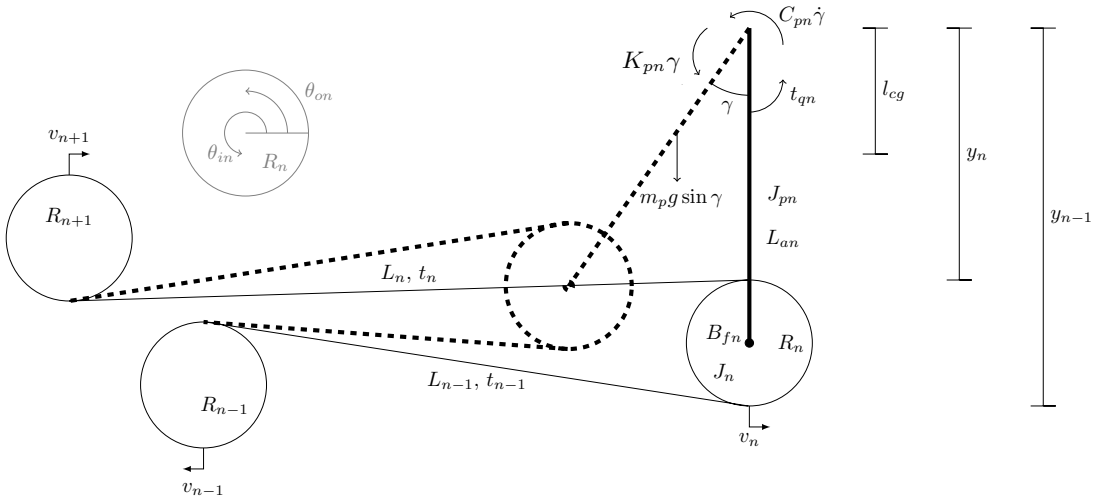



Figure 3.3: A Physical Model of Dancer Primitive Element Showing Variables and Physical Parameters

$$\frac{\Gamma(s)}{T_{n-1}(s)} = \frac{\frac{(L_{an}+R_n)}{J_{pn}}}{s^2 + \frac{C_{pn}}{J_{pn}}s + \frac{(K_{pn}+(m_p g l_{cg}))}{J_{pn}}} \tag{3.11}$$

Equation (3.11) includes the restorative moment due to gravity ($m_p g l_{cg} \sin\gamma$) which ensures that the dancer system is not a pure integrator if there is no damping or spring constant assumed in

the system. In the case of a translational dancer, a spring constant or some amount of damping in the dancer needs to be modeled or the model will reduce to a double integrator which will fail the approximation method.

### 3.2.1 Model Order Reduction

Hutton demonstrated a Routh Approximation (RA) method for reducing at least a third-order transfer function with a lower-order transfer function that retains the same initial response and stability of the original transfer function in [69, 70]. Hutton used a continued fraction form of the polynomial, along with truncation, to reduce the order of the polynomial with the least error. Hutton realized that the continued fraction coefficients could be calculated from the Routh Array. The Routh Array was originally described in [71], but is now in many textbooks, e.g. [21], as a method to check large polynomials for stability without having to find the roots of the polynomial. Figure 3.4 shows the Bode plot of the third-order transfer function created to determine controller gains for the first 6 inch wide Tyvek experiment and its second-order approximation. The metric Hutton showed in [70] for calculating the impulse energy shows that 96% of the energy from the original transfer function is in the second-order approximant. The Bode plots for the other two materials

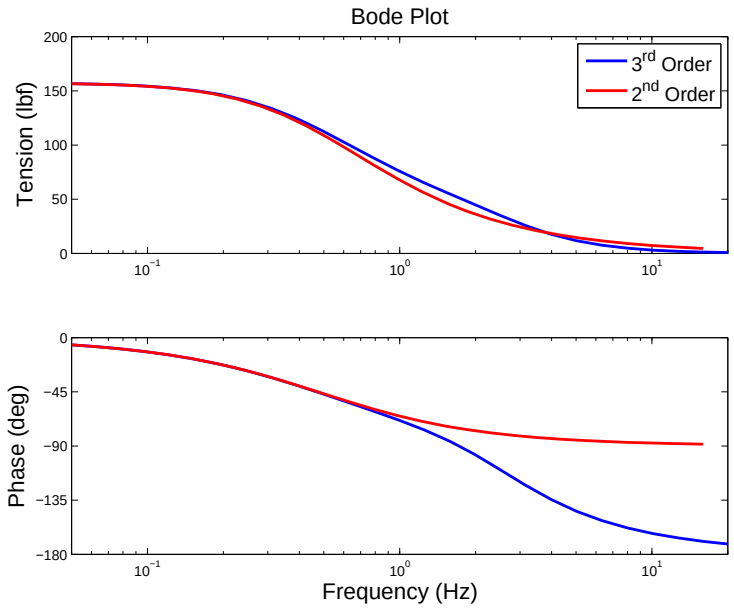



Figure 3.4: Bode Plot of a 6in Tyvek Controller Model Third-Order Transfer Function (blue) and Its Second-Order Approximation (red) Obtained Using Hutton's Routh Approximation Method. The approximate transfer function follows the third-order transfer function well in magnitude.

are in Appendix I.8.4 and have the same shape, but scaled larger. The algorithm from [69, 70] was implemented in MATLAB and the code is also in Appendix I.8.4. The examples used by Hutton were used to evaluate the MATLAB code.

## 3.3 The Routh Approximation Gain Calculation Method

The previous section showed how to model two kinds of motors, a span, and a dancer. Hutton's Routh Approximation Method for reducing polynomial order was introduced. Building a control loop transfer function is next.

The S-wrap and pull roll motors of the EWL are under pure speed control, and the unwind and rewind rolls are under speed-based tension control. On the HSWL, a similar situation exists: the unwind and rewind motors are under speed-based tension control while the pull roll is under pure speed control. Since the method requires and there is a speed control loop in both cases, the proportional and integral gains for the speed loops are found first. Then the gains for the tension loops are found.

### 3.3.1 Speed Control

There are five speed controllers in the Euclid Web line[1] and three speed controllers on the HSWL. Equation (3.4) can be used as the open-loop transfer function, $G_1(s)$ in Figure 3.5, for each speed controller on the EWL and (3.6) for each speed controller on the HSWL. A PI speed controller is placed in series with transfer function, (3.4) or (3.6), and the loop is closed. The output speed of the motor is compared with the speed reference. The block diagram in Figure 3.5 shows the speed loop where the speed feedback is the motor shaft speed.

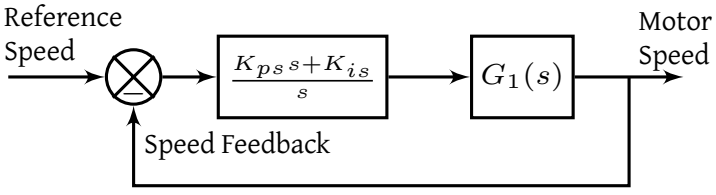



Figure 3.5: The Closed-Loop Block Diagram relating the Speed Reference to the Motor Speed with PI Control.

[1] The results of the RA method applied to the Euclid Web Line are recorded in Appendix B

The closed-loop transfer function relating the motor speed reference to the motor speed is

$$\frac{\Omega_n(s)}{\Omega_{ref}(s)} = \frac{\frac{K_{mn}}{J_n}(K_{ps}s + K_{is})}{s^2 + \left(\frac{(B_{fn}+C_{mn})}{J_n} + \frac{K_{mn}}{J_n}K_{ps}\right)s + \frac{K_{mn}}{J_n}K_{is}} \tag{3.12}$$

$$\frac{\Omega_n(s)}{\Omega_{ref}(s)} = \frac{\frac{1}{J_n}(K_{ps}s + K_{is})}{s^2 + \left(\frac{(B_{fn}+C_{mn})}{J_n} + \frac{1}{J_n}K_{ps}\right)s + \frac{1}{J_n}K_{is}} \tag{3.13}$$

where (3.12) is for DC motors and (3.13) is for AC motors. The transfer functions can be represented in terms of a natural frequency and damping ratio:

$$\frac{\Omega_n(s)}{\Omega_{ref}(s)} = \frac{\frac{K_{mn}}{J_n}(K_{ps}s + K_{is})}{s^2 + 2\zeta\omega_{nat}s + \omega_{nat}^2} \tag{3.14}$$

$$\frac{\Omega_n(s)}{\Omega_{ref}(s)} = \frac{\frac{1}{J_n}(K_{ps}s + K_{is})}{s^2 + 2\zeta\omega_{nat}s + \omega_{nat}^2} \tag{3.15}$$

where (3.14) is for DC motors and (3.15) is for AC motors. Equations (3.12) and (3.13) define a second-order linear system. The gains are displayed in the closed-loop transfer function. The motor can be driven to a specific speed with the desired performance through the selection of a damping ratio, $\zeta$, and a rise time, $t_r$, which is step IV. Equation (3.2) can be used to calculate the natural frequency, $\omega_{nat}$. The PI controller gains for each speed control may be calculated from equations that result from comparing the characteristic equations (the denominators from (3.12) and (3.14)).

$$K_{is} = \frac{J_n\omega_{nat}^2}{K_{mn}} \tag{3.16}$$

$$K_{ps} = \left[2\zeta\omega_n - \frac{(B_{fn}\mathrm{g_r} + C_{mn})}{J_n}\right]\frac{J_n}{K_{mn}} \tag{3.17}$$

where $K_{is}$ is the integral gain and $K_{ps}$ is the proportional gain for the speed control loop. The gains for an AC motor can be determined in a similar fashion using (3.13) and (3.15).

$$K_{is} = J_n\omega_{nat}^2 \tag{3.18}$$

$$K_{ps} = \left[2\zeta\omega_n - \frac{(B_{fn}\mathrm{g_r} + C_{mn})}{J_n}\right]J_n \tag{3.19}$$

The gains for each of the speed control loops can be found from these equations with the associated parameters inserted.

### 3.3.2 Tension or Position Control

Tension control with load cell feedback using a DC motor is shown as a block diagram in Figure 1.2, and Figure 1.3 shows the block diagram of the tension control with dancer position feedback. There are two sets of Proportional and Integral gains in each system. Determination of the PI gains for the tension loop is difficult because the open-loop system from reference speed to span tension is at least fourth-order in the case of the EWL if all rollers and spans are included. And after a PI controller is added, the closed-loop system relating motor speed to the span tension at the load cell is fifth-order.

The Routh Approximation Method requires that the speed loop be closed first, and then the closed-loop transfer function relating the reference speed of the motor to its output speed be determined. The transfer function is (3.12) or

$$G_s(s) = \frac{R_n\Omega_n(s)}{R_n\Omega_{ref}(s)} = \frac{V_n(s)}{V_{nref}(s)} = \frac{\frac{K_{ps}s+K_{is}}{s}G_1(s)}{1+\frac{K_{ps}s+K_{is}}{s}G_1(s)} \tag{3.20}$$

$G_1(s)$ refers to (3.4), the open-loop transfer function relating motor input current to motor speed or to (3.6), the open-loop transfer function relating motor torque to motor speed. $G_s(s)$ is the closed-loop transfer function relating motor speed reference to motor speed. Note that the output of the PI control is different in these two cases. The output of the PI controller when using (3.4) is in amps while the output of the PI controller when using (3.6) is in ft-lbf.

When load cell feedback is used to adjust the speed reference of a motor, the transfer function for the open-loop plant is the combination of (3.20) and the transfer function relating incoming tangential roller speed to span tension, (3.9), if the load cell is the first idle roller in the web line after the unwinding motor. Figure 3.6 shows the open-loop transfer function block diagram from reference speed to span tension. The $K_c$ is a conversion factor made up of the gear ratio and the unwind roll radius to convert from motor angular speed (or RPM) to web linear speed. The open-loop transfer function block diagram shown in Figure 3.6 is third-order if there is only one span between the unwind roll and the load cell. While a third-order transfer function is the ideal case, sometimes more spans are required due to space constraints in the layout of the line. If that happens, the order of the system increases by 2 with each additional idler and span.

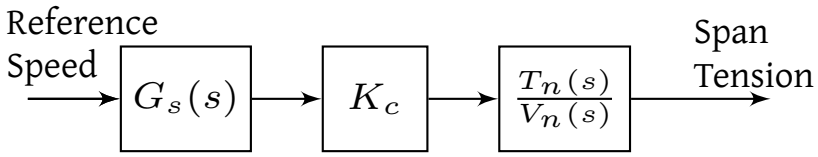



Figure 3.6: The Open-Loop Block Diagram from Reference Speed to Span Tension. The $K_c$ block is a conversion factor from the motor speed to web speed. The open-loop transfer function in the third block is given in (3.9) in terms of parameters.

The ideal open-loop transfer function, whose block diagram is shown in Figure 3.6, is third-order or higher, but this is not the case for the unwind section of the EWL. The minimum order would be fifth-order if the load cell is at roller 3, and would be higher order yet if the models for the idlers and spans between the unwind roll and load cell at roller 9 are included. However, the high order system can be reduced by assuming that the dissipative nature of the idlers can be ignored, and by treating all the spans from the unwind to the load cell at either roller 3 or 9 as one span with a length equal to the sum of the lengths of all the spans. Combining the spans into one reduces the order of the model to third-order for the open-loop load cell control. The transfer function block diagram shown in Figure 3.6 is called $G_2(s)$ once it is simplified. The transfer function has the form shown in (3.21).

$$G_2(s) = \frac{b_1 s^m + \cdots + b_{m+1}}{s^n + a_1 s^{n-1} + \cdots + a_n)} \tag{3.21}$$

When dancer position feedback is used to adjust the reference speed, the block diagram of the open-loop system is shown in Figure 3.7 where all the parts just discussed for load cell feedback exist, but one more transfer function is added. Equation (3.11) is the transfer function relating incoming span tension to dancer position. If the dancer roller is the first roller the web encounters after being unwound, the combined open-loop transfer function is fifth-order. If the dancer is not the first idler the web encounters, then the order of the system is increased by two per idler and span. Ignoring the dissipative effects of the additional idlers and summing the length of the spans together into one span will reduce the order back to fifth-order. The combined transfer function shown in Figure 3.7 as a block diagram can be simplified and put in the form of (3.21).

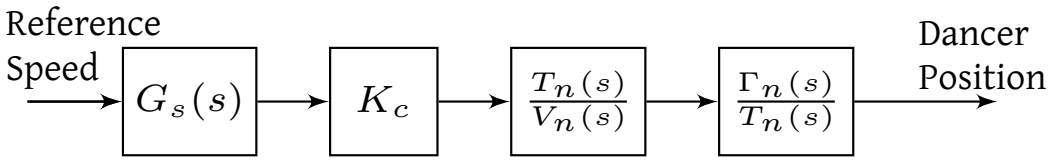



Figure 3.7: The Open-Loop Block Diagram from Reference Speed to Dancer Position. This whole transfer function becomes $G_2(s)$ once simplified.

### 3.3.3 Approximating the Transfer Function

Hutton's Routh Approximation (RA) method is used to reduce the transfer function, $G_2(s)$, from third-order in the case of load cell feedback or fifth-order for dancer position feedback to second-order as shown in Figure 3.8 which is step III. The second-order approximation is called $\widetilde{G}_2(s)$. With the open-loop approximated as $\widetilde{G}_2(s)$, the gains for the PI controller in the closed-loop system (3.23) can be found by assuming a real pole exists as is (3.24) and simultaneously solving equations (3.25). The closed-loop system block diagram is shown in Figure 3.9.

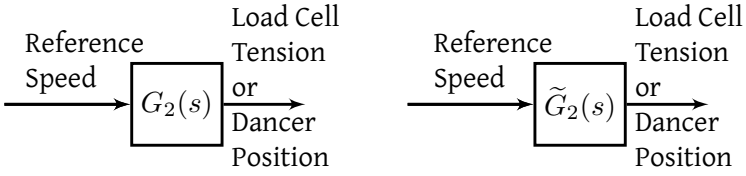



Figure 3.8: For Dancer Position or Load Cell Tension Feedback, Hutton's Routh Approximation Method Reduces the Open-Loop Transfer Function $G_2(s)$ to $\widetilde{G}_2(s)$.

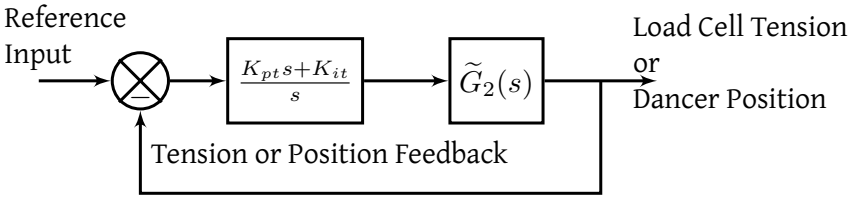



Figure 3.9: The Closed-Loop Block Diagram for Tension or Dancer Position Control Incorporating the Approximated Transfer Function from Hutton's RA Method.

The RA method will take in the open-loop tension transfer function, $G_2(s)$, of the modeled system as a numerator vector of coefficients, $[\ b_1 \quad b_2 \quad \cdots \quad b_{m+1}\ ]$ and a denominator vector of coefficients, $[\ 1 \quad a_1 \quad \cdots \quad a_n\ ]$ (see (3.21)). The RA method output is a set of two vectors[2], $M_n$ and $M_d$, which contain the coefficients of the approximated transfer function $\widetilde{G}_2(s)$. The form

[2]The value of $M_n(1) = 0$ and $M_d(1) = 1$ for the transfer functions described herein.

for a second-order approximation is

$$\widetilde{G}_2(s) = \frac{M_n(2)s + M_n(3)}{s^2 + M_d(2)s + M_d(3)} \tag{3.22}$$

which is then included in the closed-loop transfer function with the PI controller, $G_3(s)$,

$$G_3(s) = \frac{\left(\frac{K_{pt,i}s + K_{it,i}}{s}\right)\widetilde{G}_2(s)}{1 + \left(\frac{K_{pt,i}s + K_{it,i}}{s}\right)\widetilde{G}_2(s)} \tag{3.23}$$

and compared with the desired characteristic polynomial, $G_{3d}(s)$, below.

$$G_{3d}(s) = \frac{(K_{pt,i}s + K_{it,i})(M_n(2)s + M_n(3))}{(s + k_r\zeta\omega_{nat})(s^2 + 2\zeta\omega_{nat}s + \omega_{nat}^2)} \tag{3.24}$$

Once (3.23) is simplified, its coefficients may be equated to those of the desired transfer function in (3.24) which yields the following set of simultaneous equations, (3.25).

$$\begin{array}{rrcl} k_r\zeta\omega_{nat} & -K_{pt,i}M_n(2) & = & -2\zeta\omega_{nat} + M_d(2) \\ k_r\left(2\zeta^2\omega_{nat}^3 - \frac{M_n(2)}{M_n(3)}\zeta\omega_{nat}^3\right) & -K_{pt,i}M_n(3) & = & -\omega_{nat} + M_d(3) \end{array} \tag{3.25}$$

Solving for $K_{pt,i}$ and then $k_r$:

$$K_{pt,i} = \frac{-\omega_{nat} + M_d(3) + (2\zeta\omega_{nat} - M_d(2))\left(2\zeta\omega_{nat}^2 - \frac{M_n(2)}{M_n(3)}\omega_{nat}^2\right)}{M_n(2)\left(2\zeta\omega_{nat}^2 - \frac{M_n(2)}{M_n(3)}\omega_{nat}^2\right) - M_n(3)} \tag{3.26}$$

$$k_r = \frac{K_{pt,i}M_n(2) - 2\zeta\omega_{nat} + M_d(2)}{\zeta\omega_{nat}} \tag{3.27}$$

Once $k_r$ is known, the integral gain is found by

$$K_{it,i} = \frac{k_r\omega_{nat}^3\zeta}{M_n(3)}. \tag{3.28}$$

If $k_r$ is large enough, the effect of the associated pole is small compared to that for the two complex roots [21]. But in the case of applying the method to the Euclid Line, the magnitude of $k_r$ has generally been small and the effect is not small. The numerator constant, $M_n(3)$, from the approximated transfer function, $\widetilde{G}_2(s)$, is in the denominator of (3.28) which means that large magnitudes in the numerator will force small integral gains which can be problematic.

### 3.3.4 Scaling Factors

On the EWL, the RA method proportional gains had to be increased by a factor of 10 (though sometimes as low as 3, but the experimental data shown in Appendix B is with 10) to maintain tension in the spans. This was an experimental finding and arbitrary. On the HSWL scaling factors are needed as well, however, an explanation is given for each of the scaling factors.

On both the EWL and the HSWL, the Rockwell drives contain current and voltage shaping power electronics. The speed of the motor is controlled in the drive, not in the RSLogix Controller. The speed loop is executed on the drive every half millisecond compared to the RSLogix 5000® software which ran at 10 ms on the EWL and 20 ms on the HSWL. This is a peril of mixing discrete and continuous systems, and in this case, there are two discrete systems running at different rates. The analysis so far has been in the continuous time domain assuming that the sampling rates of the controllers was fast enough. Ogata showed in [72] that the discrete time operator, $z$, is related to the Laplace transform operator by

$$e^{Ts} = z \tag{3.29}$$

or

$$s = \frac{1}{T} \ln z \tag{3.30}$$

where $T$ is the sampling period. Then Ogata showed that the zero-order hold in the Laplace domain is

$$G_{0h}(s) = \frac{1 - e^{-Ts}}{s} \tag{3.31}$$

The block diagram of the speed-based tension controller including delays is shown in Figure 3.10. The $T_2$ is 0.02 seconds while the $T_1$ is 0.0005 seconds. Hong et. al show in [73] that a multi-rate tracking controller in the discrete time domain needs an undershoot quantity adjustment due to the low frequency of the outer loop. This controller operates at the slow speed. Then a conversion to the high speed inner loop is needed and, finally, another controller at the high speed. The authors of [73] draw all their theory from [74].

The analysis herein is in the continuous time domain. If a controller is converted from continuous time to discrete time using (3.30) at the slow frequency and then converted back to the

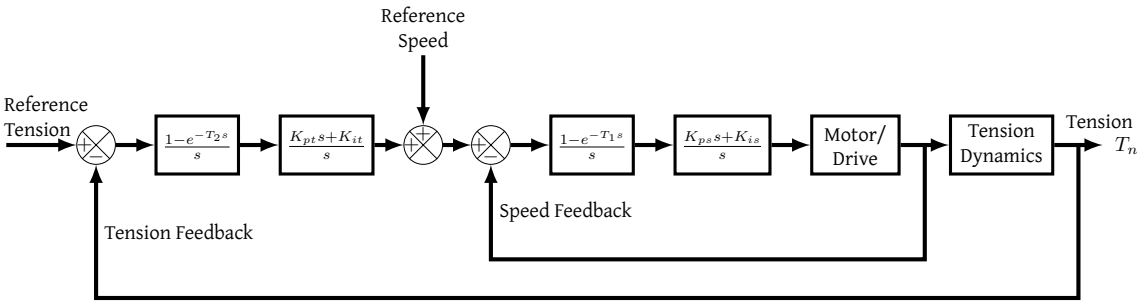



Figure 3.10: Speed-Based Tension Controller with Delays

continuous time domain at the fast frequency using (3.29), the result is (using the sampling periods from the HSWL):

$$s = \frac{1}{0.02s}\ln z = \frac{1}{0.02}\ln\left(e^{0.0005s}\right) \tag{3.32}$$

$$= \frac{1}{0.02}\left(0.0005s\right) \tag{3.33}$$

$$= \frac{0.0005s}{0.02} = \frac{1}{40}s \tag{3.34}$$

Based on (3.34), a scaling factor for the slow frequency controller, $k_{TLsr}$, should be equal to 40 to make up for the decrease from conversion to discrete time and back. Another way to look at the scaling factor $k_{TLsr}$ is for every execution of the tension loop, the speed loop executes $40$ times on the HSWL. The ratio of speed loop executions to tension loop executions was used as a scaling factor, $k_{TLsr}$, for the tension loop gains, e.g., $40$ speed loop executions per 1 tension loop execution works out to $k_{TLsr} = 40$. Scaling factor $k_{TLsr}$ would change based on the set rate of the RSLogix 5000® execution. The minimum for RSLogix is 2 ms which would set the ratio to $4$.

The second scaling factor used on the tension loop gains is a build-up ratio, $R_b$, which scales the gains based on the diameter or radius of the parent roll. This scaling factor uses the calculated roll diameter over the empty core diameter (3 in) or radius (1.5 in), quantity squared. Refer to (1.9) for an example of how a build-up ratio is implemented. The empty core diameter is whatever the spindle shaft diameter is on the line of interest. The parent roll diameter is calculated by the RSLogix program on a reoccurring basis. The diameter calculation integrates the rotating speed of the unwind motor to obtain position. It also integrates the rotating speed of the pull roll which has a fixed diameter. Assuming that the path length does not change and that the tangential speeds of both rolls have to be equal, the unwind roll diameter can be calculated. This process works well

after two or three start-ups in the case of the HSWL. A dedicated measurement of the roll diameter would not have this limitations.

The HSWL tension loop gains are determined by multiplying $K_{pt,i}$ and $K_{it,i}$ by $k_{TLsr}$and $R_b$. The initial values of $K_{pt,i}$ and $K_{it,i}$ are calculated by the RA Method per equations (3.26) and (3.28). The equations are below. This is the final step for calculating $K_{pt}$, but $K_{it}$ requires one more step.

$$K_{pt} = K_{pt,i} \cdot k_{TLsr} \cdot R_b^2 \tag{3.35}$$

$$K_{it,m} = K_{it,i} \cdot k_{TLsr} \cdot R_b^2 \tag{3.36}$$

### 3.3.5 Tension Tracking Error Specification

The specification for tension tracking is met by comparing the steady-state value of the simulation of the web line using the gains from (3.35) and (3.36) to the simulation of the web line including (3.37) in the calculation of $K_{it}$. Begin with $k_{fvt} = 1$ and simulate or run the line to see the tension error. Adjust the value of $k_{fvt}$ incrementally until the simulation shows the steady-state tension within the specification. The simulation of the web line must be used because the model developed for calculating the gains so far predicts zero error to a step input and very small error to a ramp input. For the HSWL, the value of $k_{fvt}$ is $6.5$ for 6 inch wide Tyvek. The effect of $k_{fvt}$ is to enlarge the open-loop crossover frequency.

$$K_{it} = K_{it,m} \cdot k_{fvt} \tag{3.37}$$

It is important to note that the final rise time and damping ratio of the tension loop is not what was originally intended. The rise time and damping ratio have been shifted due to the application of $k_{fvt}$ to the integral gain. Even so, maintaining the input values for the tension loop rise time and damping ratio is important because they are the starting point. The final set of gains creates a highly over-damped tension loop so rise time and damping ratio do not make sense as descriptors of the final system.

## 3.4 The Systematic Method for Determining Controller Gains

First, determine the physical parameters of the web line by calculation or measurement. These include the motor inertia, the motor damping, the motor constant (if DC), the roller inertias, the gear ratio between each motor and the spindle that contacts the web, roller bearing friction values, roller radii, span length between rollers, Young's modulus for the web, cross-sectional area of the web. If there is a dancer, determine the type of dancer, arm length (if it is a pendulum style), the dancer's mass and inertia, and applied torque (pendulum) or force (translational). Determine the final steady-state operating web speed for the line. Divide the web line into tension zones, the number of which may be different than the number of motors in the web line. These zones should break at driven rollers and can be either speed controlled or speed-based tension controlled. For each zone, develop a model for the motor speed loop, and if a tension feedback loop or position feedback loop exists, develop a model that relates the motor speed to the tension (measured by a load cell) or dancer position.

Select a rise time, $t_r$, and damping ratio, $\zeta$, for each motor and tension or position control loop. These performance criteria can be the same for each control loop in the web line or specific to each loop and each motor. On the EWL, the performance criteria for all loops had the same damping ratio and rise time. Following the experience on the EWL, the performance criteria for the HSWL speed loops were chosen to be different from the performance criteria for the tension loops so that the poles of the speed loop were in a different place than the poles of the tension loop.

Once the models for the motor speed loops are created for each tension zone in the web line, use (3.16) and (3.17) to determine the related gains. Then create the closed-loop transfer function for the speed loop, $G_s(s)$, and place it in series with the transfer function for the span tension, (3.9), or dancer position, (3.11), making $G_2(s)$. Combine and simplify the transfer functions. Then reduce the order of the transfer function using Hutton's Routh Approximation method making $\widetilde{G}_2(s)$. Once the reduced order transfer function is known, (3.25) can be simultaneously solved, and substituted into (3.28) to determine the tension or position control loop gains. Then the tension loop scaling factors can be applied and $k_{fvt}$ can be applied if a load cell is used. Since the gains

found by this method are based on several simplifying assumptions, they should be considered as a starting point in tuning controllers on the web line.

## 3.5 Summary

This chapter has presented a systematic method for determining the controller gains for a web line using speed control or speed-based tension control. The gains are found under the assumptions of linear models, linearly elastic web materials, that the effect of idle rollers may be neglected, and that second-order model approximations will work in place of higher-order models. The performance criteria are selected by the user and specified by the value-adding process. The process has been laid out with examples from the EWL and HSWL.

# CHAPTER IV

# EXPERIMENTS AND SIMULATIONS

## 4.1 Introduction

Experiments on and simulations of the High-Speed Web Line (HSWL) have been accomplished as part of this research. The key points of the experiments were to verify the ability of the Routh Approximation Method gains to control speed and tension on a web line and work with multiple materials. The key points for simulations of the HSWL were to validate the simulations against experimental data and to demonstrate the usefulness of simulations as test cases for changes to a web line.

In both the experiments and the simulations the HSWL was operated from rest to 800 FPM and allowed to dwell about 30 seconds at that speed. In the experiments the web line was returned to stop. The acceleration of the line from rest to a given line speed is called a start-up. Both the experiments and simulations used the Industrial S-Curve (see Appendix I.2) as the speed reference. The parameters for the Industrial S-Curve are the acceleration rate equals $80$ FPM/s and the jerk rate equals $200$ FPM/s$^2$ in both the HSWL software and the simulation. The reference tension for experiment and simulation was $24$ lbf. The tension feedback signal is conditioned with a lead-lag filter to reduce the noise in the signal. The tension feedback lead-lag compensator frequencies are shown in Table 4.1.

A parameter study was performed for the HSWL in simulation with various lead-lag filter lag frequencies and lower lag frequencies showed improved feedback characteristics. The lag frequency of $8$ rad/s was selected for use with the RA gains. A Bode plot magnitude of both lead-lag

Table 4.1: Tension Feedback Lead-Lag Compensator Frequencies for Rockwell Gain Calculation Method and RA Method

| Method | Lead Frequency (rad/s) | Lag Frequency (rad/s) |
|---|---|---|
| Rockwell | 40 | 20 |
| Routh Approximation | 39 | 8 |

filters is shown in Figure 4.1. The Bode plot shows how the filters will impact the magnitudes of certain frequencies. The filter used with Rockwell gains (blue) passes frequencies up to about 1 Hz with no impact. Frequencies above 1 Hz are increasingly impacted. The filter attenuates about half the magnitude of frequencies above 20 Hz. The filter used with RA gains (red) passes frequencies up to about 0.2 Hz with no impact, but attenuates frequencies above 0.2 Hz increasingly up to about 20 Hz. At 20 Hz and above, the filter used with RA gains attenuates almost 80% of the magnitude of signals. The filtered tension from each of the parameter study simulations is shown in Figure 4.2.

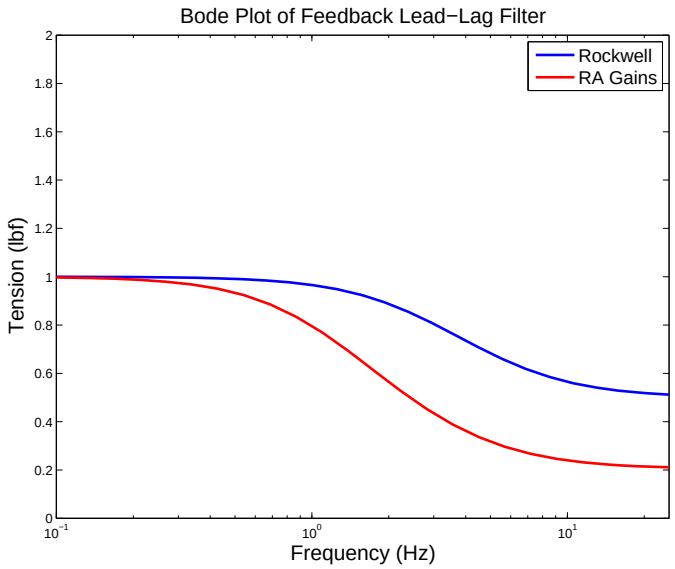



Figure 4.1: The Bode Magnitude Plots of the Lead-Lag Filters Used on the HSWL. The filter for the RA method attenuates more frequencies than the filter used with the Rockwell gains.

For completeness, the RA method was applied to the Euclid Web Line (EWL) before it was decommissioned and that research is reported in Appendix B. The work on the Euclid Web Line (EWL) is considered a stepping stone in preparation for research on the HSWL.

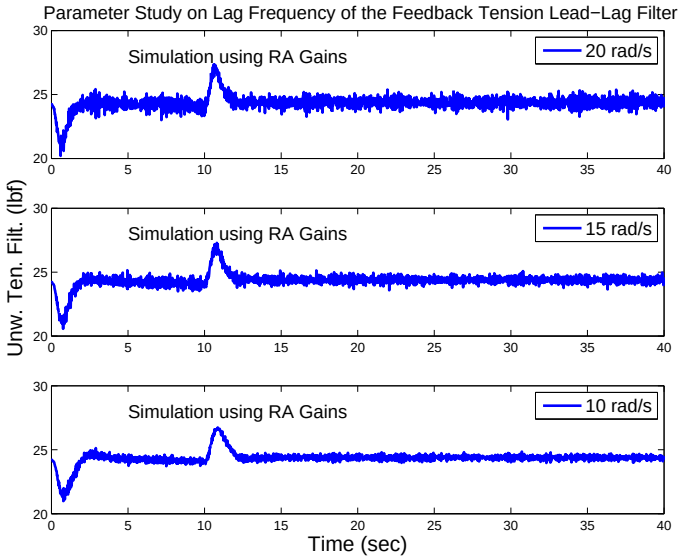



Figure 4.2: A Parameter Study on the Tension Feedback Lead-Lag Filter Lag Frequency. Lower frequencies remove more noise with the Routh Approximation method simulation.

## 4.2 Gain Calculation on the High-Speed Web Line Using the Routh Approximation Method

The Routh Approximation Method of gain calculation has been described in section 3.3 of the previous chapter. The method was applied to the High-Speed Web Line (HSWL) by writing it into the RSLogix 5000® software to take advantage of the automatic roll diameter calculations and associated updates to the roll inertia. The structured text[1] codes written into the software of the HSWL are reproduced in Appendix K.

The Routh Approximation Method of controller gain calculation was prepared and implemented into the RSLogix 5000® software. The motors were modeled as AC motors after reading [63], where modeling AC motors was emphasized. Figure 4.3 is the same as Figure 1.2 except that it shows more detail for the case of the AC motor. The AC motor has two control knobs: current affects the torque delivered by the motor and voltage affects the speed of the motor. As such, an AC motor can deliver a demanded torque at any speed within its range in microseconds [63]. Notice, in Figure 4.3, the addition of torque before the motor dynamics, $G_1$, in the block diagram. The tension loop PI control and the speed loop PI control are both only producing a trim to the next

[1] One of three types of coding used in RSLogix 5000® software.

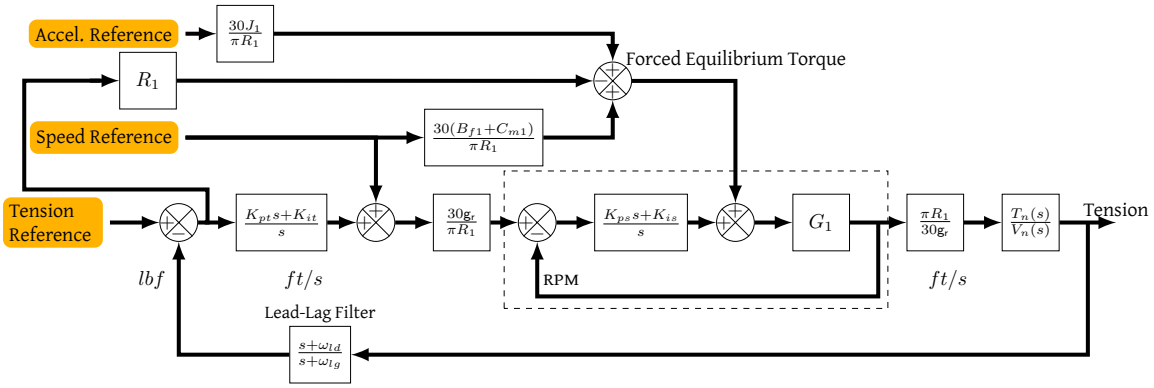



Figure 4.3: Rockwell Control Block Diagram for Use with AC Motors. The Tension Reference is input by the user along with the line speed, but the Accel. Reference and the Speed Reference are from an Industrial S-Curve using the line speed.

block downstream. The equations for $K_{ps}$ and $K_{is}$ are (3.18) and (3.19).

The RA method implemented on the HSWL accepts two types polynomials: a third order polynomial for load cell feedback or a fifth order polynomial for dancer feedback and returns a second order polynomial[2]. The second order polynomial ($\widetilde{G}_2(s)$ from the previous chapter) is used to calculate the initial tension loop $K_{pt,i}$ and $K_{it,i}$ gains using (3.26) and (3.28). These initial gains are multiplied by scaling factors defined in (3.35), (3.36), and (3.37), and are used by the respective controllers. The gains change with each calculation of parent roll diameter in RSLogix because of the build-up ratio.

Idler 2 was neglected in the compilation of the transfer function for the tension feedback using the RA method and spans 1 and 2 were summed together to one span $4.4497$ feet long in order to obtain the third order model. The RSLogix 5000® software then used Cramer's Rule for solving the set of simultaneous equations in (3.25). After solving the set of simultaneous, algebraic equations, the integral gain was calculated using (3.28). The resulting gains for 6 inch wide Tyvek are shown in Tables 4.2 and 4.3 along with the gains that were determined using the Rockwell method (see section 1.6.4) for the pull roll and the rewind. The gains were implemented in the HSWL software and the gains did not work well. The method laid out in section 3.3 was modified by adding scaling factors in section 3.3.4 using (3.35) and (3.36) and a criterion for maintaining the tension within 5% of the set point at steady-state to the rise time and damping ratio criteria. The requirement

[2]Hutton's Routh Approximation method is not limited to reducing a polynomial to second order, but for the systematic method for gain calculation, there was no need for the full capability. The MATLAB code in Appendix I.8.4 retains the full capability

for steady-state tension control may be a specification of the process and not arbitrary. The new criterion was affected by adding a multiplier, $k_{fvt}$, to impact the final value of the steady-state tension regulation per (3.37). The modified tension controller gains are shown in Table 4.4. The gains calculated for the unwind motor at the beginning and ending of each of nine runs are tabulated in Table G.2 in the Appendix. The pull roll and rewind motor gains were calculated by the Rockwell method in all cases. Table G.2 contains gains for three different material types and both Rockwell method gains and RA method gains for the unwind section.

Table 4.2: High-Speed Web Line Motor Speed Gains. The RA gain calculation method was used with with a damping ratio of 0.7977 and a rise time of 0.259 seconds for 6in wide Tyvek on the unwind motor. The Rockwell method was used on the other motors.

| Motor | Method | $K_{ps}$ | $K_{is}$ |
|---|---|---|---|
| Unwind | RA | 10.91 | 111.34 |
| Pull Roll | Rockwell | 24.52 | 91.93 |
| Rewind | Rockwell | 13.18 | 98.85 |

Table 4.3: High-Speed Web Line Tension Control Gains. The Unwind section was calculated with the RA method using a rise time of 0.239s and a damping ratio of 0.9 for 6in wide Tyvek. The rewind section used the Rockwell method. This table shows $K_{pt,i}$ and $K_{it,i}$ for the RA method.

| Section | Method | $K_{pt}$ | $K_{it}$ |
|---|---|---|---|
| Unwind | RA | 7.3462E-02 | 4.3632E-02 |
| Rewind | Rockwell | 120.40 | 240.81 |

Table 4.4: High-Speed Web Line Unwind Tension Control Gains after Modification with Scaling Factors and Tension Error Criteria. The Unwind section was calculated with the RA method using a rise time of 0.239s and a damping ratio of 0.9 for 6in wide Tyvek.

| Section | Method | $K_{pt}$ | $K_{it}$ |
|---|---|---|---|
| Unwind | RA | 86.96 | 335.72 |

## 4.3 Experimental Studies on the High-Speed Web Line

The test plan for the HSWL included three web materials, load cell feedback for the unwind, and load cell feedback for the rewind. The process for each experiment is outlined in Appendix G.2.

The unwind gains are calculated by both the Rockwell and RA methods. The controller gains for the pull roll and the rewind are calculated with the Rockwell method in all experiments. Three experiments were accomplished with each material type, one using the Rockwell gain calculation method and two using the RA method to capture the effect of changing roll radius. The reference web tension was set at 24 lbf for all tests. The initial and final diameters were recorded for each experiment. Nine sets of proportional and integral gains are shown in Table G.2 in the Appendix for the beginning of the experiment and at the end of the experiment along with the roll radii. Because of time constraints, more controlled experiments with only one parameter changing at a time were not accomplished.

### 4.3.1 PET on the High-Speed Web Line

The HSWL was configured with 6 inch wide PET film. The material properties are listed in Appendix A. A 0-800 FPM start-up was accomplished three times with data being recorded. The Rockwell method was used for the first run and then the RA method was used for gain calculations. Figure 4.4 shows the ΔSpeed of the unwind and rewind motors on the left. The ΔSpeed is

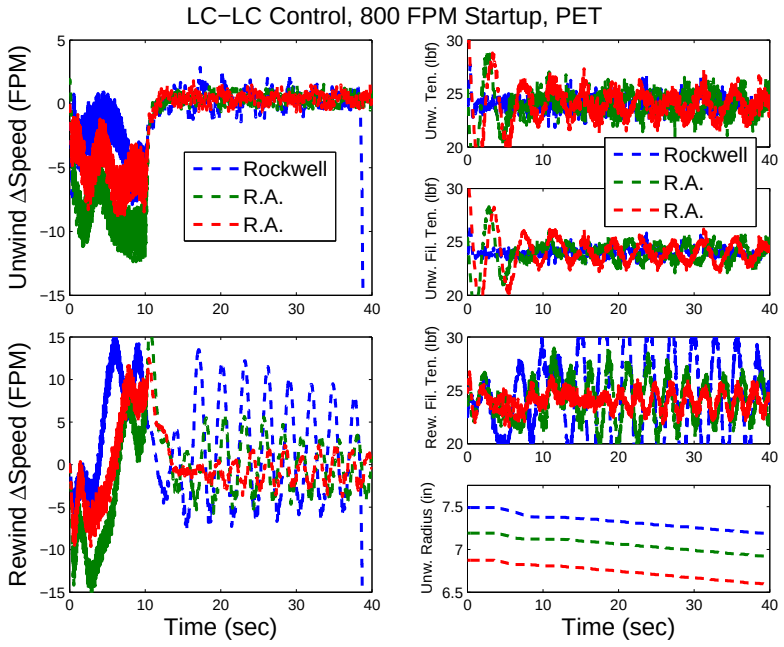



Figure 4.4: Experimental Data for 6 in Wide PET Using Rockwell and RA Gains. With the Rockwell gains tension varies less than with the RA method gains. The RA method gains tracked speed better than the Rockwell method.

found by subtracting out the reference Industrial S-Curve speed for each data sample. If $\Delta$Speed is positive, that means the motor was going faster than the reference speed at that instant. The Rockwell experiment was slightly shorter than 40 seconds which is why the $\Delta$Speed suddenly drops off. The unwind tension (top right) is the unfiltered data. The HSWL has a lead-lag filter on the tension feedback with frequencies listed in Table 4.1. The second plot down from the top is the filtered tension and is the signal the controller acted upon. The filtered tension is plotted by itself in Figure 4.5. The RA method gains maintain better tracking of the the reference speed in the unwind, but the Rockwell gains show less tension variation. The experiments with RA gains show a prominent low frequency oscillation in the tension that the experiment with Rockwell gains does not show. The rewind tension was not well controlled by the Rockwell gains. The rewind tension control seems to improve with increasing rewind diameter (as the unwind radius decreases, the rewind radius is increasing). The bottom right plot shows the unwind roll radius as recorded by the HSWL.

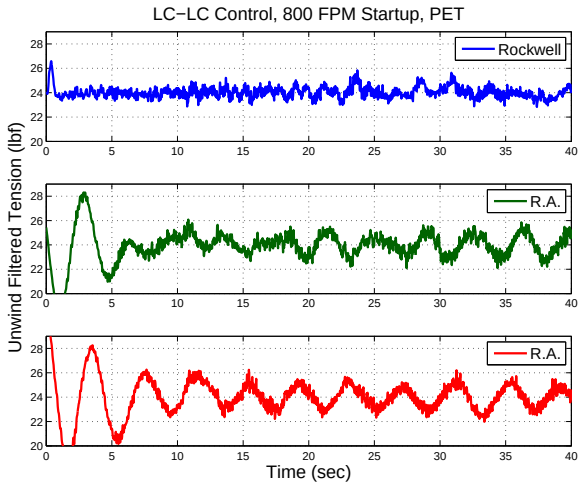



Figure 4.5: PET Filtered Tension Plot with Load Cell Feedback at Both the Unwind and Rewind. The low frequency oscillation is evident in the RA Gains.

### 4.3.2 Narrow Tyvek on the High-Speed Web Line

Tyvek is a nonwoven, spunbond product made from polyethylene which can be water-proof but allows air to pass through it. As such it has been used as house wrap or a moisture barrier. It is also tough and is used to make certain mailers, chemical suits, and wristbands for identification. It's

properties are tabulated in Appendix A. A 6 inch wide roll was mounted in the HSWL and a 0-800 FPM start-up was executed three times; once with Rockwell method gains and twice with the RA method gains. The recorded data is shown in Figure 4.6. The unwind and rewind ΔSpeed plots are on the left. The unwind speed traces are normal with the two RA method gain runs covering up the Rockwell method trace until about 10 seconds. The first RA method trace and Rockwell trace were slightly less than 40 seconds long which is why they suddenly drop off the plot. After the

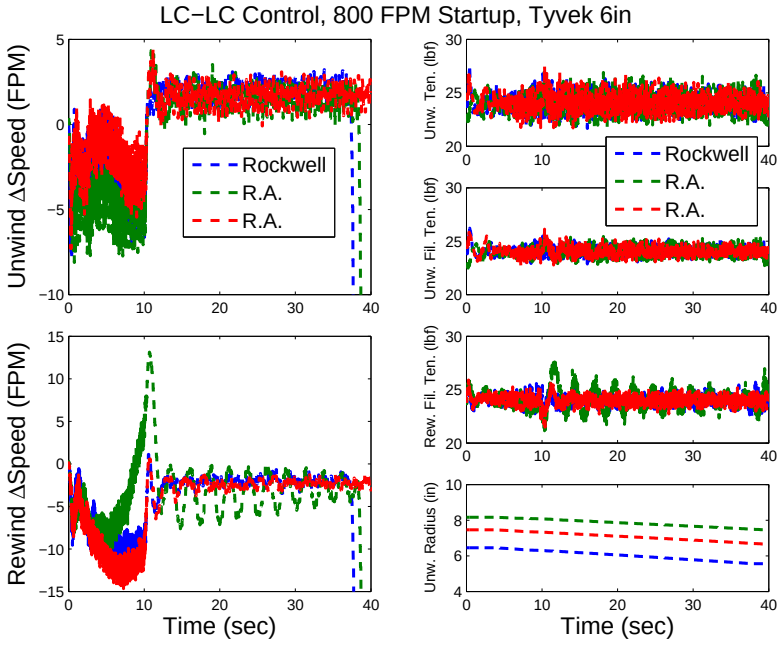



Figure 4.6: Experimental Data for 6 in Wide Tyvek Using Rockwell and RA Gains. The speed tracking is similar in both the Rockwell and the RA gains. The unwind tension has more oscillation, and has a low frequency oscillation after about 10 seconds which settls out. The two middle plots on the right are the filtered unwind and rewind tensions.

steady-state is attained at 800 FPM, the RA gain traces have a smaller oscillation than the Rockwell method trace. On the right-hand side, the tension plots show that the RA method traces have generally more oscillation than the Rockwell method trace, and has a low frequency oscillation after about 10 seconds which settls out. The unwind filtered tension is plotted alone in Figure 4.7. The rewind speed trace for the first RA method (green) has the characteristic of having a stuck diameter calculation in the RSLogix 5000® software. In bad cases, the stuck diameter calculation will cause the reporting of the line speed of the affected motor to be 200 FPM or more over the actual line speed.

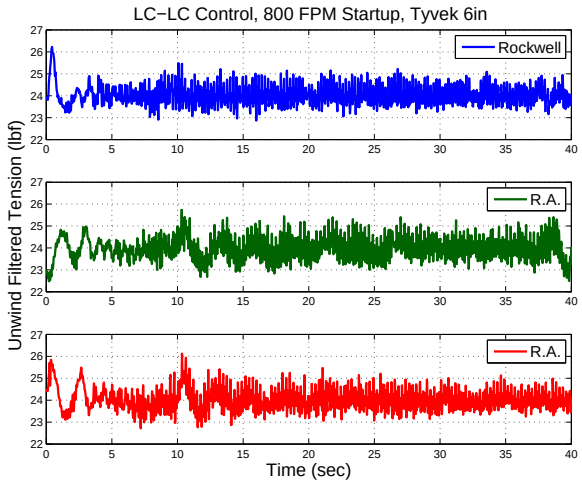



Figure 4.7: Narrow Tyvek Filtered Tension Plot Using Load Cell Feedback at Both the Unwind and Rewind. The low frequency oscillation is evident after about 10 seconds but it settles out in the RA gains plots.

### 4.3.3 Wide Tyvek on the High-Speed Web Line

A second roll of Tyvek was then mounted on the the HSWL and it was 24.5 inches wide. The parent roll weighed more than either the first Tyvek roll or the PET. A 0-800 FPM start-up was executed three times at 24.5 lbf tension (1PLI for the 24.5 inch wide roll). The recorded data is shown in Figure 4.8. The Rockwell gain method was used first so it had the largest diameter. The rewind speed trace shows that characteristic of the stuck diameter calculation which affects the rewind tension adversely. The unwind $\Delta$Speed shows that the overshoot at the transition to steady-state operation decreases with decreasing roll diameter. The unwind tension with RA gains shows a low frequency oscillation which settles out after about 20 seconds at stady-state speed. The unwind filtered tension is shown alone in Figure 4.9 and the magnitudes of the high frequency oscillations is easier to see. The high frequency oscillation of the RA gains plots is smaller in magnitude than that of the Rockwell gains plot, but the RA gains plots have the low frequency component which does not settle out like it did with the narrow Tyvek.

### 4.3.4 Comparing the Experiments

The data presented previously has all been time domain data. Statistics will be used followed by an FFT analysis. Table 4.5 shows the steady-state (after 10 seconds on the previous plots and up to their shutdown) average tension for each experiment and standard deviation. It also shows

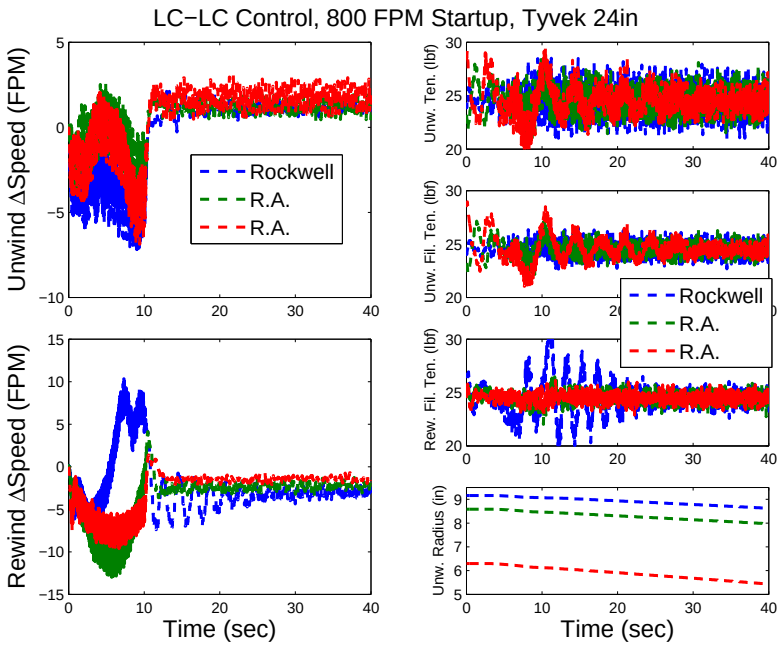



Figure 4.8: Experimental Data for 24 in Wide Tyvek Using Rockwell and RA Method Gains. The speed traces are similar. The tension traces for the RA method has a low frequency oscillation which settles out after 20 seconds. The Rockwell gains tension trace on the rewind shows the characteristic of a stuck diameter calculation.

the average and standard deviation of the the $\Delta$Speed. The experiments with Rockwell method controller gains had less average error to the tension set point and lower standard deviation than experiments with RA method controller gains in all but one case. The experiments with RA method controller gains had a smaller standard deviation from the tension set point on the first RA method experiment with 24 inch wide Tyvek. The $\Delta$Speed average is not consistently smaller with one method. The $\Delta$Speed for experiments with RA method gains was smaller for narrow Tyvek, but was larger for the same width PET and wide Tyvek. The largest percent error to the set point tension (24 or 24.5 lbf) is less than 0.5% and the largest percent error to the speed set point (800 FPM) is less than 0.25%.

The FFT of the experimental tension data is shown in Figures 4.10 and 4.11. The data is again reduced to just the steady-state operation which is the time from about 10 seconds on. Each plot has black and pink dotted lines that indicate the 1-per-rev frequencies of the unwind at the given diameters (black, $< 6$ Hz), the pull roll (pink, 6.3 Hz) , and the idlers (pink, 12.7 Hz) for 800 FPM. The idlers show up in each plot while the pull roll is not nearly as evident. The unwind 1-per-rev is visible in each plot. The major difference between the two methods is the peak around 0.3 Hz

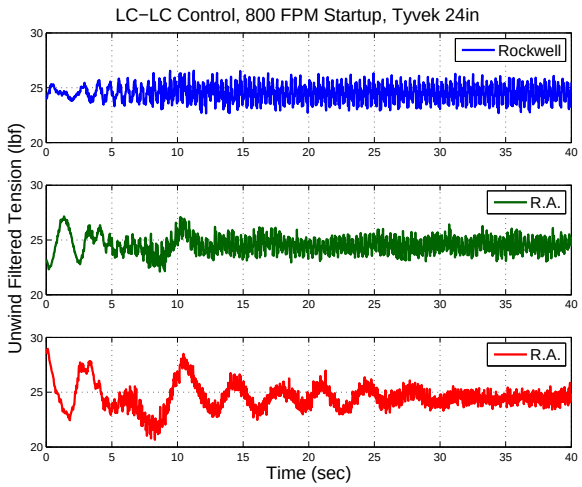



Figure 4.9: Wide Tyvek Filtered Tension Plot. The low frequency oscillation is evident after about 10 seconds and like its narrower predecessor, it settles out in the 24 inch material plots, too, after about 20 seconds. The high frequency oscillation is lower in magnitude for the RA gains than for the Rockwell gains.

which shows up in both materials for the RA

Table 4.5: Average and Standard Deviation of Steady-State Control Feedback Signals at 800 FPM from Experimental Data

| Material | Gain Calculation Method | Ave. Tension (lbf) | St. Deviation (lbf) | Ave. $\Delta$Speed (FPM) | St. Dev. (FPM) |
|---|---|---|---|---|---|
| Tyvek 6in | RA | 24.008 | 0.5221 | 1.517 | 0.976 |
| Tyvek 6in | RA | 24.019 | 0.4426 | 1.704 | 0.800 |
| Tyvek 6in | Rockwell | 24.005 | 0.3836 | 1.946 | 0.770 |
| PET 6in | Rockwell | 24.031 | 0.4472 | 0.236 | 0.993 |
| PET 6in | RA | 24.057 | 0.7517 | 0.244 | 1.003 |
| PET 6in | RA | 24.064 | 0.86 | 0.336 | 0.796 |
| Tyvek 24in | Rockwell | 24.517 | 0.7373 | 1.007 | 0.601 |
| Tyvek 24in | RA | 24.551 | 0.6578 | 1.137 | 0.530 |
| Tyvek 24in | RA | 24.618 | 0.8825 | 1.667 | 0.779 |

method but is not present in the Rockwell method FFT. This low frequency peak is an artifact of the RA method. It has shown up on both the EWL as well as the HSWL tension FFTs.

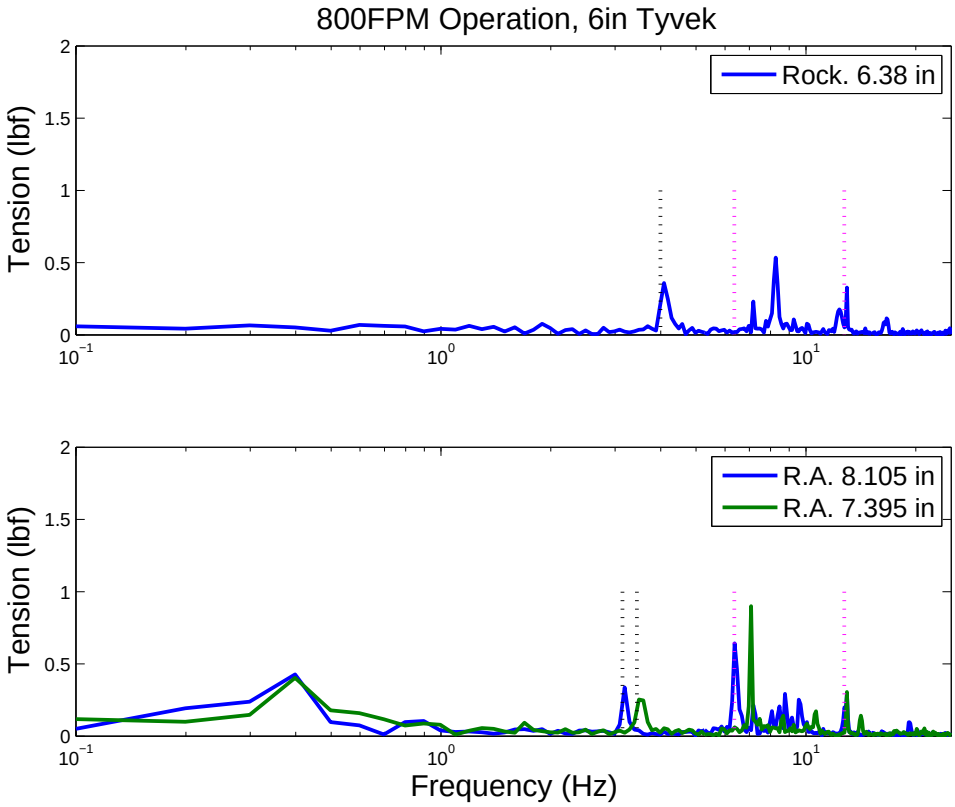



Figure 4.10: FFT of Experimental Tension Data for 6 in Wide Tyvek Using Rockwell and RA Method Gains. The upper plot has the Rockwell method data. The lower plot has the two RA method data sets. The black dotted lines indicate the unwind 1-per-rev and the pink dotted lines indicate the idler (12.7Hz) and pull roll (6.3Hz) frequencies. There is a peak at about 0.4Hz in the RA data that can be seen in the time domain data, too.

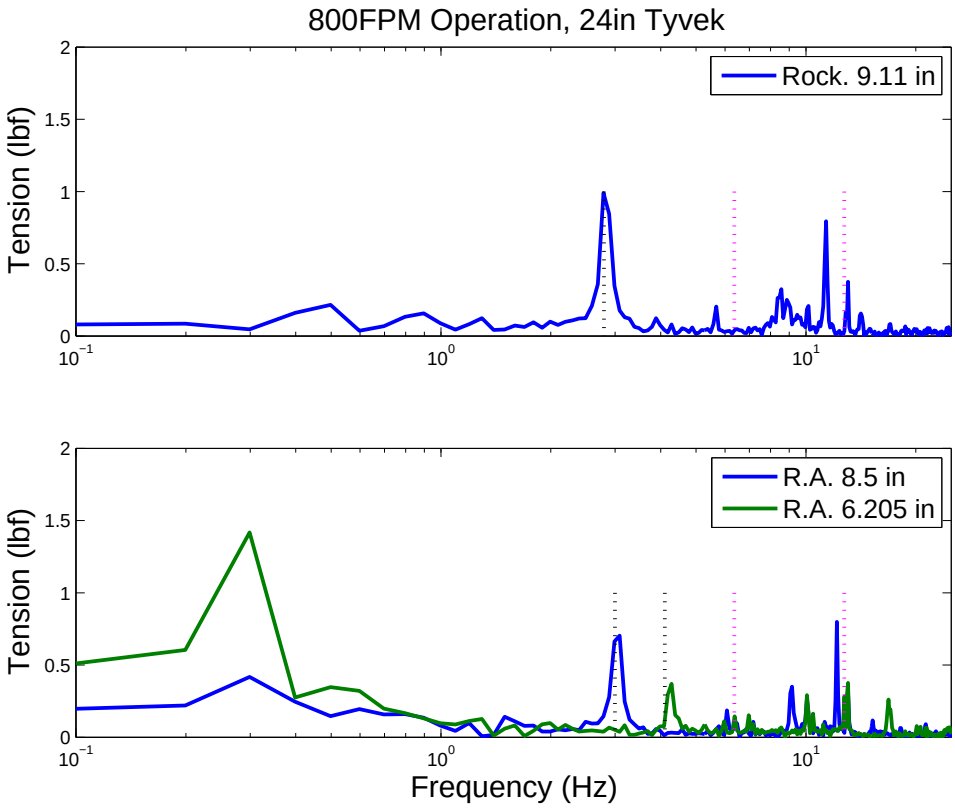



Figure 4.11: FFT of Experimental Tension Data for 24 in Wide Tyvek Using Rockwell and RA Method Gains. The upper plot has the Rockwell method data. The lower plot has the two RA method data sets. The black dotted lines indicate the unwind 1-per-rev and the pink dotted lines indicate the idler (12.7Hz) and pull roll (6.3Hz) frequencies. There is a peak at about 0.3Hz that can be seen in the time domain data, too.

## 4.4 Simulation Studies on the High Speed Web Line

Simulation is a synergy of mathematical models, process flow and timing, physical parameters, and control structures and feedback devices which can predict process variables through time. Web line designers can use simulation to predict problems or alternatively, use parameter studies to isolate root cause in an existing problem. For simulation to be effective, the models used have to be accurate and the process conditions have to be correct. With models and conditions correct, the designer may compare controller parameters, different kinds of feedback to existing controls, or entirely new control structures.

With a simulation, product does not have to be ruined to test a controller concept. Physical constraints of the web line can be evaluated like how far apart to space idle rollers. Material and process constraints can be studied like where air entrainment will cause web handling difficulty which would cause a line speed constraint. Sometimes the subject of a study is an event that is intrinsic with web handling like parent roll diameter change as web is paid out to the process. The changing roll diameter affects the torque moment arm and the inertia of the roll which both play a part in the accurate control of speed and tension in a web line.

Simulating the HSWL requires modeling the physical web line, its control structure, and intrinsic features of the line. The model will be described first and then simulations using the model will be compared with experimental results.

### 4.4.1 Modeling

The simulation of the HSWL is accomplished using differential equations so that nonlinear components can be included in the modeling. MATLAB will be used for the integration, specifically, the `ode45.m` routine is used (more information can be found in [75]). Because of the variable time step nature of `ode45`, the `events` function/feature of `ode45` has to be used to guarantee certain operations line up with the discrete nature of the RSLogix 5000® software being simulated. To capture the state of the model when certain other, for lack of a better term, events happen, a separate `RecordEvents` function will be used which can capture variables calculated by the model, which

are not part of the state as well. The Industrial S-Curve is discussed more in Appendix I.2 with the MATLAB code.

The focus of the simulation is on the unwind section of the HSWL so that motor is modeled in angular speed with a time-varying roll radius following section 2.2.2 of chapter 2. The capability for eccentricity is included in the model, but the offset ($e_1$ in (2.26)) is set to zero. The rewind motor is modeled as a tangential speed with invariant radius using (2.4) with no eccentricity. The pull roll motor (R9 in Figure 1.10) is modeled with a nipped roller using equation (2.11). Both the pull roll and the rewind are treated as AC motors with a equilibrium torque calculated for each one instead of a input current and motor constant. The pull roll and the rewind motors both use the Rockwell method for calculating gains. The feedback device in both unwind and rewind sections is a load cell. Both outputs on the HSWL are filtered through lead-lag filters. The simulation uses the equations from section 2.1.5 for the lead-lag filters. The MATLAB code is in section I.8.6 of the Appendix. The filter time constants are the reciprocals of the values in Table 4.1. The control structure for both unwind and rewind sections is a speed-based tension control shown in Figure 1.4. The control structure for the pull roll is a speed control.

The state of the model is the list of variables that have differential equations. For the HSWL, the state includes a variable for each roller speed, a variable for each span tension, a variable for each angular position of an eccentric roll or roller, a variable for the angular speed of each motor (if modeled to that level), a variable for each lead-lag filter used in the model, a variable for each parent roll radius and a variable for each integrator in the control structure. The HSWL has 29 rolls and rollers, 28 spans, 1 eccentric roll, 1 motor being modeled as angular speed, 2 lead-lag filter states, 1 variable roll radius, and 5 integrator states (one for each controller; there are two in Figure 4.12 which is used for both the unwind and rewind motors) for controlling 3 motors. That makes a 67 variable state for the simulation.

Several intrinsic phenomena of the web line are modeled in the simulation. Friction, stiction, and viscous friction are modeled between each idle roller and the web using a model from [23]. The model serves to simply model instantaneous stick/slip situations at the roller and gives a better indication of the web speed at the roller for span tension calculations. The model is not the same

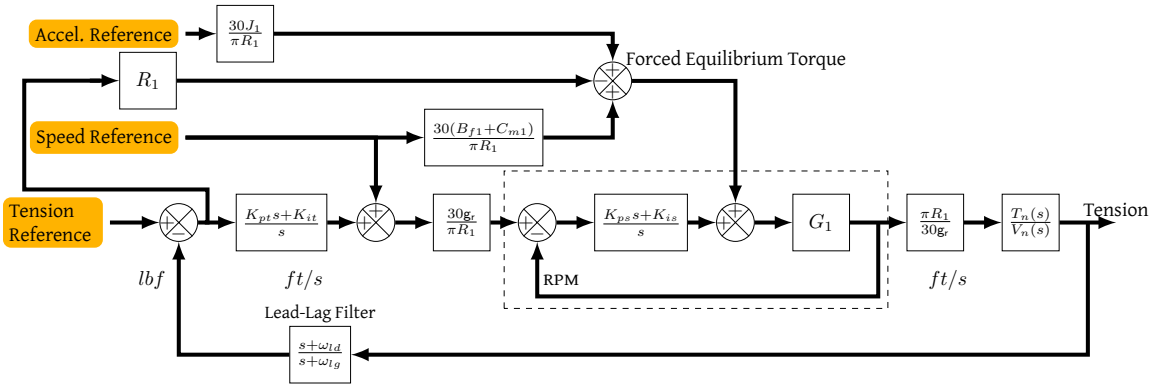



Figure 4.12: Speed-Based Tension Controller Block Diagram for the Unwind and Rewind Motors. Each PI controller has one integrator.

as Ducotey-Good (DG) air entrainment or Whitworth slip which were both longer duration events and discussed in detail in chapter 5. The span tension primitive element used is the nonlinear one, (2.2). The importance of using the nonlinear equations was emphasized in section 2.2.5. The unwind is modeled with a time-varying parent roll diameter using (2.24) for the derivative of the radius and the other elements of the time-varying roll diameter primitive element. Noise is added to the feedback signal from both load cells. That noise is modeled on the unwind with a random number from a normal distribution with the standard deviation taken from Table 4.5 and a zero mean. The gains calculated by the Rockwell method for the unwind were unstable in simulation before the application of feedback noise and became stable after the application of the noise in the simulation. The motors have a limited amount of torque and power. The Rockwell drives can only supply so much power so the simulation has a power limiting feature based on the demanded speed and torque from the controller. The torque is limited to the rated motor torque separately. The Rockwell drives operate in RPM and not rad/s, so the unwind motor model was converted to integrate in RPM instead of base units. This applied to the speed control loop for the unwind motor. It operated in RPM as well. Following that, variables calculated from the motor angular speed had to have a conversion factor installed so that the units canceled properly. The motors all had different starting and rolling torques. These were measured values found through experimentation that were added to the model to simulate the constant losses due to friction and other phenomena at the motor.

The process for compiling a MATLAB code is not without difficulty, but examples from the

simulations used in this research are in Appendix J. Functions and classes have been created to modularize the code into similar units and are contained in Appendix I. The codes include support for reading the data file from Web Transport System (WTS), parent rolls, motors, Gain calculation for each loop, roll shape, roll eccentricity, Ducotey-Good (DG) traction models, event recording, visualizing the web line layout from the position and diameter of the rollers, lead-lag filters, calculating FFTs from data or files, span tension derivative calculation, torque component angle calculation for pendulum dancers, and the Industrial S-Curve input.

### 4.4.2 Simulation Results

Simulation has to answer two questions, one before the other:

- How well does the simulation match the experiment?
- How can it extend planning?

The first question must be answered prior to expecting anything useful out of the the second question. To answer the first question, the experiments on the HSWL are simulated from the same initial conditions as the experiments. After those are shown, the simulation will be used to extend the HSWL by changing the feedback device and changing the length of span between the unwind roll and the load cell.

#### Narrow Tyvek Experiment #1, RA Gains

The HSWL was exercised through a 0-800 FPM start-up at 24 pounds web tension. Figure 4.13 shows the simulation result against the experimental result. The Routh Approximation method gains were calculated for both the experiment and the simulation in the same way. The unwind radius began at 8.10 inches and the rewind diameter was 3.60 inches. The rewind $\Delta$Speed plot (bottom, left) shows a characteristic of the diameter calculation failing to operate in the experiment. The unwind $\Delta$Speed shows similar upsets at the end of the acceleration around 10 seconds. The initial drop in $\Delta$Speed is larger and longer in duration in the experiment than in the simulation. This may be due to the simulation using continuous control methods and the experiment

using discrete control on a continuous system. The tension feedback is noisier in the experiment than in the simulation. The filtered tension is even more attenuated in the simulation. The tension measurement from the simulation is right on the average of the experiment.

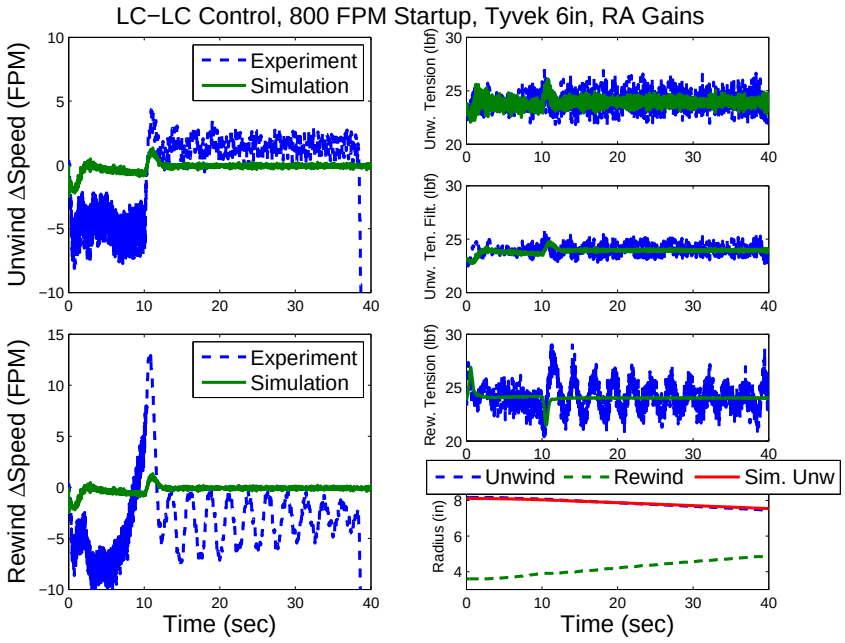



Figure 4.13: Simulation Vs. Experiment for 6in Tyvek with RA Gains, Experiment 1. The ΔSpeed for the unwind shows similar upsets at the end of the acceleration around 10s in both experiment and simulation. The unwind tension has the same upset at 10s just like the experiment. The experiment tension is noisier than the simulation even with noise added.

### Narrow Tyvek Experiment #2, RA Gains

The HSWL was exercised through a 0-800 FPM start-up at 24 pounds web tension. Figure 4.14 shows the simulation result against the experimental result. The Routh Approximation method gains were calculated for both the experiment and the simulation in the same way. The unwind radius began at 7.37 inches and the rewind radius was 4.98 inches. The unwind roll radius is the main difference between this experiment and the previous one. The rewind diameter calculation seems to be working this time. Again, similar upsets in both the simulation and the experiment for ΔSpeed. The unwind tension follows the experiment, but again, the oscillation magnitude is not as large as the experimental tension. The rewind tension show similar upsets at the beginning and at the end of the acceleration around 10 seconds, but otherwise, the simulation is right on the set point tension while the experiment is oscillating a couple of pounds.

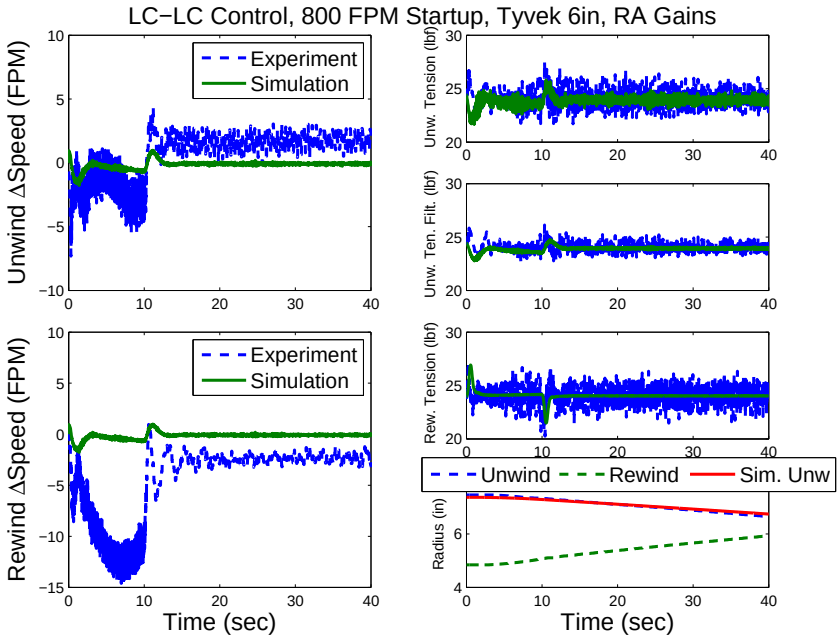



Figure 4.14: Simulation Vs. Experiment #2 for 6in Tyvek with RA Gains. The unwind ΔSpeed plot shows similar magnitudes between the experiment and the simulation. The tension on the unwind has similar upsets in both the experiment and simulation. The rewind tension shows the upsets in both and the simulation stays on the set point.

### Narrow Tyvek, Rockwell Gains

The HSWL was exercised through a 0-800 FPM start-up at 24 pounds web tension. Figure 4.15 shows the simulation result against the experimental result. The Rockwell method gains were calculated for both the experiment and the simulation in the same way. The unwind radius began at 6.38 inches and the rewind radius was 6.18 inches. The ΔSpeed has similar characteristics to the second RA method experiment shown in Figure 4.14. The ΔSpeed for the unwind has more oscillation during the acceleration (0-10 seconds) than the RA method experiments showed. The oscillation in speed translated over to oscillations in tension as well. The oscillation magnitude in unwind tension is almost equal to the oscillations in unwind tension of the experiment. The filtered tension shows the oscillations, but at a smaller magnitude. The rewind tension is similar in both the simulation and the experiment to what is shown in Figure 4.14.

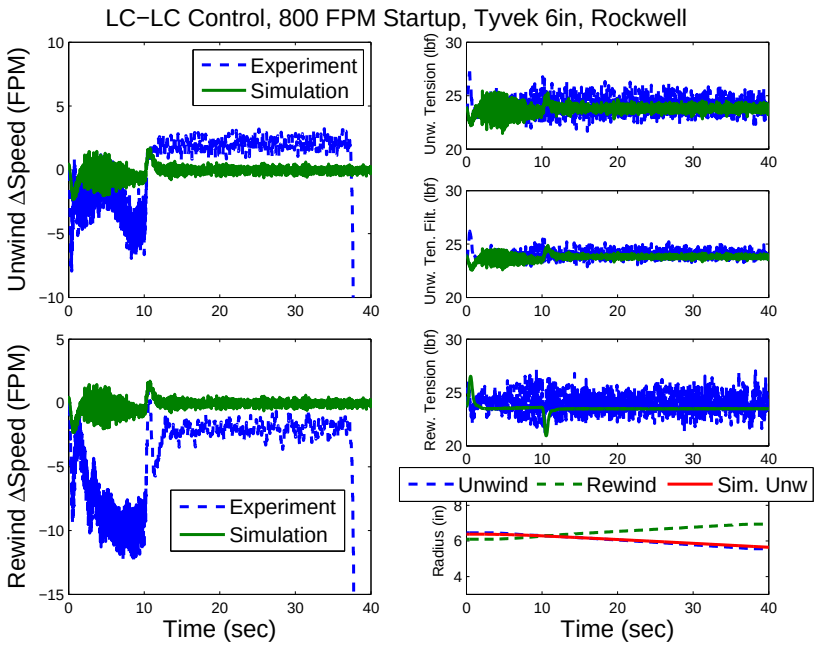



Figure 4.15: Simulation Vs. Experiment for 6in Tyvek with Rockwell Gains. The unwind ΔSpeed plot shows similar upset magnitudes between the experiment and the simulation, though there is more oscillation in the simulation during the acceleration than with the RA gains. The tension on the unwind has oscillation in the simulation from the oscillations in ΔSpeed. The oscillations in the unwind tension have nearly the same magnitude as the experiment.

### Wide Tyvek, Rockwell Gains

The HSWL was exercised through a 0-800 FPM start-up at 24.5 pounds web tension. Figure 4.16 shows the simulation result against the experimental result. The Rockwell gains were calculated the same way on both the experiment and the simulation. The unwind radius began at 9.11 inches and the rewind radius was 2.52 inches. The Rockwell method simulation again shows a large magnitude oscillation in the ΔSpeed during the acceleration (0-10 seconds). Then the oscillations settle out during the steady-state operation. The unfiltered unwind tension simulation has oscillations much larger than the experiment during the start-up. A zoomed-in view is shown in Figure 4.19 where the simulation is plotted at the back for both the unfiltered and filtered tensions. The rewind ΔSpeed again shows the characteristic of a stalled diameter calculation during the acceleration.

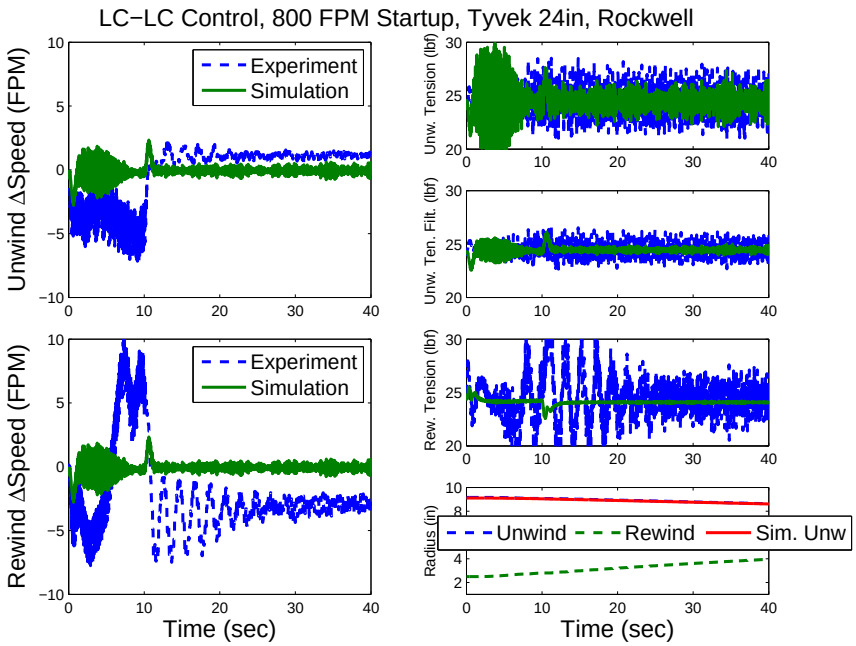



Figure 4.16: Simulation Vs. Experiment for 24in Tyvek with Rockwell Gains. The oscillations in ΔSpeed are large during the acceleration. The oscillations in unwind tension of the simulation are larger magnitude than the experimental tension during the acceleration. The simulation settles down during the steady-state operation. The rewind ΔSpeed shows the characteristics of a stalled diameter calculation.

### Wide Tyvek Experiment #1, RA Gains

The HSWL was exercised through a 0-800 FPM start-up at 24.5 pounds web tension. Figure 4.17 shows the simulation result against the experimental result. The unwind radius began at 8.50 inches and the rewind radius was 4.13 inches. The unwind ΔSpeed for the simulation does not show oscillation like Figure 4.16. The magnitude upsets are similar to the experiment at the beginning and end of the acceleration. The unfiltered simulated unwind tension is closer in magnitude of oscillations to the magnitude of the experimental tension. The simulated filtered tension is smoother than the experimental filtered tension, but that has been the case throughout all of the simulations. The rewind ΔSpeed indicates the diameter calculation was working. The simulated rewind tension settles in on the set point and stays there.

### Wide Tyvek Experiment #2, RA Gains

The HSWL was exercised through a 0-800 FPM start-up at 24.5 pounds web tension. Figure 4.18 shows the simulation result against the experimental result. The unwind radius began at 6.20

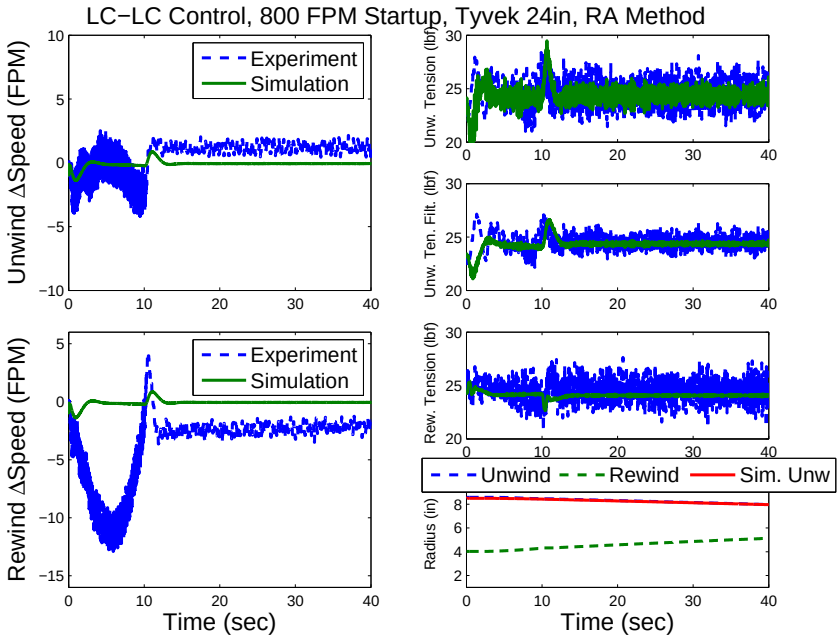



Figure 4.17: Simulation Vs. Experiment #1 for 24in Tyvek with RA Gains. The unwind ΔSpeed for the simulation does not show the oscillation. The magnitude upsets are similar to the experiment at the beginning and end of the acceleration. The unfiltered tension is closer to the magnitude of the experiment. The simulated filtered tension is smoother than the experiment. The rewind ΔSpeed indicates the diameter calculation was working. The simulated rewind tension settles in on the set point and stays there.

inches and the rewind radius was 7.10 inches. The unwind ΔSpeed shows similar characteristics to the previous experiment. The experiment's unwind unfiltered tension has a low frequency oscillation that lasts around 20 seconds into the steady-state operation. The simulation does not show this, but once it attenuates the experimental and simulated tensions are similar to those shown in Figure 4.17. The rewind ΔSpeed and tension are consistent with the previous plots in that they both attain their set points and maintain them. The rewind does not show that the diameter calculation stalled.

### Simulation Versus Experiment Discussion

The last several plots compare simulations of a mathematical model of the High-Speed Web Line (HSWL) to experiments on the same. The differences and similarities are of interest. The discussion will lead off with differences and then go to similarities.

Differences between the simulation and the experiment show up in the ΔSpeed and tension. The acceleration period (0-10 seconds) always show the experimental ΔSpeed dropping negative

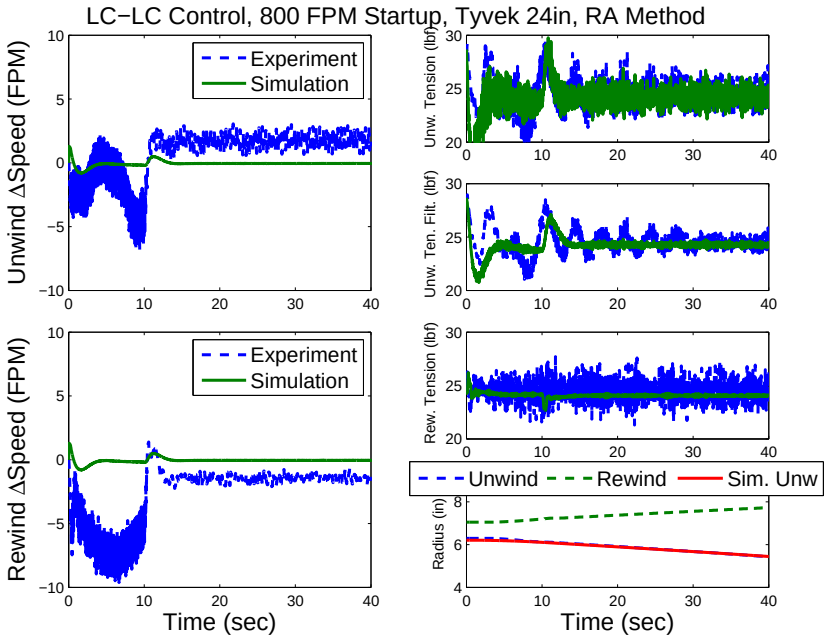



Figure 4.18: Simulation Vs. Experiment #2 for 24in Tyvek with RA Gains. The unwind ΔSpeed is similar in general movement to the experiment. The unfiltered tension compares in magnitude except for the low frequency oscillation after the acceleration finished. The simulation does not show this. The rewind ΔSpeed and tension both attain their set points and maintain them.

on the unwind motor. The simulation has a second or two drop but then reattains the $0$ error and continues. The main difference is that the actual web line is running a discrete controller on a continuous system. That can slow down the response of the controller. Then there is the steady-state error in the unwind of about $+2$ FPM while the simulations are very near zero error. This can be partly explained by the estimate of the wound in tension in the unwinding roll not being accurate. Increasing the wound-in tension simulated in the roll raises the steady-state operating speed, but not all the error can be explained that way. The other component is the measurement of the roll radius on the line. Comparing the simulated radius to the experimental radius in the wide Tyvek experiment with Rockwell gains (Figure 4.16) at $19.4$ seconds, the simulation radius was $8.909$ inches while the experimental recorded radius was $8.945$ inches. Less than $0.04$ of an inch makes a bit more than $3$ FPM difference at $171.5$ RPM. Tension shows a marked difference between Rockwell method and RA method in both the $6$ inch wide Tyvek and the $24.5$ inch wide Tyvek during the acceleration. The Rockwell method simulations have a large magnitude oscillation that increases and then decreases as the line speed is met. Some of the tension oscillation is due to speed oscillation which is happening at the same time. The system is resonating in simulation

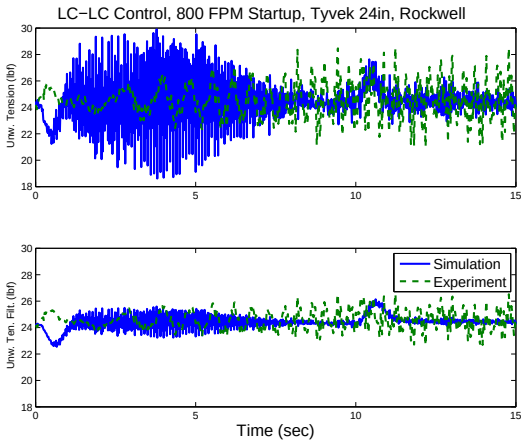



Figure 4.19: Simulation Vs. Experiment for 24in Tyvek with Rockwell Gains Zoomed in on 0-15 second. The simulation is plotted behind the experiment so that the larger oscillations are behind the experimental plot for both unfiltered and filtered unwind tension.

and the resonance is not present in the experiment (see Figure 4.19). The reason the resonance does not show up in the RA method simulations is that the RA method used a different feedback tension filter lag frequency than the Rockwell method used. Figure 4.20 shows a parameter study on the lag frequency of the lead-lag filter used on the unwind tension in simulation. Figure 4.2 shows a parameter study on the lag frequency of the lead-lag filter used on the unwind tension in simulation with RA gains. The resonance is removed by reducing the lag frequency $5$ rad/s. From Table 4.1, the RA method used an $8$ rad/s lag frequency. The rewind $\Delta$Speed and tension differences in Figures 4.13 and 4.16 are due to the diameter calculation stalling and that impacts the recorded speed and the tension reacts to the speed.

Similarities between the simulations and the experiments also show up in the $\Delta$Speed and the tension. In $\Delta$Speed there is an initial drop in speed and then an upset at the end of the acceleration. This comes from the demanded torque for the motor being calculated in the same way on the HSWL and in simulation. The speed upsets cause tension upsets both experiment and simulation. Sometimes it is easier to observe the upsets in tension than others (cf. Figures 4.13, 4.14, 4.17, and 4.18 for examples in the unwind tension). These cover the expected outputs of a simulation. The simulation is going to be more smooth in response and more accurate because it is only happening in the realm of mathematics. Evens so, the simulation produces the general character of the response: it follows the reference ramp, it has overshoot, it oscillates around the reference. Another similarity that is hard to put a value on is that the gains for the HSWL can be directly used in the

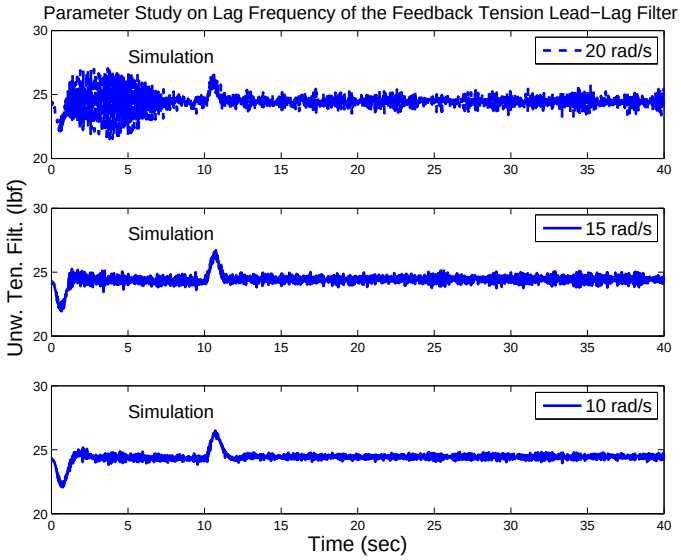



Figure 4.20: A Parameter Study on the Tension Feedback Lead-Lag Filter Lag Frequency. A little lower lag frequency removes the resonance on the Rockwell method simulation.

simulation with no conversion and vice-versa. This capability of the simulation is not an easy one to replicate.

### 4.4.3 Simulation for the Web Line Designer

The previous section showed that the simulation described in this chapter produces usable results that are similar to experimental results from the HSWL. Using the same process to build the simulation for another web line would produce the same level of results. The web line designer can use simulation to predict effects on an existing web line or predict a line that is not built yet. A simulation can show best case results for operating the line at a higher line speed or changing the web material and/or width. One example has already been given in studying the lead-lag filter lag frequency effects on the Rockwell and RA method simulations. In section 2.2.5, a disturbance was added to the EWL and simulated with nonlinear primitive elements. The nonlinear simulation was accurate to the speed of the unwind roll, dancer position, and tension in the span compared to the experimental data. The simulation did not have as much frequency content as the experiment, but it was created with idealized models and captured the general trends. Simulation can be used to extract root cause for disturbances in the web handling process. Simulation can also give insight into changing the structure of the web line. The following are two examples using the

HSWL simulation.

Suppose the unwind motor had to be moved to make room for bigger parent rolls. That would cause a longer span distance to the load cell for feedback (double the length of spans 1 and 2 in Figure 1.10). The simulation in Figure 4.21 shows what the unwind filtered tension feedback would be along with the change to the gains calculated for each case for the duration of the simulation. The simulation used 6 inch wide Tyvek and RA method gains. Doubling the span length between the unwind and the load cell basically doubled the magnitude of the gains using the RA method. The tension feedback for the doubled span length has slightly smaller magnitude than the normal case. From the simulation, the designer could conclude that the load cell could stay in place when the unwind motor was reconfigured.

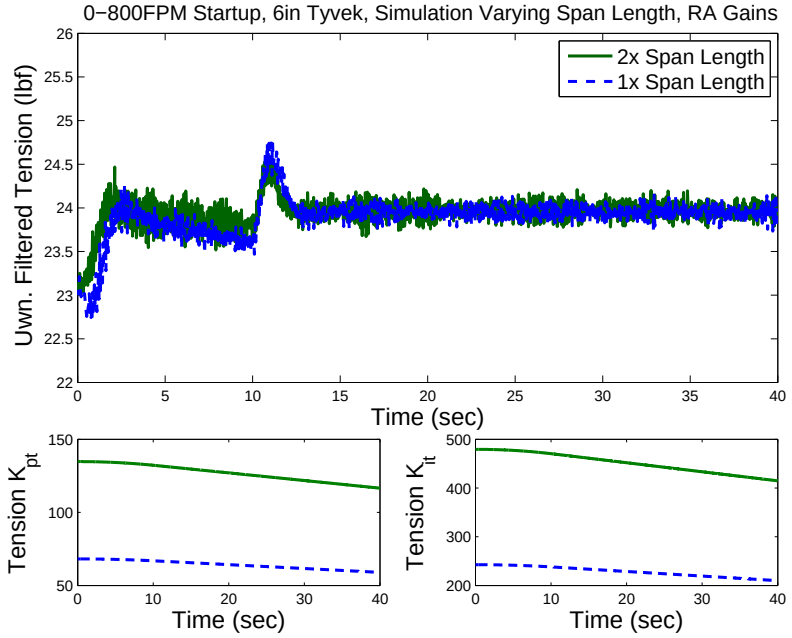



Figure 4.21: Simulating Doubling the Span Length between the Unwind and the Load Cell. The top plot show the filtered tension feedback. The doubled span length tension has less oscillation magnitude than the normal case. The lower plots show the variation between cases and over the duration of the simulation of $K_{pt}$ and $K_{it}$.

Suppose the unwind section needed a feedback dancer to better attenuate tension disturbances. The situation is well known and Appendix C and [76] show better than a 60% reduction in tension disturbances using a dancer as compared to load cell feedback on the EWL. Reconfiguring the simulation of the HSWL to include a dancer at roller 8 added two states to the state of the model simulation. The simulation uses 6 inch wide Tyvek and shows both Rockwell gains and

RA gains for the unwind dancer feedback. The rewind uses load cell feedback and the both the pull roll and the rewind use the Rockwell method to calculate gains. The span extension from the previous example is also included for the RA method. Figure 4.22 shows the tension at the current load cell position in the unwind section at the top. The filtered dancer position in percent of maximum travel is in the middle for all three cases. The RA gains predict a smaller displacement than the Rockwell method does. Doubling the span length between the unwind and the load cell caused a larger oscillation in tension at the load cell, but only a little oscillation in position of the dancer. In the simulation, no $k_{fvt}$ was required for the performance shown. The web line designer can conclude that a dancer at roller $8$ will perform well as a feedback device for the unwind. An added benefit is that the gains calculated for the simulation of the RA or Rockwell method can be used directly in the RSLogix 5000® software. The designer knows starting values for the controller gains from the simulation.

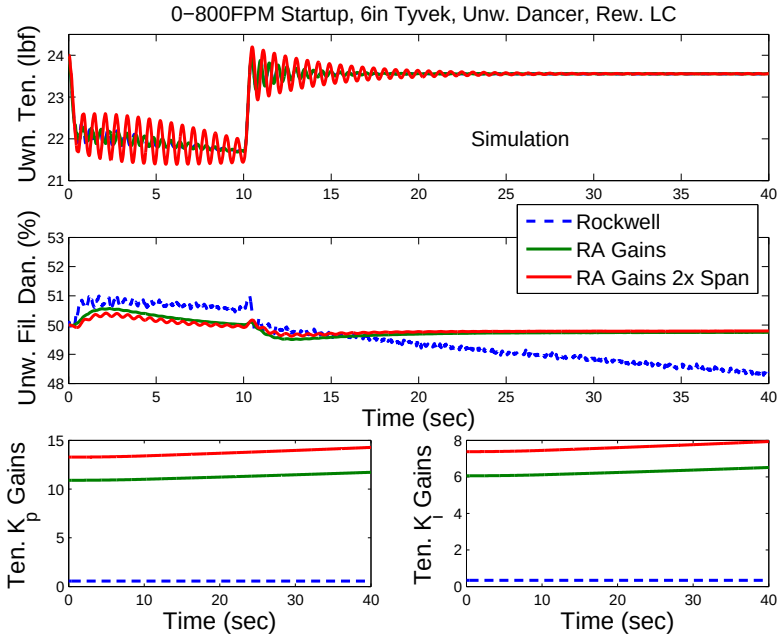



Figure 4.22: Simulating Dancer Feedback on the HSWL with Rockwell and RA Gains. The tension at the load cell (top) is similar in all three cases with the doubled span length case having the most oscillation. The dancer position (middle) has smaller displacement from 50% with RA gains compared to Rockwell. The change in $K_{pt}$ and $K_{it}$ gains between methods as well as across the duration of the simulation are shown at the bottom.

## 4.5 Summary

Gains were calculated for the HSWL using the Rockwell method for the pull roll and rewind and for comparison on the unwind and the Routh Approximation Method was used for gain calculation on the unwind motor assuming AC motor characteristics. Six sets of PI gains were calculated by the Routh Approximation Method for different starting radii on the unwind roll and three sets of PI gains were calculated by the Rockwell method for the unwind. The RA method gains were comparable to the Rockwell method gains overall. The Rockwell method produced lower amplitude oscillations in tension, but the RA method produced better speed tracking. Both methods were sensitive to diameter calculation failures when comparing tension control.

The Rockwell method and the RA method both depend on user-defined specifications, but the Rockwell method requires experience to select the specifications. The RA method does not. The Rockwell method is based on frequency domain techniques, whereas the RA method is based on time domain techniques.

# CHAPTER V

# MODELING SLIP BETWEEN A WEB AND A ROLLER

When a roller is moving faster or slower than the web passing over it, the condition is called slip. Slip can cause surface defects in the web because there continues to be contact between the two surfaces which can drive the quality of the product down. Low quality product is a loss and undesirable. Understanding the mechanism of slip can shed light on ways to mitigate the problem. Modeling the condition is the path this study will take.

Coulomb friction plays a key roll in slip as the basic model and criteria for describing slip [23, 44, 45]. Slip may be partial or total, or a combination of both. This research considers total slip only, i.e. slip over the entire area of contact between a web and a roller. Partial slip is an interface phenomena which is beyond the scope of this work. When total slip occurs, the instantaneous speed of the web is different than the surface speed or tangential speed of the roller. The translational movement of the web is not canceled out by the rotation of the roller as it is under the no-slip assumption. This idea is used later in this chapter to develop a model for slip.

The Euclid Web Line (EWL) is used as the testbed to study slip both analytically and experimentally. The EWL is one of three full scale lines in the Web Handling Research Center (WHRC) at Oklahoma State University. Figure 1.5 is a schematic diagram of the EWL. Figure 5.1 is a photograph of the EWL taken from the unwinding end. The line has three primary control sections: a dancer-controlled unwind section, the process section between the S-wrap and the pull roll, and the rewind section. The unwind section of the EWL contains one unwind roll, seven idlers, one

Figure 5.1: Photo of the Euclid Web Line. The unwind section is in the foreground.

dancer, nine free spans, and terminates at the lead S-wrap driven roller. The process section contains the 2 driven rolls that make up the S-wrap, four idlers, six spans, and terminates at the pull roll (R16 in Figure 1.5). The rewind section contains the pull roll, eight idlers, nine spans, and the rewind roll. The dynamic model used for simulating any section of the EWL is a coupled set of algebraic and differential equations (primitive elements) from chapter 2. It is made specific by the parameters used in the models and the parameters are tabulated in Appendix A. For the simulations in this research, the parameters have been measured, calculated from experiments, or found in documentation of the EWL and its components.

## 5.1 Slip Models

Whitworth defined criteria for determining if slip is occurring or not and developed an approach to estimate the tensions in the spans upstream and downstream of a roller where slipping occurs [45]. The criteria utilize the Capstan equation:

$$\left(\frac{t_n}{t_{n-1}}\right) = e^{(-\mu_n \theta_{wn})} \tag{5.1}$$

where $n$ designates the $n^{\text{th}}$ roller, $\mu_n$ is the coefficient of friction of the web on the roller, and $\theta_{wn}$ is the wrap angle of the web on the roller. The derivation of the Capstan equation along with notes on Whitworth's process shown in terms used in this research are in Appendix E.

### 5.1.1 Whitworth Criteria and Model

Rearranging (5.1) and recognizing that the equation can be represented as two inequalities results in the two criteria defined by Whitworth.

$$t_n - t_{n-1}e^{(-\mu_n\theta_{wn})} < 0 \tag{5.2}$$

$$t_n - t_{n-1}e^{(\mu_n\theta_{wn})} > 0 \tag{5.3}$$

Equation (5.2) indicates the case when the web speed is less than the roller speed and (5.3) indicates the case when the web speed is greater than the roller speed. The upstream and downstream tensions for the case where there is slip are $t^s_{n-1}$ and $t^s_n$ respectively. A derivation involving the total strain from roller $n-1$ to roller $n+1$ gives the results (modified versions of (1.10) and (1.11)).

$$t^s_{n-1} = \frac{t_{n-1}L_{n-1} + \phi t_n L_n}{L_{n-1} + L_n\phi e^{Ind_n\mu_n\theta_{wn}}} \tag{5.4}$$

$$t^s_n = t^s_{n-1}e^{Ind_n\mu_n\theta_{wn}} \tag{5.5}$$

where $\phi = \frac{E_{n-1}A_{n-1}}{E_nA_n}$ and is the ratio of web properties across the roller number $n$, $t_{n-1}$ is the tension in the incoming span assuming adhesion (which is replaced by $t^s_{n-1}$ if a slip condition is true), $L_{n-1}$ is the span length of the incoming span assuming adhesion (instead of the $L^e_{n-1}$ as shown in [46]. Whitworth assumes $L^e_{n-1} \approx \overline{L_{n-1}}$, the average span length, which he assumes is $L_{n-1}$ in [45]), and $Ind_n$ is the slip indicator taking a value of $-1, 0$, or $1$ based on (5.2) and (5.3). If (5.2) is true, then $Ind_n = -1$. If (5.3) is true, then $Ind_n = 1$. If neither equations (5.2) nor (5.3) are true, then $Ind_n = 0$ [45]. Equations (5.4) and (5.5) are the results of equalizing the strain in the effective span length when the web is slipping to the strains determined when a no slip condition is assumed to exist. Whitworth's model is a two-step process: (a) a web-roller system is simulated at an instant in time assuming no slip, then (b) the criteria for slip, (5.2) and (5.3), are checked and the span tensions are adjusted using (5.4) and (5.5) if slip exists.

### 5.1.2 Ducotey-Good Traction Model

Ducotey and Good presented a model in [38] and [42] that predicts a coefficient of traction between a web and a roller. The coefficient of traction depends on the air film thickness between the web and roller and the surface roughness of both the roller and the web. The model shows that the coefficient of traction virtually decreases to zero as the air film thickness increases. Once the air film thickness is large enough so that the traction becomes sufficiently small, no appreciable input torque is supplied by the web to the roller. In this case, a roller that is first spinning would spin down to a stop because of bearing friction alone.

The DG traction model is not a slip model because the two surfaces are not touching when the air film has driven the coefficient of traction to zero. However, the DG model causes consequences like slip does. Web and roller could be at different speeds. Tension disturbances when web contacts the roller would be present. Tension changes due to no adhesion to the roller could also happen, so it is included here.

### 5.1.3 The Sliding Friction Driven Roller (SFDR) Model

The Whitworth model is limited to the regime of slip where the change in span tension is affected by the total strain of the web from roller $n-1$ to roller $n+1$. The effects of slip are passed to the roller through decreases in the tension difference across the roller. The DG model assumes an air film exits between the web and roller, which if the air film is thick enough, would allow for an effective disconnect of the web and roller. But, both the web and the roller have surfaces with asperities (roughness). Some level of adhesion due to contact between asperities is necessary if torque is to be transmitted from the web to the roller. If the air film thickness exceeds the asperities, torque can only be transmitted due to the viscosity of the air. This would be very small in a practical case.

The roughness of either the web or roller is assumed sufficient such that the normal force between the web and roller is sufficient, and torque will be transmitted through sliding friction. This assumption is the basis for the Sliding Friction Driven Roller (SFDR) model. A constant coefficient of friction between the web and roller and Whitworth's tension model, (5.4) and (5.5), as well as

Whitworth's criteria, (5.2) and (5.3), are used in the SFDR. The result is a model for the roller during sliding as shown in the equations below (see Appendix F for derivation of (5.6)):

$$F_n = \frac{-t_{n-1}\sin(\theta_{in}) + t_n \sin(\theta_{on})}{\cos(\delta)} \tag{5.6}$$

$$\frac{\mathrm{d}v_n}{\mathrm{d}t} = \frac{1}{J_n}\left(-B_{fn}V_n + Ind_n F_n \mu_n R_n^2\right) \tag{5.7}$$

where $\theta_{in}$ and $\theta_{on}$ are the incoming and outgoing span angles relative to the roller, $\delta$ is the angle to the resultant normal force which has to be determined based on geometry, $\mu_n$ is the dynamic friction coefficient which is assumed to be equal to the static coefficient of friction, and $Ind_n$ is the slip indicator used in (5.4) and (5.5). The indicator $Ind_n$ indicates whether the sliding friction force is positive or negative on the roller. Otherwise, the roller model used is (2.1).

Some logic is needed to know when to switch between the sliding case and the non-sliding case. The Whitworth criteria are that logic. Equations (5.2) and (5.3) are used to select which case is present and when to switch. When the system or subsystem model is being simulated, the Whitworth criteria are evaluated at every integration step. If slip is indicated, the SFDR model is used to define the friction force resulting from sliding contact and use that to apply torque to the roller rather than the model in (2.1).

Initial work on this project was accomplished using the MATLAB `ode45` [77] integration routine with events defined for the switching of roller equations in the SFDR model. On the surface this sounds reasonable. But, the exactness with which MATLAB executes finding the time at which the event occurs is problematic. To circumvent this problem, a Runge-Kutta 4 integration algorithm with fixed time step (see Appendix I.7 for the code) was used when simulations of the SFDR model were performed.

### 5.1.4 Observations from the Model

One of the items of interest in looking at slip from a modeling perspective is when slip occurs. The Whitworth criteria only indicates when slip initiates and not how much, for example, the roller velocity decreases. The steady-state condition is one of those times when the initiation of slip may be indicated, i.e., when the roller velocity derivative is zero. However, there are times where an

upset in either speed or tension causes the velocity derivative to be zero. It is this situation where slip can have its greatest impact. If the time derivative in (2.1), is set to zero, the equation becomes

$$B_{fn} v_n = R_n^2 (t_n - t_{n-1}) \tag{5.8}$$

So bearing friction can cause slip to initiate. Barring other disturbances, lower bearing friction means higher line speeds with no slip. Solving (5.8) for the speed of the roller and using estimated values for bearing friction, shows that the EWL would have to run in excess of 1200 FPM for slip to become a problem at the assumed tension levels.

## 5.2 Experimental Studies on the Unwind Section

Experiments were performed on the Euclid Web Line (EWL). The experiments included idler speed tracking with an additional torque applied to the idler, evaluation of the coefficient of friction between the web and roller before and after the surface was treated, and idler bearing friction determination.

### 5.2.1 Additional Torque Applied to Roller R9

The EWL can operate at line speeds up to 500 FPM. Slip could not be detected at any speed up to and including 500 FPM. An additional torque was applied to roller R9 (Figure 1.5) to cause slip at line speeds feasible for the Euclid line. Figure 5.2 shows the experimental setup where a piece of cloth was draped over the roller and weights were suspended from the cloth to apply a known tension. The difference between the tension meter reading and the hanging weights was assumed to be the force applied to the roller in addition to its normal bearing friction. A wheel encoder was used to capture the speed of the roller or web during each run. The encoder had very little bearing friction compared to the idler.

With the additional torque added to roller 9, the EWL was run through a start-up from 0 to 400 FPM, held at 400 FPM for about 80 seconds, and then shutdown. The force meter was read after the line speed reached 400 FPM. The hanging weight was recorded for each run and the roller speed was captured with the wheel encoder. Traces of idler speed as indicated by the wheel encoder

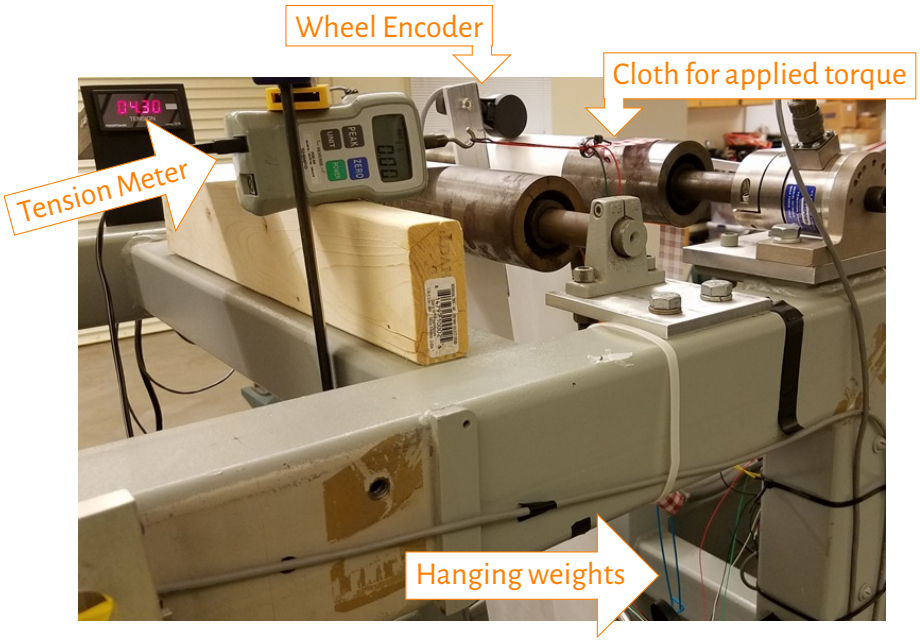



Figure 5.2: Additional Torque Setup on Euclid Web Line Caused Slip to Occur at Attainable Speeds. The encoder is pictured on the web, but it could be moved to capture the roller speed.

are shown in Figure 5.3. Between 0 and 1.745 lbf hanging weight, no slip is evident in the idler speed. The additional torque causes slip at 2.245 lbf. Higher hanging weights cause the idler to attain lower quasi-steady-state speeds until at 2.569 lbf, the idler stops several times during the steady running at 400 FPM. With that much hanging weight, the idler did not reach 100 FPM before slowing down. Hanging weights up to 3.379 lbf were applied, but above 2.569 lbf, the results were similar to that of the idler with 2.569 lbf hanging weight.

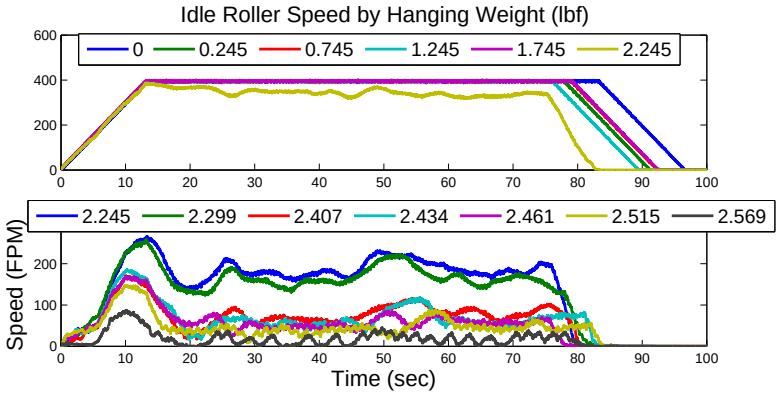



Figure 5.3: Traces of the Idler Speed as Indicated by the Wheel Encoder. Slip initiates with a hanging weight of about 2.2 lbf and the idler comes to rest with a hanging weight of about 2.56 lbf.

### 5.2.2 Determining Coefficient of Friction between Web and Roller

Since the coefficient of friction of the web on the roller is important to the SFDR model, the surface finish of the idler was examined. The roller was original equipment with the line and it had become rusty. A coefficient of friction experiment was accomplished in accordance with the method described by R. J. Lynch [78]. The average coefficient of friction of the original roller was 0.24. The post cleaning result was 0.12. Figure 5.4 shows the before and after rust removal idler photos and the coefficient of friction test results.

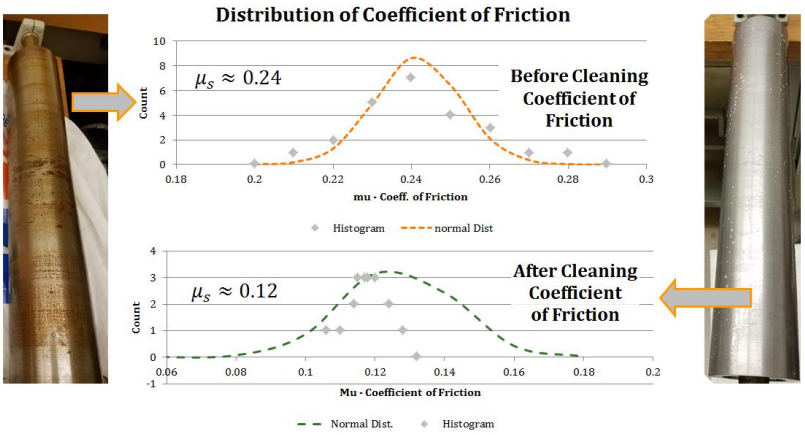



Figure 5.4: The Difference a Rust Free Roller Makes in Coefficient of Friction is 50%.

### 5.2.3 Additional Torque Applied to Roller R9 – Post Rust Removal

With the additional torque added to the rust free idler 9, the EWL was run through the same procedure described two sections earlier. Traces of idler speed as indicated by the wheel encoder are shown in Figure 5.5. The idler speed indicates that slip initiated with only 1.419 lbf hanging weight instead of 2.245 lbf before rust removal. The idler came to rest during the run with a hanging weight of only 1.568 lbf instead of 2.569 lbf as before. The results in Figure 5.3 and Figure 5.5 show that when the primary mechanism for providing torque to an idler is sliding friction, the better the surface finish the more likely slip will occur at that idler at lower line speeds.

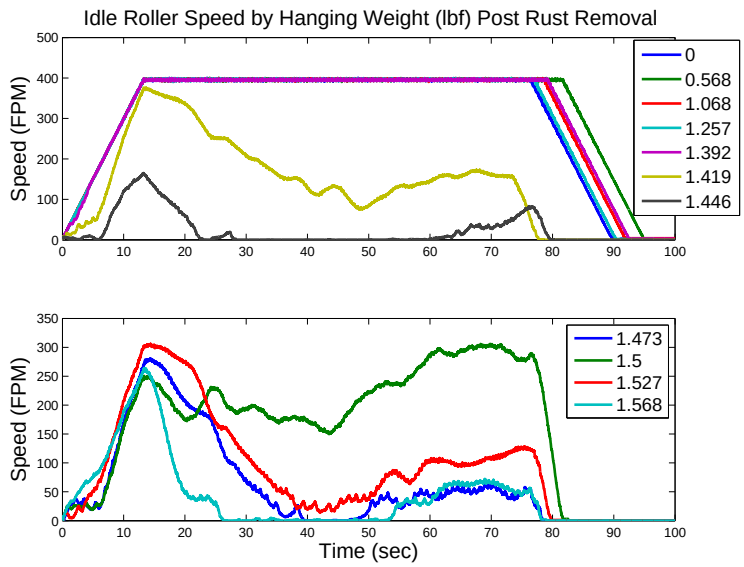



Figure 5.5: Idler Speed After Rust Removal with Varying Additional Torques Shows That Slip Initiates at About 1.4 lbf and the Idler Comes to Rest During the Run with About 1.56 lbf Hanging Weights.

### 5.2.4 Bearing Friction Experiment

The bearing friction of the idler was evaluated using a spin-down test as described in [78]. Two cases were considered - when the bearing was cold and then after running the line for 10 minutes. Figure 5.6 shows the "cold" and "warm" spin-down speed traces from the wheel encoder. Four runs were made for each case. Different initial speeds were used in order to separate the traces so the general character could be seen. The final speed for all traces in Figure 5.6 was 200 FPM. Two types of models were considered to represent the spin down tests – a viscous friction based model and a Coulomb friction based model.

A roller mounted on bearings which provide a resisting torque that is a simple function of rotary speed, can be modeled as a first order linear system as shown by the following equation,

$$J_n\dot{\omega}_n + B_{fn}\omega_n = 0 \tag{5.9}$$

where $J_n$ is the rotary inertia, $B_{fn}$ is the rotary viscous coefficient, $\dot{\omega}_n$ is the rotary acceleration of the roller, and $\omega_n$ is the rotary speed of the roller. The solution of this differential equation is

$$\omega_n = \omega_{n,0}e^{-\left(\frac{B_{fn}}{J_n}\right)t} \tag{5.10}$$

where $\omega_{n,0}$ is the value of $\omega_n$ when $t = 0$. This solution is the decreasing exponential shown by the blue dot-dashed curves ("viscous frict.") in Figure 5.6.

All of the experimental traces for each case in Figure 5.6 have an almost constant negative slope. That is, the deceleration rate is nearly constant. An appropriate model of this behavior is that rotary speed is a simple function of time, $\omega_n = -Kt$. In this equation, $\omega_n$, is the rotary speed of the roller, $K$ is a constant, and $t$ is time. This model best represents the spin down data. And, it is this behavior that suggests that the resisting torque in the spin down tests of the roller is due primarily to Coulomb friction rather than viscous friction in the bearings.

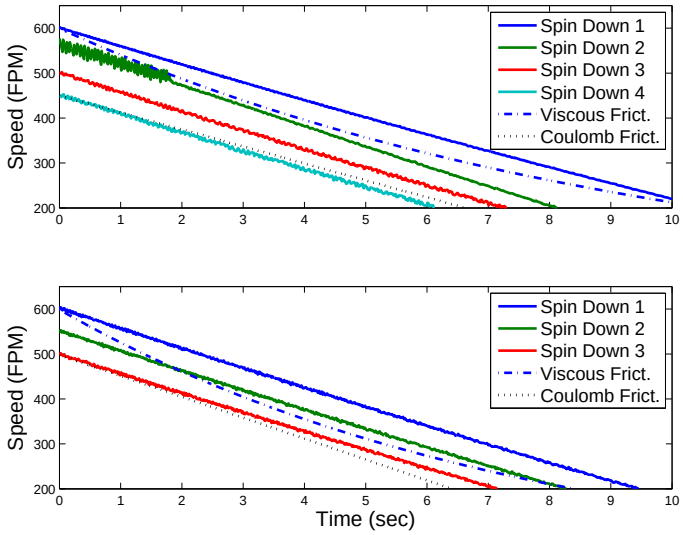



Figure 5.6: Spin-Down Test Speed Traces are Used to Calculate the Viscous Bearing Friction. The viscous friction simulation does not track the experimental data well for either the "cold" (top) or "warm" (bottom) bearings. A Coulomb friction model matches the deceleration rate reasonably well.

## 5.3 Simulation Results

The Sliding Friction Driven Roller (SFDR) model is used to simulate some of the runs from the experimental section. The results drove the surface finish and the bearing friction studies discussed above. Since the S-wrap section essentially isolates the unwind section from the process and rewind sections, the studies reported in the next two sections were limited to the unwind section. The Ducotey-Good traction model was applied to all the idlers in the web line simulation.

### 5.3.1 The SFDR Model with Additional Torque

The conditions of the Additional Torque experiment were simulated using both the Whitworth model and the SFDR model. The Whitworth model shows the limiting characteristic in Figure 5.7 when it (red line) drops about 50 FPM with 3.379 lbf simulated hanging weight while the experimental data shows that the idler was stopped with that level of hanging weight. The blue and green lines are the experimental data showing the no slip and slipping conditions. The SFDR model with 1.7 lbf hanging weight tracks up the ramp in Figure 5.8, but then shows slip as it settles in on a speed equal to the speed of the 2.245 lbf hanging weight experiment. The black line in Figure 5.8 shows the SFDR model with 2.515 lbf hanging weight simulated and it hits the experimental speed after slipping. These two weights bracket the beginning of slip in the experiment so the model showing slip is not terrible. The simulation data not following the experimental data with 1.7 lbf hanging weight in the model caused the testing of the coefficient of friction between web and roller and the bearing friction study to define parameters.

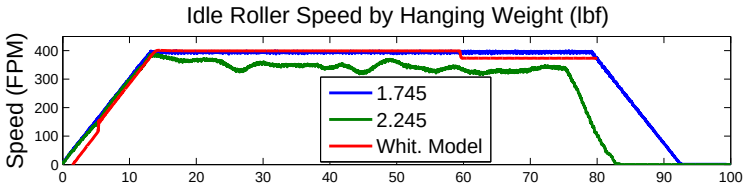



Figure 5.7: To Illustrate a Limit in the Whitworth Model, the Blue Line is Experimental Data with No-Slip, the Green is with Slip, and the Red Line is Simulation Data Using Only the Whitworth Model and 3.379 lbf Hanging Weight. The Whitworth model only allowed about a 50FPM drop in speed while the experimental data for that hanging weight was stopped.

### 5.3.2 Surface Finish and Bearing Friction Impacts

These experiments refined parameter values in the simulation models. There was a little simulation in the bearing friction study to cross-check the evaluated bearing friction with the experimental data. As Figure 5.6 showed with the dashed and dotted lines, the viscous model of friction does not follow the experimental data which lead to the creation of a Coulomb friction model for the Euclid Web Line's bearing friction.

Applying the results of the coefficient of friction study and the bearing friction study to the

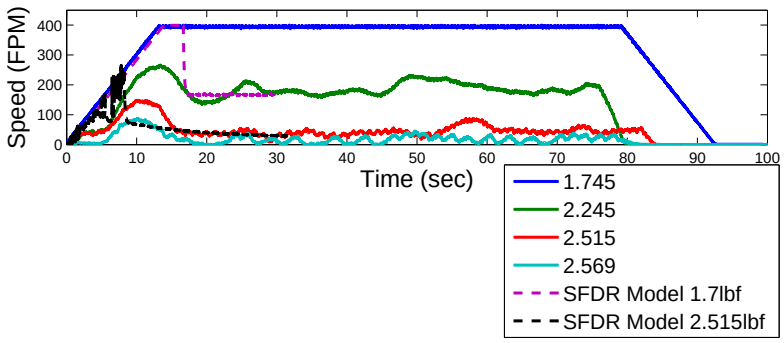



Figure 5.8: SFDR Model Plotted Against Experimental Data. The hanging weight in the simulation seems to impact when slip initiates and what speed is finally attained. At the lighter hanging weights, the SFDR model shows slip where there is not any, but as heavier hanging weights are applied, the SFDR does a good job of settling in on the slipping speed of the idler (see the black dotted line overlaying the red line).

simulation gives Figure 5.9, a simulation of a 0-400 FPM start up with a polished roller R9. On the left is the Whitworth model output for roller R9 speed and spans 8 and 9 tensions with 1.419 lbf hanging weight applied. The speed difference due to slip is unnoticeable. On the right, the SFDR model with the same set of parameters and hanging weight shows a 50 FPM average decrease in speed. For reference, see Figure 5.5 showing how the EWL decreased more than 50 FPM with 1.419 lbf hanging weight. The red dots in the bottom right plot show the tension level assuming full adhesion.

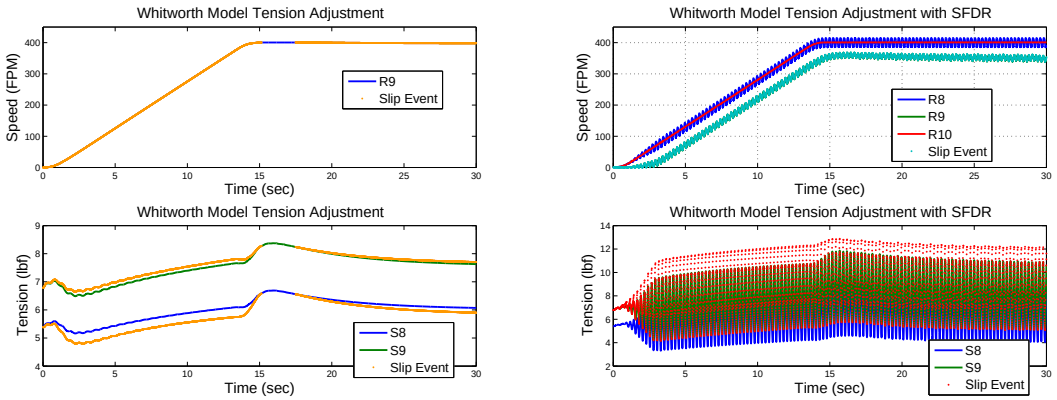



Figure 5.9: The Whitworth Model Shows Little Variation in Speed Due to Slip While the SFDR Model Predicts a 50 FPM Decrease in Speed with a 1.419 lbf Additional Torque Applied. The experimental data for this situation shows the idler speed dropping throughout the run.

## 5.4 Ducotey-Good Traction Model Simulation

The previous experiments and simulations were conducted under the assumption that slip would occur at speeds the experimental web line could not attain. That assumption is true as long as the investigation was focused on constant friction coefficient models for slip and rollers with near $90^\circ$ of wrap angle. The Whitworth criteria and model were used with constant coefficient of friction to check for slip during a startup for all idlers in the EWL and no slipping was found. The DG traction model decreases the coefficient of friction based on the air film thickness which means a variable coefficient of friction. The DG model was incorporated, validated, and parameters studies produced impacts estimated. A simulation using the DG model with estimated parameters indicated three rollers might slip. Experiments were accomplished to gather data from those three rollers and simulations were recorded and compared to the experimental data.

### 5.4.1 Validation and Parameter Studies

The Ducotey-Good (DG) traction model due to air entrainment from [38] was incorporated into the MATLAB model of the EWL. Simulations of the results in [38] were repeated to validate that the DG traction model was coded correctly. Then parameter studies of roller surface roughness, roller diameter, web tension, and web permeability, were accomplished with the DG model. The idle rollers on the Euclid Web line were purchased from Fife Corporation according to the drawings. Fife Corporation has a subsidiary that manufactures rollers now, and the surface finish on the rollers is $32$ $\mu$in normally. The range of surface roughness used in the simulation was from $8$ to $500$ $\mu$in due to surface impurities being present on the original rollers. The roller diameters were selected by the idle roller sizes on the $3$ web processing lines owned by the Web Handling Research Center (WHRC). The Tyvek material was sent to a lab for testing its coefficient of friction, permeability, tensile strength, and surface roughness. The Dupont website produced specification sheets on several types of Tyvek, but none of them could be directly linked to the Tyvek in use. The values from the specification sheets are used in the parameter studies along with values from [38]. TAPPI inspection procedures [79, 80] were studied to tie the listed values of from the specification sheets to parameters for the DG model. In the following parameter studies, the situation of idle

roller R9 on the Euclid Web line was used: $88^\circ$ wrap angle, $5$ lbf tension, $149.6$ $\mu$in web roughness, $3$ in diameter roller, $6.07$ in wide web, and $0.02828$ ft$^3$/s/(ft$^2$-psi) permeability.

Figure 5.10 shows the parameter study of the effect of web speed and roller roughness on coefficient of traction, $\mu_T$, for the parameters given in the figure. The theory used by the DG model is that an air film is created between the web and roller because air is being carried along beside the web due to viscosity. Where the web and roller meet is a converging nozzle which increases the air pressure locally and forces the web up off the surface of the roller and initial amount, $h_0$. The value of $h_0$ is not linear with web speed, but the limits are shown on the right hand axis of the figure to aid in understanding the physical mechanism that is causing the reduction in coefficient of traction. Higher roller surface roughness causes a slower decrease in the coefficient of traction because a larger air film has to be created to overcome the larger surface roughness.

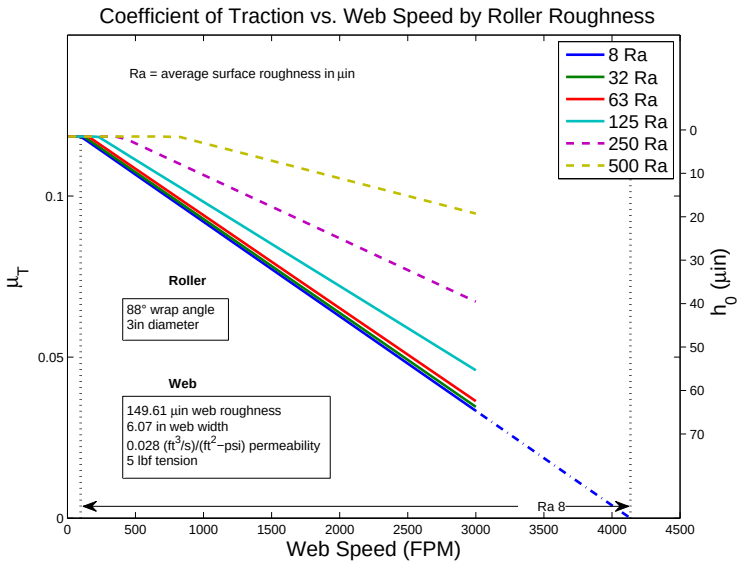



Figure 5.10: Ducoty-Good Traction Model Parameter Study of the Effect of Web Speed and Roller Roughness on the Coefficient of Traction. The black dotted lines indicate the limits of the transition from the full value of the coefficient of friction to 0 based on web speed for the 8 Ra roughness case. The other roughnesses would have different limits. The right hand vertical axis is the initial air film height estimate for 8 Ra roughness roller. Increasing web speed decreases the coefficient of traction and increases the air film height.

Figure 5.11 shows the parameter study of the effect of web speed and web permeability on $\mu_T$. The four permeability values are given in Table 5.1 are listed smallest to largest. From Figure 5.11 the relationship between increased permeability and smaller slope of decreasing coefficient of traction can be seen. This makes sense because increasing permeability means more air passes

Table 5.1: Air Permeability of Materials for Parameter Study Using Ducotey-Good Traction Model.

| Name | Permeability $ft^3/s/(ft^2\text{-}psi)$ |
|---|---|
| NP-3 | 2.828E-02 |
| Tyv 1443R | 4.166E-02 |
| NP-1 | 5.158E-02 |
| Tyv 1025D | 8.983E-02 |

through the web which decreases the air film height and the lower air film height means a larger coefficient of traction. Increasing speed continues to decrease coefficient of traction.

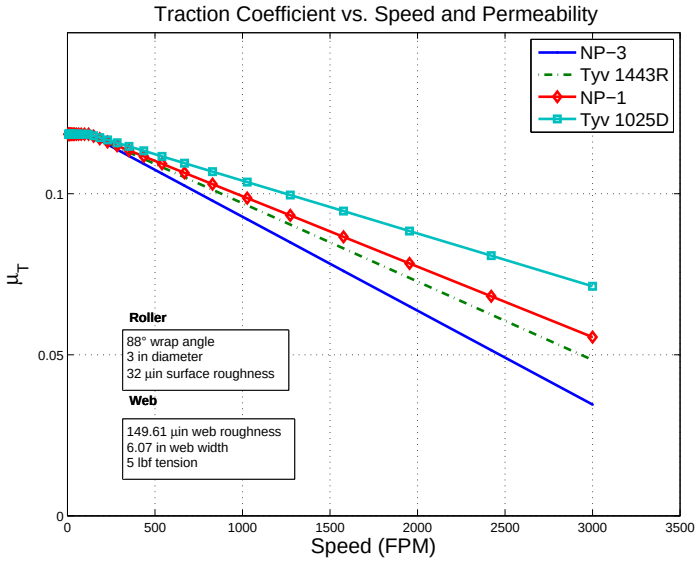



Figure 5.11: Ducotey-Good Model Traction Parameter Study of the Effect of Web Speed and Permeability on Coefficient of Traction. The permeability changes the slope of the line. The speed parameter is continuing to show that increasing speed decreases coefficient of traction.

Figure 5.12 shows the impact of air film thickness by roller diameter on coefficient of traction. The figure was originally versus web speed, but the lines were collinear. The coefficient of traction decreases as the air film thickness increases, but the roller diameter scales the air film height.

Figure 5.13 shows the effect of tension and speed on coefficient of traction. The coefficient of traction increases with increasing tension which makes sense because the tension is the force pushing the web down onto the roller against the pressurized air. Increasing speed decreases

traction by entraining more air. The key parameters from the parameter studies using the DG model are web speed, web tension, web permeability, and the roughness of the interface of the two materials. Roller diameter was shown to not have much effect on $\mu_T$. The coefficient of friction is an input to the DG model, but it is a property of the interface of the two materials involved and is not a controllable parameter. It can be affected by polishing the roller surface which also changes the surface roughness of the roller.

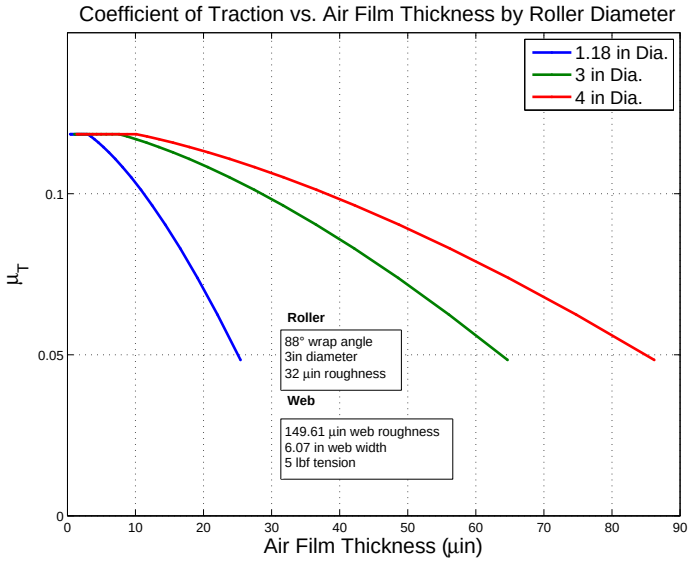



Figure 5.12: Ducotey-Good Traction Model Parameter Study Showing the Effect of Air Film Thickness by Roller Diameter on Coefficient of Traction. The coefficient of traction is not effected by diameter: the values pass through the same range for all three diameters. The initial air film height increases with increasing diameter.

Moving to a more specific parameter study, the $0 - 400$ FPM startup of the EWL was simulated with roller R20 as the focus using the Ducotey-Good traction model and the effects of web permeability, web roughness, and roller roughness were examined. Roller R20 was simulated in the polished condition. The line tension was set to 5 lbf. The default web parameters for the study are in Table A.1 in the Appendix. The default roller parameters for the study are in Table A.2 in the Appendix. In the following figures, the reference speed has been subtracted out of the roller speed giving $\Delta$Speed. The reference speed profile is indicated in Figure 5.17 by the blue box.

Figure 5.14 shows the effect of varying the web permeability on the speed of R20. The legend is listed in ascending order from zero permeability to the largest value simulated. The specific values are tabulated in Table 5.1. The effect to observe is that smaller permeabilities cause larger

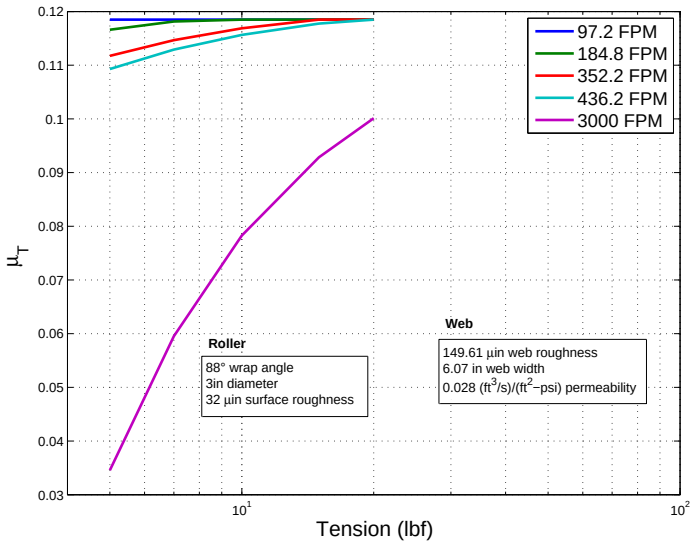



Figure 5.13: Ducotey-Good Traction Model Parameter Study of the Effect of Tension and Speed on Coefficient of Traction. The increase of tension increases the coefficient of traction while increasing speed decreases $\mu_T$.

delta speeds indicating that the air film is thicker and thus the traction coefficient is smaller. The maximum deviation from the reference speed is 6 FPM below. The other values group around 5 FPM below.

Figure 5.15 shows the effect of roller roughness on the delta speed of R20. Permeability of the web was set to that of Tyvek 1443R. The surface finish of the roller was simulated through a range from an average roughness of 8 $\mu$in to 500 $\mu$in. The smoothest surface finish (Ra 8) caused the largest deviation from the reference speed of about 5.5 FPM below. The largest surface roughness (Ra 500) caused the smallest deviation of about 4.7 FPM below the reference speed.

Figure 5.16 shows the effect of varying the web surface roughness. The roller surface roughness was set at Ra 63 or 63 $\mu$in. The three values of roughness come from Tyvek specification sheets for Tyvek 1025DR and Tyvek 1073D (Tyvek 1443R did not have a reported value) and the 40 $\mu$in value was to simulate a web roughness more like what a plastic film would have. The smallest roughness (Ra 40) caused the largest deviation from the reference speed of about 5-7 FPM below. The roughest value caused the smallest deviation. The Ra 149 and Ra 185 are for types of Tyvek, so it is reasonable to assume that the Tyvek used in the experimental sections of this research would have started with a surface roughness in the neighborhood of those two values.

The EWL was simulated using the Ducotey-Good model to identify rollers that could slip at

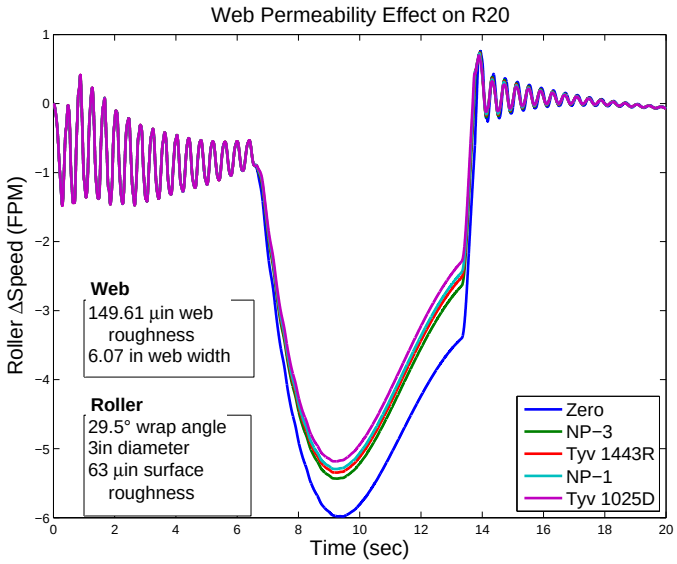



Figure 5.14: The Effect of Varying Web Permeability on Roller ΔSpeed. Notice that the zero web permeability line shows the largest impact to roller speed which is a delta of only 6 FPM.

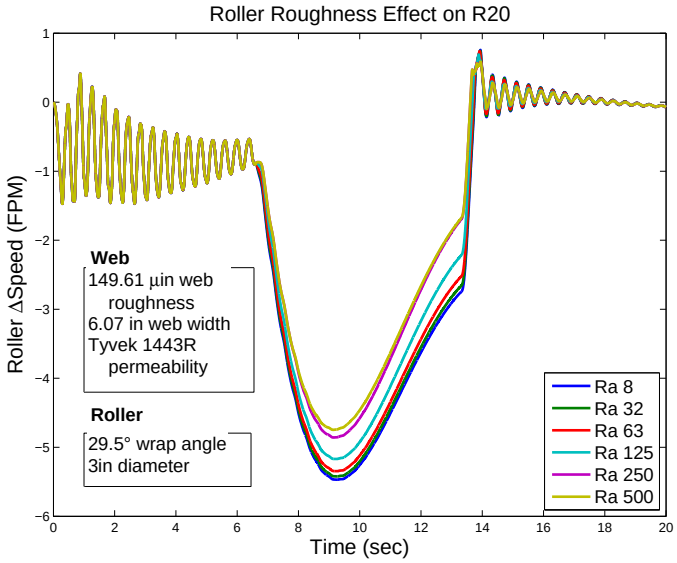



Figure 5.15: The Effect of Varying Roller Surface Roughness on ΔSpeed. Again, the smoothest surface finish had the larges effect on the roller speed which was a delta of -5.5 FPM.

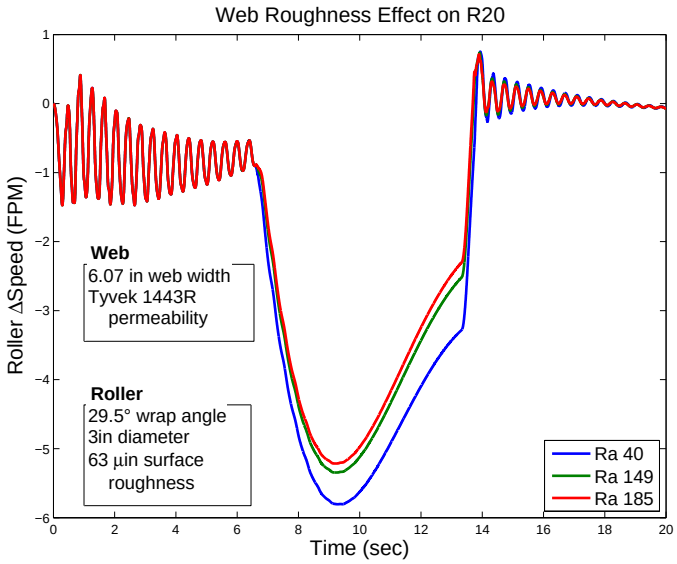



Figure 5.16: The Effect of Web Surface Roughness on $\Delta$Speed. The Ra 40 value was from a plastic film material while the other two were from Tyvek specification literature. Again, the smoother surface has the largest impact on speed.

speeds that the line could attain using worst case values for the parameters. The web permeability was set to zero. The roller roughness was set to Ra 8, and the web roughness was set to Ra 40. All the idle rollers were evaluated with DG model and the results of the simulation of a 0-400 FPM startup indicated that three rollers might slip. They were, referring to Figure 1.5, rollers R3, R12, and R20. Rollers R12 and R20 have wrap angles of $50^\circ$ or less unlike the $88^\circ$ wrap angle of roller R9.

## 5.5 Experimental Results following the Ducotey-Good Model Simulations

The EWL was instrumented with an encoder on three rollers: R3, R12, and R20, and the sequence of speeds shown in Figure 5.17 was followed. All the rates of change in the speed of the line were governed by the industrial S-curve mechanism for setting the reference speed. The sequence involved a 0-400 FPM start-up, a deceleration and two more accelerations, and finally, a shutdown. The sequence captures the same 0-400 FPM reference speed transition used in the simulation with the DG model. The tension level in the simulations and the experiment was 10 lbf. Simulations of

this experiment were run to show where the no slip condition and the DG model would have the three rollers' speeds.

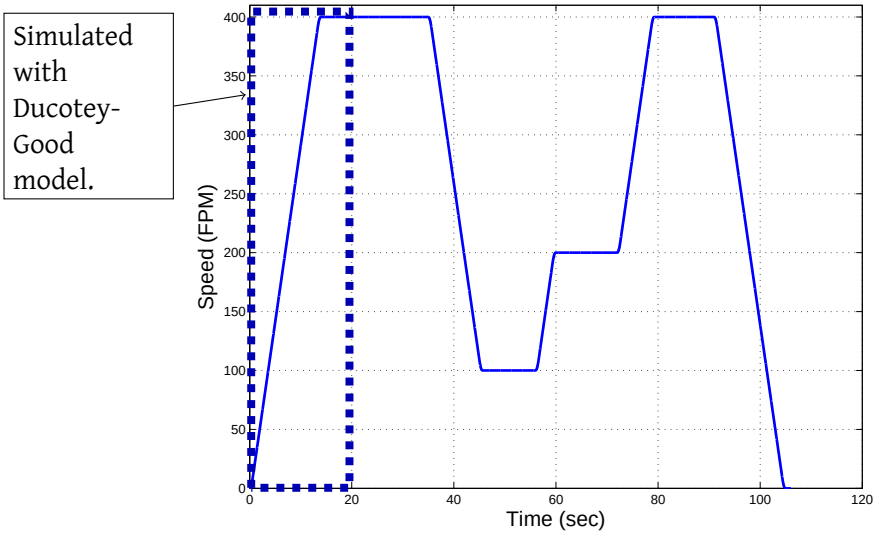



Figure 5.17: Reference Speed Profile for Slip Experiments Following the Ducotey-Good Slip Model Simulation. The speed transition that the simulation using the Ducotey-Good traction followed is boxed.

Figure 5.18 shows the experimentally recorded speed of rollers R3, R12, and R20. The top row is for roller R3, the middle row is for roller R12, and the bottom row is for R20. Roller R3 showed experimentally that it was running slower than expected, but not nearly as slow as the DG model predicted (on the right). This could be indicative of a rougher surface than the simulation assumed, based on the parameter study of roller roughness. Roller R12 showed a very similar behavior to roller R3. Again, roller roughness or web permeability could have been the cause of the difference between the predicted roller speed and the experimental roller speed. Roller R20 experimentally ran below the speed expected by the Ducotey-Good model. Roller R20 was slipping because it was recorded at speeds below the expected speed. Rollers R3 and 12 were also slipping because they were recorded at speeds less than the expected speeds for a no-slip situation, but not as slowly as was predicted for each roller by the DG model. The fact that the same web passed over all three rollers suggests that the roller surface finish is the culprit for the difference in experimental speed compared to predicted speed.

Roller R20 was selected to be modeled by the SFDR model. The simulation was a 0-400 FPM startup operation at 5 lbf tension. The EWL was operated at 5 lbf tension, the roller speed was

recorded with the encoder and a polished roller was used in the R20 position during the experiment. The speed reference sequence from Figure 5.17 was followed. Figure 5.19 shows the experimental speed of roller R20 and the speed of roller R20 from the two simulations using the DG traction model and the SFDR slip model, respectively. All three lines on the plot began by following the reference speed ramp. Then, just below 200 FPM, the SFDR model had a slip event, indicated by roller R20 slowing down, and then began to ramp up in speed a few seconds later. Around 200 FPM, the experimental speed dropped off to align with the SFDR model speed. The experimental speed of roller R20 accelerated back up to the DG model speed after a few seconds. At the top of the acceleration slope, the experimental roller speed dropped down to the SFDR model speed again, but then settled to a speed lower than either the DG model or the SFDR model expected. The Ducotey-Good model was closer to the experimental result in the steady-state than the SFDR model was.

Another way to view the speed transition is to remove the industrial S-curve which is the reference speed and look at the delta from the reference speed for each line: the experimentally recorded speed, the simulation using the Ducotey-Good model and the simulation using the SFDR model. Figure 5.20 shows the $\Delta$Speed plots of the no-slip condition, the DG traction model simulation, the SFDR model simulation, and the experimentally recorded speed of R20. Following the light blue line of the experimental data, the SFDR model is being tracked initially. Then the SFDR model slips and slows down. Shortly after that, the experimental speed drops off as well and aligns with the SFDR model speed again (at nearly 35 FPM below the reference speed). Then the experimental data shows that the roller accelerated up to the level being maintained by the Ducotey-Good model, around 10 seconds. The DG model begins to fall off in speed and the experimental data shows a similar, though smaller, drop in speed. Then, as the acceleration ramp ends around 14 seconds, the experimental data drops again. It aligns with the SFDR model speeds as that model is accelerating up to the steady-state right after 14 seconds. The experimental data shows a steady-state speed lower than either the DG model or the SFDR model, but closer to the DG model speed.

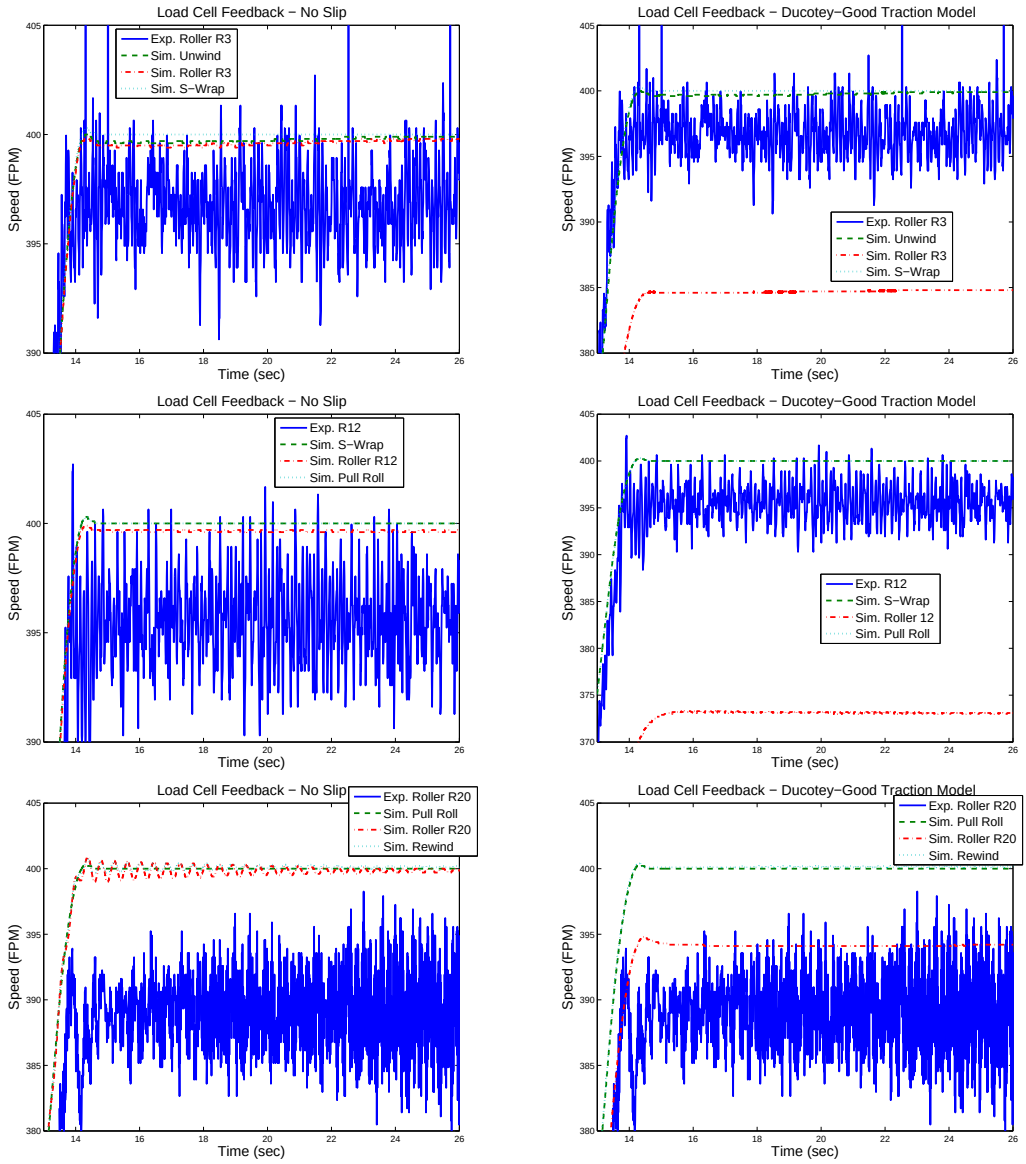



Figure 5.18: Simulations of Roller Speeds for Rollers 3, 12, and 20 (rows, dashed red line) Along with Speed Recorded from the Encoder (solid blue line). On the left for each row, the figure shows simulated roller speed assuming a no slip condition and on the right the Ducotey-Good model simulation speed is shown. Each plot has the driven roller speeds that surround the roller of interest shown, as well. All three rollers are slipping in the sense that the experimental speed is below the no slip condition speed.

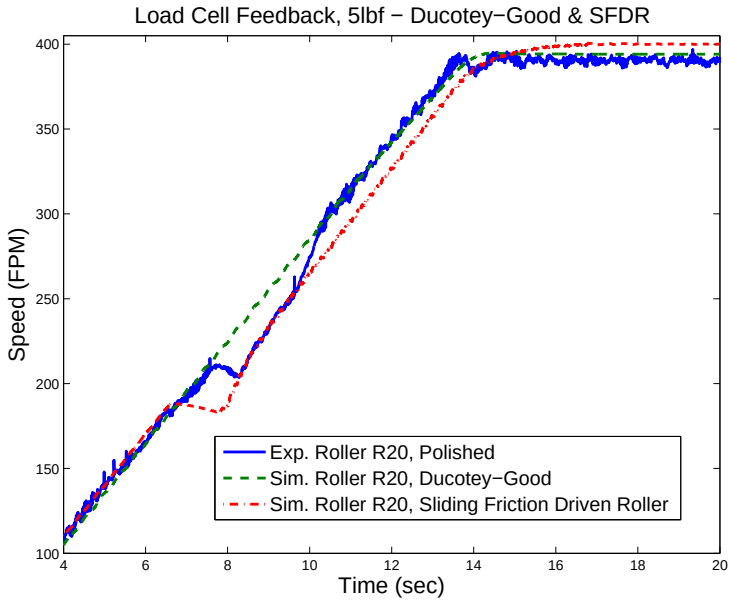



Figure 5.19: Roller R20 Speed from Experiment with a Polished Roller at 5 lbf Tension and Two Simulations Using the Ducotey-Good Traction Model and the SFDR Slip Model. The experiment shows the roller tracking the Ducotey-Good model until 210 FPM where it drops off to the speed simulated by the SFDR model. Then the experimental speed accelerates back up to the Ducotey-Good model speed between 250 and 300 FPM.

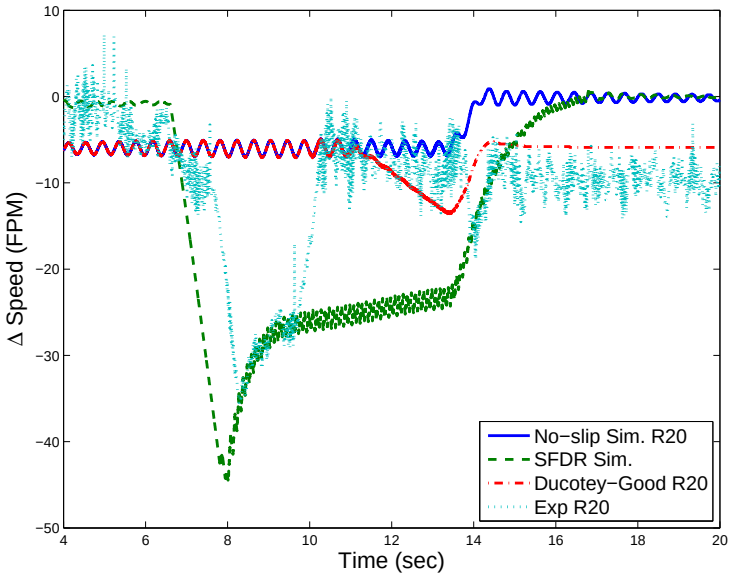



Figure 5.20: The $\Delta$Speed is the Difference from the Reference Speed of each Simulation of Roller R20 and the Experimentally Recorded Speed of R20. During the acceleration, the roller drops to the speed predicted by the SFDR model but at the steady-state at 400 FPM, the roller settles at a lower speed than predicted by either the SFDR model or the Ducotey-Good traction model.

## 5.6 Summary

This research has shown the required differential equations for simulating the Euclid Web Line. Three slip models are discussed, (i) Whitworth's model and criteria, (ii) the Sliding Friction Driven Roller (SFDR) model, and (iii) the Ducotey-Good (DG) traction model. An experiment was conducted that involved adding an additional torque to a roller in order to create a slip condition while running the EWL at speeds that were achievable. Physical parameter values for bearing friction and coefficient of friction between web and roller were determined by experiment. A second experiment was conducted to record the roller speed of three rollers during a start-up sequence.

Comparison of the simulations with results from the first experimental study showed that the Whitworth model was valid only when the additional torque was small. In contrast, the experimental study showed that the SFDR model was valid only when the additional torque was large. Comparison of the second experiment with simulations using the DG model showed that air entrainment may be affecting the rollers and last experiment compared to the DG model and the SFDR model showed that the roller slipped and acted like both models at different times during the acceleration ramp. This indicates that both models are needed to accurately simulate a web line, but also that different phenomena occur between the web and roller under the varying conditions of a start-up sequence.

The Whitworth model covers the initiation of slip while the SFDR model covers the slip situation where the roller speed is distinctly different than the web speed. The Ducotey-Good model is in the middle. A unified model is needed to cover the entire spectrum of slip. A unified model of slip was deemed beyond the scope of this work on modeling longitudinal dynamics of a web line.

# CHAPTER VI

# CONCLUSION

Modeling and simulation of web handling systems and a method of determining controller gains is the focus of this dissertation. Proportional–Integral (PI) control is used extensively in industry which drives its use in the simulations and in the experiments on web lines in this research. The simulations in this research are composed such that the gain values used in a simulation can be used directly on a web line. The simulation also allows evaluation of the Rockwell method of controller gain calculation on the web lines used herein. The web lines used in this research are part of the Web Handling Research Center and are described in in chapter 1. An abbreviated history of modeling web lines is also given with introduction to primitive elements, both linearized and nonlinear, and control schemes.

The primitive elements used to build up the web line roller by span are first-principles, ideal models of the real elements. The ideal nature and the assumptions used to derive the models limit the variability of the speed and tension response more than their non-ideal, physical counterparts. The primitive element models are described in chapter 2. Models include ones from the literature as well as three derived for this research: the online non-circular roll, pendulum dancer with nonlinear span length effects, and the S-wrap dancer with nonlinear span length effects. The models are grouped by being linearized or nonlinear and the importance of the nonlinear primitive elements are shown through simulation of a disturbance on the unwind of the Euclid Web Line and compared to measured data. Simulations with nonlinear primitive elements capture more of the response in tension caused by the disturbance than the linearized primitive elements do.

The two prerequisites necessary to calculate controller gains are the performance criteria and

the method of gain calculation. In chapter 3, the second-order system performance characteristics of damping ratio and rise time and an additional goal of limited steady-state error for the High-Speed Web line are used to define the performance goals. The gains are determined by modeling the open-loop systems being controlled and solving for the gains that bring the desired performance criteria. The control structure studied in this research is the speed-based tension control, a cascaded control. The inner speed loop gains are easily solved for, but the outer tension loop has too high an order for the second-order performance criteria to apply. The the open-loop tension model's transfer function is approximated by a reduced order transfer via Hutton's Routh Approximation Method. Then the gains that provide the second-order performance goals can be determined.

Experiments on the High-Speed Web Line using the gains calculated by the Routh Approximation Method verify the method's ability. The RA method appears to be as good if not better than the Rockwell method. Simulation is the synergy of modeling, process timing, and feedback working together. The output of simulation can be directly used in a controller as is the case with model reference adaptive control or it can be used offline for planning and design changes. The second case is the focus of the second half of chapter 4. The culmination is a set of outputs that allows analysis of existing web lines and theoretical ones. The High-Speed Web Line has been simulated using the initial conditions of several experiments. The simulation results are compared to the experimental results to provide a level of trust in the model underlying the simulation. The simulation is used to study the effects of moving the unwind roll farther away from the feedback load cell and then to study the effect of changing the feedback device to a translational dancer from a load cell.

The problem of slip between a web and roller is considered in chapter 5 where experiments and simulations are used to gain insight into the mechanism. The Sliding-Friction Driven Roller model was developed for modeling slip. Comparison of the simulations with results from experiments show that the Whitworth model (from the literature) is valid only when the additional torque was small. In contrast, the experimental study shows that the Sliding-Friction Driven Roller model was valid only when the additional torque was large. Comparison of the second experiment with sim-

ulations using the Ducotey-Good Traction model showed that air entrainment may be affecting the rollers and last experiment compared to the Ducotey-Good Traction model and the Sliding-Friction Driven Roller model shows that the roller slipped and acted like both models at different times during the acceleration ramp. This indicates that both models are needed to accurately simulate a web line, but also that different phenomena occur between the web and roller under the varying conditions of a start-up sequence. The Whitworth model covers the initiation of slip while the Sliding-Friction Driven Roller model covers the slip situation where the roller speed is distinctly different than the web speed. The Ducotey-Good model is in the middle. The finding from this research is that a unified model is needed to cover the entire spectrum of the slip condition. A unified model of slip was deemed beyond the scope of this work on modeling longitudinal dynamics of a web line.

The research has focused on modeling web lines and a systematic method for determining controller gains for web handling systems. The following are topics that could extend this work:

- In both the Euclid and High-Speed Web Lines, the diameter calculation is calculated following the method described in section 2.2.2. This method works well in steady-state conditions, but has evident problems in the start-up regime. As noted in this research, a difference of $0.04$ of an inch contributed to a more than $3$ FPM difference in tangential speed. Also, this method will give incorrect results for situations where the web path length is variable, like in the case of dancer feedback. Kalman filtering or a model reference for measurement correction could be explored.

- The performance goals used in the Routh Approximation Method for determining controller gains used rise time and damping ratio which are descriptors for second-order dynamic systems. Changing the set of performance goals to allow for selecting real poles and zeros for the transfer function instead of the imaginary poles and zeros currently being used may improve performance of the tension control system even more.

- The Routh Approximation Method of gain selection creates a pole at $s = -k_r \zeta \omega_{nat}$. Further study is needed in the interaction of that pole and the tension specification multiplier, $k_{fvt}$.

- The spindle of the High-Speed Web Line's unwind motor appears to have eccentricity. A parameter study could be accomplished with simulation and measurements of the actual runout to quantify the impact the eccentricity has on web tension.

- The webs in this study were all high Young's modulus, elastic materials. Materials with low modulus are used in the clothing and tissue industries and have to be transported under high strains which are generally beyond the elastic limit of the material. The Narrow Tyvek experiments in this study are[1] in this category of being transported under high levels of strain in its inelastic region.

- The High-Speed Web Line was configured to use the Bypass path in this research. What was bypassed was the process section of the High-Speed Web Line. Research in comparing the tension control in the process section of the web line between the Routh Approximation Method of gain calculation and the Rockwell method is needed. A well defined process section with tension measurement and control is needed.

- The High-Speed Web Line used lead-lag filters for conditioning the feedback signal that the PI controller reacted. The performance criteria of tension oscillation from the set point being less than 5% was not met in experiments. The unmet performance criteria suggests a different sort of filter is required. A pure lag filter may be a good option.

- The High-Speed, Low-Tension web line was discussed in the introduction, but was not part of this research. It is used, as the name suggests, with low tension applications like melt-blown, spun-bond materials which are stretchy and have inconsistent density. The line also has an accumulator for zero-speed splice research. The accumulator is a primitive element that could by modeled in more in-depth ways than the literature shows.

- This research found that a unified model of slip is necessary. The experiment from this research showed that the measured roller speed transitioned from the Sliding-Friction Driven Roller model response to the Ducotey-Good Traction model speed response. A unified model

[1] Those experiments had the 6 inch wide Tyvek under 4 pounds per linear inch (pli) of tension which is just past the elastic region on the stress-strain plots from Dupont.

of slip could incorporate Whitworth's criteria for slip, Whitworth's slip model, the Sliding-Friction Driven Roller model, and the Ducotey-Good Traction model. The Whitworth model and criteria describes the initiation of slip well. That will continue to be used for the combined model. The Sliding-Friction Driven Roller model assumed a constant coefficient of friction in this research. A method to combine the effects of the Ducotey-Good Traction model with the Sliding-Friction Driven Roller model is to have the Ducotey-Good method calculate the coefficient of traction and use that in the Sliding-Friction Driven Roller model instead of the constant value originally used.

- The situations simulated in this study do not allow for slack to occur. Deriving a differential span tension equation that tracked slackness instead of rendering negative tensions, which is what the span tension model used in this study would do, would give rise to more realism in the simulation especially in the modeling of an accumulator. The control loop could be tested for its response to situations where slack occurred.

# REFERENCES


[1] Siemens Product Lifecycle Management Software, I., 2018. Siemens plm software: Key simulation enablers. ebook, Plano, TX. www.siemens.com/plm.

[2] Damour, J., and Gap, W., 2010. The mechanics of tension control. Online, October. Converter Accessory Coporation, Wind Gap, PA.

[3] Pagilla, P. R., 2017. Web handling seminar. Printed by the Oklahoma State University Web Handling Research Center, March.

[4] Pagilla, P. R., Siraskar, N. B., and Dwivedula, R. V., 2006. "Decentralized control of web processing lines". *IEEE Transactions on control systems technology,* **15**(1), pp. 106–117.

[5] Pagilla, P. R., Dwivedula, R. V., and Siraskar, N. B., 2007. "A decentralized model reference adaptive controller for large-scale systems". *IEEE/ASME Transactions on mechatronics,* **12**(2), pp. 154–163.

[6] Pagilla, P. R., and Siraskar, N. B., 2006. "Robust controllers for large-scale interconnected systems: applications to web processing machines". In *Current Trends in Nonlinear Systems and Control*. Springer, pp. 387–406.

[7] Reid, K. N., Shin, K.-H., and Lin, K.-C., 1993. "Variable-gain control of longitudinal tension in a web transport system". *ASME APPLIED MECHANICS DIVISION-PUBLICATIONS-AMD,* **149**, pp. 87–100.

[8] Reid, K. N., and Shin, K. H., 1991. "Variable-gain control of longitudinal tension in a web transport system". In WHRC [81], pp. 220–233. https://shareok.org/handle/11244/320247.

[9] Pagilla, P. R., King, E. O., Dreinhoefer, L. H., and Garimella, S. S., 2000. "Robust observer-based control of an aluminum strip processing line". *IEEE Transactions on Industry Applications,* **36**(3), pp. 865–870.

[10] Pagilla, P. R., Garimella, S. S., Dreinhoefer, L. H., and King, E. O., 2001. "Dynamics and control of accumulators in continuous strip processing lines". In IEEE Transactions on Industry Applications, Vol. 37, No. 3, IEEE, pp. 934–940.

[11] Pagilla, P. R., 2017. "Modeling and computer simulation of zero speed splice unwinds". In WHRC [82]. https://shareok.org/handle/11244/320247.

[12] Pagilla, P. R., and Diao, Y., 2011. "Resonant frequencies in web process lines due to idle rollers and spans". *Journal of Dynamic Systems, Measurement, and Control,* **133**(6), pp. 1018–1027.

[13] Pagilla, P. R., Singh, I., and Dwivedula, R. V., 2003. "A study on control of accumulators in web processing lines". In American Control Conference, 2003. Proceedings of the 2003, Vol. 5, IEEE, pp. 3684–3689.

[14] Campbell, D. P., 1958. *Dynamic Behavior of the Production Process: Process Dynamics*. John Wiley & Sons, Inc., New York, NY. https://hdl.handle.net/2027/uc1.b3729022.

[15] King, D., 1969. "The mathematical model of a newspaper press". *Newspaper Techniques,* **1**, pp. 3–7.

[16] Grenfell, K., 1964. "Tension control paper-making and converting machinery". *IEEE Transactions on Applications and Industry,* **83**(73), pp. 234–240.

[17] Brandenburg, G., 1976. "New mathematical models for web tension and register error". In Proc. 3rd International IFAC Conf. on Instrumentation and Automation in the Paper, Rubber, and Plastics Industries, Vol. 1, IFAC, IFAC, pp. 411–438.

[18] Shelton, J. J., 1986. "Dynamics of web tension control with velocity or torque control". In American Control Conference, 1986, IEEE, pp. 1423–1427.

[19] Shin, K.-H., 1991. "Distributed control of tension in multi-span web transport systems". PhD thesis, Oklahoma State University. http://hdl.handle.net/11244/20737.

[20] Reid, K. N., and Lin, K.-C., 1993. "Control of longitudinal tension in multi-span web transport systems during start-up". In Proceedings of the second international conference on web handling, WHRC, ed., Vol. 2, Web Handling Research Center (WHRC), Oklahoma State University, pp. 77–95. https://shareok.org/handle/11244/320247.

[21] Dorf, R., 2011. *Modern Control Systems*. Pearson Prentice Hall, Upper Saddle River, N.J.

[22] Shelton, J. J., 1999. "Limitations to sensing of web tension by means of roller reaction forces". In WHRC [83], pp. 105–124. https://shareok.org/handle/11244/320247.

[23] Dwivedula, R. V., and Pagilla, P. R., 2005. "Modeling of web slip on a roller and its effect on web tension dynamics". In Proc. of the ASME International Mechanical Engineering Congress and Exposition, ASME, pp. 1–8. IMECE2005-81501.

[24] Young, G. E., and Reid, K. N., 1993. "Lateral and longitudinal dynamic behavior and control of moving webs". *Journal of dynamic systems, measurement, and control,* **115**, pp. 309–317.

[25] Carlson, D. H., 2001. "Considerations in the selection of a dancer or load cell based tension regulation strategy". In PROCEEDINGS OF THE SIXTH INTERNATIONAL CONFERENCE ON WEB HANDLING, WHRC, ed., Vol. 6, The Web Handling Research Center, Oklahoma State University. https://webhandling.okstate.edu.

[26] Raul, P. R., 2015. "Design and analysis of feedback and feedforward control systems for web tension in roll-to-roll manufacturing". PhD thesis, Oklahoma State University, Stillwater, OK.

[27] Dwivedula, R. V., Zhu, Y., and Pagilla, P. R., 2006. "Characteristics of active and passive dancers: A comparative study". *Control Engineering Practice,* **14**(4), pp. 409–423.

[28] Pagilla, P. R., Dwivedula, R. V., Zhu, Y., and Perera, L. P., 2003. "Periodic tension disturbance attenuation in web process lines using active dancers". *Journal of Dynamic Systems, Measurement, and Control,* **125**, September, pp. 361–371.

[29] Martin, J. R., 1973. "Inertia compensated dancing roller". *Penrose Annual,* **66**, pp. 209–214.

[30] Martin, J. R., Cederholm, R., and Curtin, L. E., 1990. Inertia compensated festoon assembly, Apr. 10. US Patent 4,915,282.

[31] Fox, S. J., and Lilley, D. G., 1991. "Computer simulation of web dynamics". In WHRC [81], pp. 270–290. https://shareok.org/handle/11244/320247.

[32] Brown, J. L., 1999. "Propagation of longitudinal tension in a slender moving web". In WHRC [83]. https://shareok.org/handle/11244/320247.

[33] Branca, C., Pagilla, P. R., and Reid, K. N., 2013. "Governing equations for web tension and web velocity in the presence of nonideal rollers". *Journal of Dynamic Systems, Measurement, and Control,* **135**(1), pp. 11–18.

[34] Boulter, B., 1999. "Estimating modulus of elasticity, torque loss, and tension using an extended kalman filter". In WHRC [83], pp. 177–194. https://shareok.org/handle/11244/320247.

[35] Kuhm, D., and Knittel, D., 2012. "New mathematical modelling and simulation of an industrial accumulator for elastic webs". *Applied Mathematical Modelling,* **36**(9), pp. 4341–4355. http://www.sciencedirect.com/science/article/pii/S0307904X11007487.

[36] Gassman, V., and Knittel, D., 2011. "Robust PI-LPV tension control with elasticity observer for roll-to-roll systems". In Proceedings of the 18th World Congress of the International Federation of Automatic Control, IFAC, pp. 8639–8644.

[37] Rice, B. S., Cole, K. A., and Müftü, M. S., 1999. "An experimental and theoretical study of web traction over a nonvented roller". In WHRC [83]. https://shareok.org/handle/11244/320247.

[38] Ducotey, K. S., and Good, J. K., 1999. "Predicting traction in web handling". *Transactions-American Society of Mechanical Engineers Journal of Tribology,* **121**, pp. 618–624.

[39] Ducotey, K. S., and Good, J. K., 1995. "The importance of traction in web handling". *Journal of Tribology,* **117**(4), pp. 679–684.

[40] Ducotey, K. S., and Good, J. K., 1998. "The effect of web permeability and side leakage on the air film height between a roller and web". *Transactions-American Society of Mechanical Engineers Journal of Tribology,* **120**, pp. 559–565.

[41] Blevins, R. D., 1992. *Applied Fluid Dynamics Handbook.* Krieger Pub. Co, Malabar, Fla.

[42] Ducotey, K. S., and Good, J. K., 2000. "A numerical algorithm for determining the traction between a web and a circumferentially grooved roller". *Journal of Tribology,* **122**(3), pp. 578–584.

[43] Schüler, D., Welp, E. G., and Kopp, O., 1999. "Closed solid-state-fluid mechanical model for calculating the transferable torque on wrapped rolls". In WHRC [83]. https://shareok.org/handle/11244/320247.

[44] Brandenburg, G., 1972. "The dynamics of elastic webs threading a system of rollers". *Newspaper Techniques: the monthly publication of the INCA-FIEJ Research Association*, September, pp. 12–25.

[45] Whitworth, D. P. D., 1979. "Tension variations in pliable material in production machinery". PhD thesis, Loughborough University of Technology. https://dspace.lboro.ac.uk/dspace-jspui/handle/2134/8427.

[46] Whitworth, D. P. D., and Harrison, M., 1983. "Tension variations in pliable material in production machinery". *Applied Mathematical Modelling,* **7**(3), pp. 189–196.

[47] Mathur, P. D., and Messner, W. C., 1998. "Controller development for a prototype high-speed low-tension tape transport". *IEEE Transactions on Control Systems Technology,* **6**(4), pp. 534–542.

[48] Maciejowski, J. M., 1991. *Multivariable Feedback Design.* Addison-Wesley, Reading, MA.

[49] Kuhm, D., Knittel, D., and Bueno, M.-A., 2012. "Robust control strategies for an electric motor driven accumulator with elastic webs". *ISA transactions,* **51**(6), pp. 732–742.

[50] Oedl, G., 2017. "Calculation of film tension from film position and roller position". In WHRC [82], pp. 1–9. https://hdl.handle.net/11244/322056.

[51] Gassman, V., and Knittel, D., 2008. "$H_\infty$-based PI-observers for web tension estimations in industrial unwinding-winding systems". In Proceedings of the 17th World Congress of The International Federation of Automatic Control, IFAC, pp. 1018–1023.

[52] Gassman, V., Knittel, D., Pagilla, P. R., and Bueno, M.-A., 2012. "Fixed-order $H_\infty$ tension control in the unwinding section of a web handling system using a pendulum dancer". In IEEE Transactions on Control Systems Technology, Vol. 20, No. 1, IEEE, pp. 173–180.

[53] Branca, C., Pagilla, P. R., and Reid, K. N., 2013. "On the governing equation for web tension with out-of-round rolls". In Proc. of the Twelfth International Conference on Web Handling, Web Handling Research Center, Oklahoma State University.

[54] Abjadi, N. R., Askari, J., and Soltani, J., 2008. "Nonlinear decoupled control for multi-motors web winding system using the sliding-mode technique". In Networking, Sensing and Control, 2008. ICNSC 2008. IEEE International Conference on, IEEE, pp. 212–216.

[55] Muthukumar, N., Srinivasan, S., Ramkumar, K., Kannan, K., and Balas, V. E., 2016. "Adaptive model predictive controller for web transport systems". *Acta Polytechnica Hungarica,* **13**(3), pp. 181–194.

[56] Raul, P. R., and Pagilla, P. R., 2015. "Design and implementation of adaptive pi control schemes for web tension control in roll-to-roll (r2r) manufacturing". *Isa Transactions,* **56**, pp. 276–287.

[57] Chang, P. H., and Jung, J. H., 2008. "A systematic method for gain selection of robust pid control for nonlinear plants of second-order controller canonical form". *IEEE Transactions on Control Systems Technology,* **17**(2), pp. 473–483.

[58] Boulter, B. T., 2001. "Applying drive performance specifications to systems applications. I. speed performance". *IEEE Transactions on Industry Applications,* **37**(4), pp. 1082–1087.

[59] Boulter, B. T., 2001. "Applying drive specifications to systems applications. II. current/torque regulation". In Conference Record of the 2001 IEEE Industry Applications Conference. 36th IAS Annual Meeting (Cat. No.01CH37248), Vol. 36, Industry Applications Conference, IEEE, pp. 1–6. doi:10.1109/IAS.2001.955697.

[60] Boulter, B. T., 2002. "Applying drive specifications to systems applications. III. position regulation". In Conference Record of the 2002 IEEE Industry Applications Conference. 37th IAS Annual Meeting (Cat. No.02CH37344), Vol. 37, Industry Applications Conference, IEEE, pp. 1–6. doi:10.1109/IAS.2002.1044082.

[61] Boulter, B. T., 1997. "The effect of speed loop bandwidths and line-speed on system natural frequencies in multi-span strip processing systems". In IAS'97. Conference Record of the 1997 IEEE Industry Applications Conference Thirty-Second IAS Annual Meeting, Vol. 3, IEEE, pp. 2157–2164.

[62] Cowern, E., 2017. Baldor basics: Understanding torque. Tech. rep., Power Transmission Engineering, Apr. www.powertransmission.com.

[63] Chan, T.-F., and Shi, K., 2011. *Applied Intelligent Control of Induction Motor Drives*. IEEE Press John Wiley & Sons (Asia) Pte Ltd, Piscataway, NJ & Singapore.

[64] Koç, H., Knittel, D., Mathelin, M. D., and Abba, G., 2002. "Modeling and robust control of winding systems for elastic webs". *IEEE Transactions on control systems technology,* **10**(2), pp. 197–208.

[65] Palm, W., 2014. *System Dynamics*. McGraw-Hill, New York, NY.

[66] Newton, J. P., and Lin, K. C., 1997. Web transport system for windows - v2.1. Executable. The Web Handling Research Center.

[67] The Mathworks, I., 2020. Lead-lag (discrete or continuous): Discrete-time or continuous-time lead-lag compensator. Online. https://www.mathworks.com/help/physmod/sps/ref/leadlagdiscreteorcontinuous.html.

[68] IEEE, 2016. IEEE recommended practice for excitation system models for power system stability studies. Tech. rep., IEEE Std 421.5-2016 (Revision of IEEE Std 421.5-2005).

[69] Hutton, M., and Rabins, M. J., 1975. "Simplification of high-order mechanical systems using the routh approximation". *Journal of Dynamic Systems, Measurement, and Control,* **97**(4), December, pp. 383–392.

[70] Hutton, M., and Friedland, B., 1975. "Routh approximations for reducing order of linear, time-invariant systems". *IEEE Trans. on Automatic Control,* **AC–20**(3), June, pp. 329–337.

[71] Routh, E. J., 1877. "A treatise on the stability of a given state of motion, particulary steady motion". PhD thesis, University of Cambridge, London. http://hdl.handle.net/2027/hvd.32044080808934.

[72] Ogata, K., 1995. *Discrete-time control systems*. Prentice Hall, Upper Saddle River, N.J.

[73] Hong, H., Li, Z., Fan, S., Zhou, Q., and Fan, D., 2013. "Multi-rate tracking controller analysis and design for target tracking systems". *Journal of Central South University,* **20**(11), pp. 3049–3056.

[74] Salt, J., and Albertos, P., 2000. "Multirate controllers design by rate decomposition". In Proceedings of the 39th IEEE Conference on Decision and Control (Cat. No. 00CH37187), Vol. 5, IEEE, pp. 4895–4900.

[75] Shampine, L. F., and Reichelt, M. W., 1997. "The MATLAB `ode` suite". *SIAM Journal on Scientific Computing,* **18**(1), pp. 1–22.

[76] Reish, B., Abudia, M., and Reid, K., 2017. "Disturbance rejection in a web transport system using a pendulum dancer". In AIMCAL R2R Conference USA 2017, The Association of International Metallizers, Coaters and Laminators (AIMCAL), pp. 1–9.

[77] The MathWorks, Inc., 2013. Matlab. Natick, Massachusetts, United States.

[78] Lynch, R., 2018. Private communications. unpublished.

[79] Walkinshaw, J., 2006. *Air Resistance of Paper (Gurley method) TAPPI T 460*. TAPPI Journal, Peachtree Corners, GA.

[80] TAPPI, 1997. *Roughness of paper and paperboard (Print-surf method) T 555*. TAPPI Press, Peachtree Corners, GA.

[81] WHRC, ed., 1991. PROCEEDINGS OF THE FIRST INTERNATIONAL CONFERENCE ON WEB HANDLING, Vol. 1, Web Handling Research Center, Oklahoma State University. https://shareok.org/handle/11244/320247.

[82] WHRC, ed., 2017. PROCEEDINGS OF THE FOURTEENTH INTERNATIONAL CONFERENCE ON WEB HANDLING, Vol. 14, Web Handling Research Center, Oklahoma State University. https://shareok.org/handle/11244/320247.

[83] WHRC, ed., 1999. PROCEEDINGS OF THE FIFTH INTERNATIONAL CONFERENCE ON WEB HANDLING, Vol. 5, Web Handling Research Center, Oklahoma State University. https://shareok.org/handle/11244/320247.

[84] WHRC, ed., 2019. PROCEEDINGS OF THE FIFTEENTH INTERNATIONAL CONFERENCE ON WEB HANDLING, Vol. 15, Web Handling Research Center, Oklahoma State University. https://shareok.org/handle/11244/320247.

[85] Cooley, J. W., and Tukey, J. W., 1965. "An algorithm for the machine calculation of complex Fourier series". *Mathematics of Computation,* **19**(90), pp. 297–301.

[86] Chapra, S., and Canale, R. P., 2002. *Numerical methods for engineers: with software and programming applications*. McGraw-Hill Higher Education, Boston.

[87] Reish, B. D., and Reid, K. N., 2019. "Controller gain scheduling in a multi-span web line". In WHRC [84], pp. 1–20. https://hdl.handle.net/11244/320343.

[88] Reish, B. D., and Reid, K. N., 2019. "Systematic method for determining the controller gains in a multi-span web line". In WHRC [84], pp. 1–18. https://hdl.handle.net/11244/320342.

# APPENDICES

# APPENDIX A

# TABLES OF PHYSICAL PARAMETERS OF WEB LINES

The parameter values in Tables A.1, A.2, A.3, A.4, A.5, and A.6 are specific to the Euclid Web Line (EWL) owned by the WHRC in Stillwater, OK. The values are either measured or taken from the drawings of the line. Tables A.7, A.8, A.9 and A.10 are specific to the High-Speed Web Line (HSWL) also owned by the WHRC. Again the values are either measured or taken from the manufacturer's drawings. Table A.11 contains Tyvek properties received from DuPont[1] after testing a sample from the roll used in this study.

Table A.1: Web Properties for the Start-up Parameter Study

| Parameter | Value | $Units$ |
|---|---|---|
| Permeability | 0 | $ft^3/s/(ft^2 - psi)$ |
| Roughness (Ra) | 149.61 | $\mu$in |
| RMS Roughness (Rq) | 164.51 | $\mu$in |

[1]Thanks to Tom Manning, Kelsey Chung, and Tom Estep from Dupont for these test results.

Table A.2: Roller Properties for the Start-up Parameter Study

| Parameter | Value | $Units$ |
|---|---|---|
| Radius | 1.5 | $in$ |
| Wrap angle | $29.59°$ | |
| Roughness (Ra) | 63 | $\mu$in |
| RMS Roughness (Rq) | 69.3 | $\mu$in |

Table A.3: Physical Parameters for the Euclid Unwind.

| Parameter | Value | $Units$ |
|---|---|---|
| Motor Inertia | 4.786E-02 | $slug - ft^2$ |
| Gear Ratio | 2.635E-01 | shaft rotations per motor rotation |
| Shaft inertia | 3.680E-01 | $slug - ft^2$ |
| Roller inertia | 4.635E-03 | $slug - ft^2$ |
| Inertia of wound web (14in dia.) | 5.300E-02 | $slug - ft^2$ |
| Bearing Friction | 6.073E-04 | $lbf - ft - s$ |
| Motor Damping | 0.000E+00 | $lbf - ft - s$ |
| Motor Constant | 1.274E+01 | $lbf - ft/A$ |
| Roller radius | 1.250E-01 | $ft$ |
| Young's modulus (Tyvek) | 6.667E+06 | $lbf/ft^2$ |
| Web cross-sectional area | 2.262E-04 | $ft^2$ |
| Span Length | 5.079E+00 | $ft$ |
| Steady-state speed | 6.667E+00 | $ft/s$ |
| Dancer Arm Length | 1.327E+00 | $ft$ |
| Dancer Inertia | 2.090E-01 | $slug - ft^2$ |
| Dancer applied torque | 1.845E+01 | $lbf - ft$ |

Table A.4: Physical Parameters for the Euclid S-Wrap. The S-Wrap is made up of two identical motors and rollers.

| Parameter | Value | $Units$ |
|---|---|---|
| Motor Inertia | 4.072E-02 | $slug - ft^2$ |
| Gear Ratio | 7.168E-02 | shaft rotations per motor rotation |
| Shaft inertia | 1.315E+00 | $slug - ft^2$ |
| Bearing Friction | 6.073E-04 | $lbf - ft - s$ |
| Motor Damping | 0.000E+00 | $lbf - ft - s$ |
| Motor Constant | 6.249E+01 | $lbf - ft/A$ |
| Roller radius | 5.000E-01 | $ft$ |

Table A.5: Physical Parameters for the Euclid Pull Roll

| Parameter | Value | $Units$ |
|---|---|---|
| Motor Inertia | 4.072E-02 | $slug - ft^2$ |
| Gear Ratio | 7.766E-02 | shaft rotations per motor rotation |
| Shaft inertia | 4.686E-01 | $slug - ft^2$ |
| Bearing Friction | 6.073E-04 | $lbf - ft - s$ |
| Motor Damping | 0.000E+00 | $lbf - ft - s$ |
| Motor Constant | 3.090E+01 | $lbf - ft/A$ |
| Roller radius | 2.500E-01 | $ft$ |

Table A.6: Physical Parameters for the Euclid Rewind

| Parameter | Value | $Units$ |
|---|---|---|
| Motor Inertia | 4.786E-02 | $slug - ft^2$ |
| Gear Ratio | 2.635E-01 | shaft rotations per motor rotation |
| Shaft inertia | 3.680E-01 | $slug - ft^2$ |
| Inertia of wound web (14in dia.) | 3.900E-02 | $slug - ft^2$ |
| Bearing Friction | 6.073E-04 | $lbf - ft - s$ |
| Motor Damping | 0.000E+00 | $lbf - ft - s$ |
| Motor Constant | 1.274E+01 | $lbf - ft/A$ |
| Roller radius | 1.250E-01 | $ft$ |
| Young's modulus (Tyvek) | 6.667E+06 | $lbf/ft^2$ |
| Web cross-sectional area | 2.262E-04 | $ft^2$ |
| Span Length | 1.141E+01 | $ft$ |
| Steady-state speed | 6.667E+00 | $ft/s$ |

Table A.7: Physical Parameters for the High-Speed Web Line Unwind

| Parameter | Value | $Units$ |
|---|---|---|
| Motor Inertia | 2.2068E-01 | $slug - ft^2$ |
| Rated Speed | 1.4200E+03 | RPM |
| Rated Current | 4.8000E+01 | $A$ |
| Rated Torque | 1.3648E+02 | $ft - lbf$ |
| Rated Horsepower | 3.6900E+01 | $Hp$ |
| Gear Ratio | 1E+00 | shaft rotations per motor rotation |
| Shaft inertia | 5.5673E-02 | $slug - ft^2$ |
| Inertia of wound web (16.21in dia.) | 1.4357E-01 | $slug - ft^2$ |
| Bearing Friction | 2.0000E-04 | $lbf - ft - s$ |
| Motor Damping | 2.000E-02 | $lbf - ft - s$ |
| Roller radius | 1.6667E-01 | $ft$ |
| Young's modulus (Tyvek) | 9.9360E+06 | $lbf/ft^2$ |
| Web cross-sectional area | 2.0833E-04 | $ft^2$ |
| Span Length | 4.4497E+00 | $ft$ |
| Steady-state speed | 1.3333E+01 | $ft/s$ |

Table A.8: Physical Parameters for the High-Speed Web Line Pull Roll

| Parameter | Value | $Units$ |
|---|---|---|
| Motor Inertia | 4.0716E-02 | $slug - ft^2$ |
| Rated Speed | 1.7700E+03 | RPM |
| Rated Current | 2.0100E+01 | $A$ |
| Rated Torque | 4.4509E+01 | $ft - lbf$ |
| Rated Horsepower | 1.5000E+01 | $Hp$ |
| Gear Ratio | 1E+00 | shaft rotations per motor rotation |
| Shaft inertia | 3.51756E-01 | $slug - ft^2$ |
| Bearing Friction | 2.0000E-04 | $lbf - ft - s$ |
| Motor Damping | 0.000E+00 | $lbf - ft - s$ |
| Roller radius | 3.3333E-01 | $ft$ |
| Steady-state speed | 1.3333E+01 | $ft/s$ |

Table A.9: Physical Parameters for the High-Speed Web Line Rewind

| Parameter | Value | $Units$ |
|---|---|---|
| Motor Inertia | 2.2068E-01 | $slug - ft^2$ |
| Rated Speed | 1.1500E+03 | RPM |
| Rated Current | 4.8000E+01 | $A$ |
| Rated Torque | 1.3701E+02 | $ft - lbf$ |
| Rated Horsepower | 3.0000E+01 | $Hp$ |
| Gear Ratio | 1E+00 | shaft rotations per motor rotation |
| Shaft inertia | 5.34738E-02 | $slug - ft^2$ |
| Inertia of wound web (14.93in dia.) | 1.1062E-01 | $slug - ft^2$ |
| Bearing Friction | 2.0000E-04 | $lbf - ft - s$ |
| Motor Damping | 0.000E+00 | $lbf - ft - s$ |
| Roller radius | 1.6667E-01 | $ft$ |
| Young's modulus (Tyvek) | 9.9360E+06 | $lbf/ft^2$ |
| Web cross-sectional area | 2.0833E-04 | $ft^2$ |
| Span Length | 4.7660E+00 | $ft$ |
| Steady-state speed | 1.3333E+01 | $ft/s$ |

Table A.10: Web Properties for the High-Speed Web Line

| Parameter | $Units$ | Narrow Tyvek | Wide Tyvek | PET |
|---|---|---|---|---|
| Width | $ft$ | 5.000E-01 | 2.0417E+00 | 5.000E-01 |
| Thickness | $ft$ | 4.1667E-04 | 4.1667E-04 | 1.1667E-04 |
| Young's Modulus | $psi$ | 6.9000E+04 | 6.9000E+04 | 6.2500E+05 |
| Density | $slug/ft^3$ | 8.6610E-01 | 8.6610E-01 | 2.6970E+00 |

Table A.11: Web Properties for 6 inch Wide Tyvek from DuPont. Properties seem to match Tyvek style 1025DF best.

| Parameter | $Units$ | Average Value | Notes |
|---|---|---|---|
| Width | $ft$ | 5.000E-01 | |
| Thickness | $mils$ | 5.0000E+00 | for 1025DF |
| Basis Weight | $oz/yd^2$ | 1.2500E+00 | |
| Young's Modulus | $psi$ | 8.0000E+04 | for 1025DF |
| Tensile SM | $lbf/in$ | 2.2680E+01 | |
| Parker Rough | $\mu m$ | 7.8730E+01 | |
| Parker Smooth | $\mu m$ | 7.1280E+01 | |
| Thickness | $mils$ | 7.8200E+00 | measured from sample |
| Gurley | $sec$ | 2.4880E+01 | a measure of permeability |
| Strain at break | $\%$ | 8.8300E+00 | |
| Static COF | | 2.3960E-01 | St. Dev. 2.7317E-02 |
| Dynamic COF | | 1.2365E-01 | St. Dev. 1.7138E-02 |

# APPENDIX B

# EXPERIMENTS AND SIMULATIONS ON THE EUCLID WEB LINE

## B.1 Simulations of the Rewind Controller on the Euclid Web Line

In this research, the rewind control system of the Euclid Line is investigated in order to show a comparison with the investigation of the hypothetical system in [8] and shown in Figure 1.12. The systems differ substantially. The hypothetical system in [8] uses a PID controller. Only tension feedback is used and there is only one span leading into the rewind roll. The rewind zone of the EWL uses a speed-based tension control system that has an inner speed loop and an outer tension loop. A simulation with three build-up ratios was conducted for the model described in (1.20) using the parameters from the EWL (see Table A.6).

In the fixed gains case, the speed gains were fixed at values calculated for a roll diameter of 3 inches. But, the inertias used in the simulations were for the 12 inch and 18 inch diameter rolls. In the variable gains case, the speed gains were recalculated with each new inertia. The simulation was conducted for a step input in the tension reference for the rewind. The tension gains were fixed at $K_{pt} = 10$ and $K_{it} = 30$ (found by trial and error to work well by a previous researcher that used the Euclid Line for experimental studies [26]). The speed gains were calculated using (1.15) and (1.16).

Figure B.1 shows the step responses in tension for the Euclid rewind at the three build-up ratios. For the plot on the left, the speed gains were not allowed to vary with increased inertia.

Between build-up ratios of 1 and 4, very little difference is seen in the responses, but afterward the overshoot increases with increases in build-up ratio. For the plot on the right, the speed gains were allowed to vary with the roll inertias. In this case, the responses overlay one another for all three build-up ratios. As was the case with the system considered in [8], scheduling the controller gains to account for the changing diameter leads to improved performance.

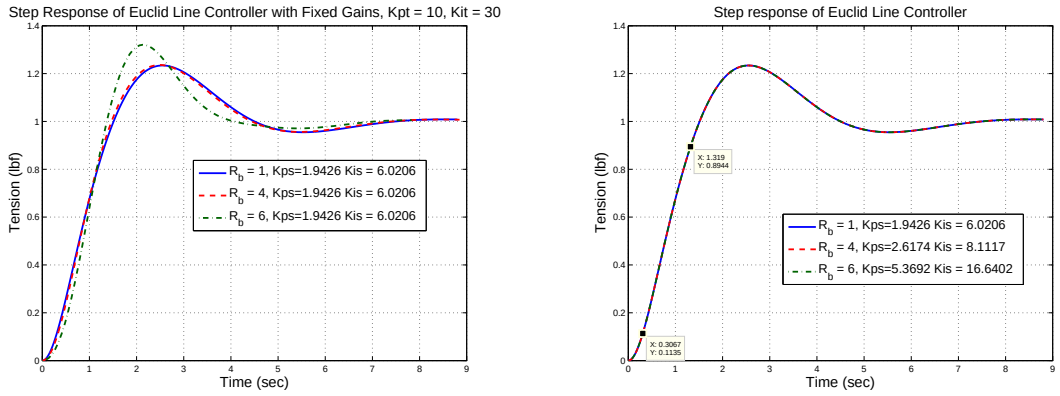



Figure B.1: Euclid Web Line Rewind Tension Responses to a Step Input in Reference Tension. Using the build-up ratio idea, the blue line is a bare shaft, the red dashed line is a 12 inch diameter, and the green dash-dotted line is an 18 inch diameter. The left plot shows the response of the rewind with fixed gains as the roll radius increases. The overshoot increases as the diameter increases. The right plot shows results with variable speed gains.

Two methods of gain scheduling both show improved performance in the web line. Therefore, accounting for the the parent roll diameter change, which is intrinsic to the web line, is important to the simulation of a web line. The two models were subtly different as well. The one from [8] was a differential equation model in the time domain. The second model was entirely executed in the Laplace Domain (frequency domain).

## B.2 The Routh Approximation Method Applied to the Euclid Web Line

The Euclid Web Line at Oklahoma State University's Web Handling Research Center was used to evaluate the systematic method. The physical parameters collected from the line are tabulated in Table A.3 through A.6 in the Appendix. Figure 1.5 on page 9 shows the Euclid Web Line control sections. The unwind section is the unwind roll on the left up to the S-wrap Lead (roll 10), the

S-wrap section is between rolls 10 and 11, the process section is from the S-wrap Follow (roll 11) to the pull roll (roll 16), and the rewind section is from the pull roll to the rewind roll on the far right. The unwind and rewind sections have an outer-loop feedback while the S-wrap and process sections are under pure speed control.

Following the process laid out in the previous section, a model for each control section needs to be created. Equation (3.4) is the speed control model for each of the five motors. Multiplying both sides of (3.4) by $R_n \mathsf{g}_{r_n}$ will convert the equation to tangential speed of the roll instead of angular speed of the motor. A rise time of $0.3$ seconds and a damping ratio of $0.9$ were selected for the experiment. The same rise time and damping ratio are used for each controller in the web line. Using (3.2), the natural frequency of the control loop is $20.6$ rad/s. The inertia, $J_1$, is calculated by reflecting the motor inertia, $J_{m1}$, through the gear ratio, $\mathsf{g}_{r1}$, to the shaft where the web is and adding it to the shaft and wound web inertia, $J_{s1}$. Equations (3.16) and (3.17) are used after selecting a rise time and damping ratio to calculate the proportional and integral gains for the speed loops which are shown in Table B.1.

The unwind section has three different feedback devices: a dancer, a load cell at roller 3, or a load cell at roller 9. The rewind section has a load cell at roller 18. The process for creating the model of the unwind section with dancer position feedback follows. Convert the motor model, (3.4), and the speed control into a transfer function using (3.20) with the unwind parameters which is assigned to $G_s(s)$. After simplifying $G_s(s)$, form $G_2(s)$ by placing $G_s(s)$ in series with two more transfer functions which are (3.9) (the incoming roller speed to span tension model) and (3.11) (the incoming span tension to dancer displacement model) (see Figure 3.7). The lumped span model is assumed so the span length, $L_n$, in (3.9) is $L_n = L_1 + L_2 + L_3$ or $5.079$ft. Transfer functions in series multiply. (3.17) is multiplied by (3.9) and (3.11). Once $G_2(s)$ simplified, the coefficients are processed by the Routh Approximation Method and the proportional and integral gains are calculated by solving Equations (3.24), (3.27), and (3.26). Repeat this process with the rewind section where a load cell is used. Form $G_2(s)$ with (3.9) in series with (3.17) for the rewind. Table B.2 shows the proportional and integral gains for the unwind and rewind sections determined following this method using a damping ratio of $0.9$ and a rise time of $0.3$ seconds.

Table B.1: Euclid Web Line Motor speed gains found using the process with a damping ratio of 0.9 and a rise time of 0.3 seconds.

| Motor | Type of control | $K_p$ | $K_i$ |
|---|---|---|---|
| Unwind | Speed | 2.7928 | 16.1576 |
| S-Wrap Lead | Speed | 5.5421 | 32.0624 |
| S-Wrap Follow | Speed | 5.5421 | 32.0624 |
| Pull Roll | Speed | 5.9853 | 34.6265 |
| Rewind | Speed | 3.91 | 22.697 |

Table B.2: Euclid Web Line position and tension control gains for the unwind and rewind sections.

| Section | Type | $K_p$ | $K_i$ |
|---|---|---|---|
| Unwind | Position | 1.0694 | 0.6054 |
| Rewind | Tension | 0.3475 | 0.0003101 |

## B.3 Experimental Study on the Euclid Web Line

The Euclid Web Line was exercised after applying the gains calculated in the previous section. The gains calculated using the method in the previous section are expecting feedback errors in base units. This implies that the speed loop gains expect speed errors in ft/s. The tension loop gains expect errors in pounds. The dancer position gains are calculated for errors in radians. If those units are not the units used in the web line control and feedback system, the gains must be converted into the correct units. The Euclid Web Line used RPM for the speed loops, percent of maximum load in the tension loop, and degrees in the dancer position loop.

The Euclid Web line was exercised through a 400 FPM start up procedure following an industrial S-curve reference twice using the $0.3$ second rise time and $0.9$ damping ratio. The load cell at roller #3 was used for feedback in the unwind section. The rewind section only had load cell feedback. Figure B.2 shows the speed tracking performance on the left and the tension tracking performance on the right. The speed plots on the left show the difference from the reference S-curve speed for easier comparison. The $0$-$400$ FPM S-curve ends at about $14$ seconds. The solid line S-Wrap and dash-dotted Unwind line go with the as-calculated tension loop gains. The pro-

portional gain in the tension loop for the unwind was increased by $10$ fold to obtain better performance. The dashed S-Wrap line and the dotted Unwind line go with the $10\times$ $K_{pt}$ tension loop gain. The rewind section proportional gain was also increased by $10$ fold to obtain better reference tracking, but little change in performance was noted.

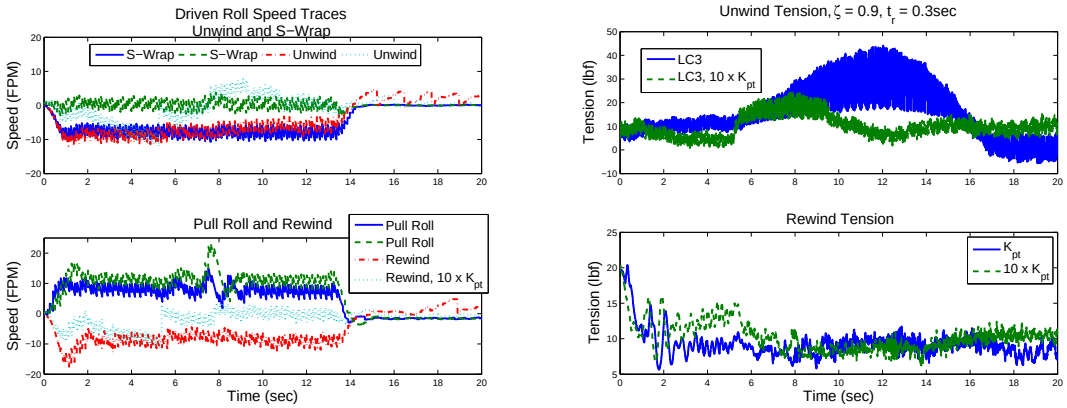



Figure B.2: The left plot shows speed tracking performance of the unwind and s-wrap (above), and the pull roll and rewind motors (below) during a 400 FPM start-up following an industrial S-curve (the S-curve reference has been subtracted out) with $\zeta = 0.9$ and $t_r = 0.3$ seconds. Load cell #3 was the feedback device for the unwind section. The proportional gain for the unwind tension was increase $10$ fold to obtain better performance. The $K_{pt}$ gains for the rewind section were also increased by $10$ fold but the performance is similar during most of the operation. Only during the first few seconds are the two records different.

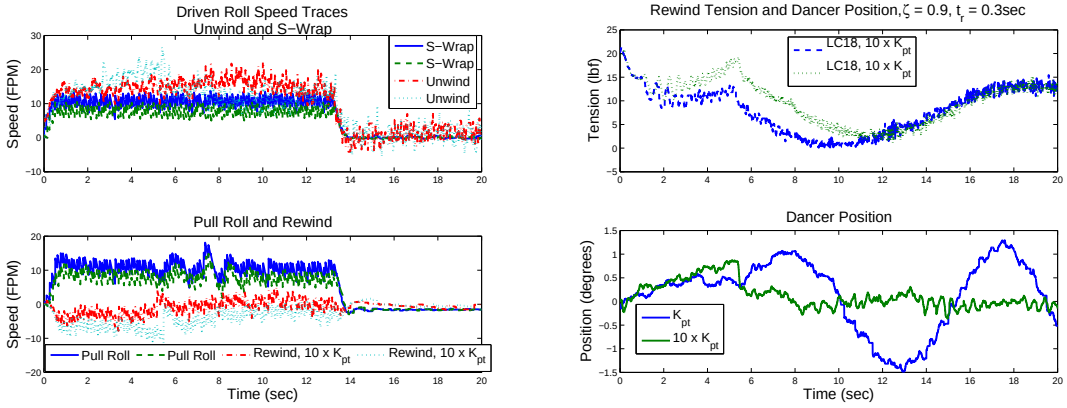



Figure B.3: The Euclid line with dancer feedback control of the unwind section goes through a $400$ FPM start up procedure following an industrial S-curve reference (the S-curve reference has been subtracted out). The rewind gains were unchanged between runs and similar performance was recorded for both runs. The unwind proportional gain for the dancer position was increased $10$ fold to obtain better position tracking.

The unwind section of the Euclid Web Line was switched to dancer position feedback and the $400$ FPM start up procedure using an industrial S-curve was accomplished. Figure B.3 shows the speed tracking performance for the unwind and rewind sections on the left with the S-curve speed reference subtracted out. The solid S-Wrap line and the dash-dotted Unwind line are from the case

where the Unwind tension has not been multiplied by 10. The figure shows the rewind tension and dancer position on the right. The dancer position had a larger magnitude of oscillation before multiplying the proportional gain by 10. The dashed S-Wrap line and the dotted Unwind line are for the case where the proportional gain has been multiplied by 10. There was no change in the rewind tension performance because the proportional gain was increased 10 fold in both cases.

## B.4 Summary

Gains were calculated for the EWL following the method in this chapter assuming DC motor characteristics. Seven sets of PI gains were determined. Experimental studies on the EWL with the calculated gains were found to be good for the speed loops, but were not successful for the tension loops. Successful performance could be obtained if the proportional gains in the tension loops were increased. Even though the gains determined may not be unique and fully optimal, they provide a good starting point for tuning a line after it is built or after a major change is made in the line.

# APPENDIX C

# EXPERIMENTALLY EVALUATING THE EFFECTIVENESS OF A DANCER

The EWL in the WHRC at Oklahoma State University is a roll-to-roll line. It is pictured in Figures 1.6 and 1.5 on page 9 with the spans and rollers numbered in Figure 1.5. The unwind roll is controlled either using feedback tension from a load cell (at roller 9, t1) or dancer position (roller 4, Figure 1.5), and the unwind section is 10 rollers and 9 spans. Tension will be recorded from the load cell and the data will be analyzed for the components of the disturbance. The disturbances are a bump in the unwind roll, an eccentric roller upstream from the dancer, and an eccentric roller downstream of the dancer. Then the dancer is locked out and only the feedback from the load cell is used for a comparison.

## C.1 Identifying Disturbances from FFT of Tension

Using the load cell t1(roller 9), the tension of the web can be recorded in real time. The measured tension shows if the web line is tracking a ramp start-up or if the tension is within tolerances in the vicinity of the load cell. The Fast Fourier Transform (see Appendix I) can be used for converting raw data into the frequency domain and then plotted with MATLAB. The plot can be quite helpful because the rotation rates of many components of the real web line are known or can be derived (see Table C.1). Then the data can be plotted against frequency and the large magnitudes (peaks)

can then be investigated. Finding a component of the line that operates at the frequency of the large magnitude will shine a light on possible culprit for that disturbance. In Figure C.1 the largest peak is just a little past 2 Hz for the blue, 400 FPM data. Looking at Table C.1, the S-wrap has a 1-per-rev frequency of 2.1 Hz at 400 FPM line speed. Thus, the probable cause of the peak in the data near 2 Hz is that one of the S-wrap rollers has a small eccentricity or miss-alignment or surface irregularity.

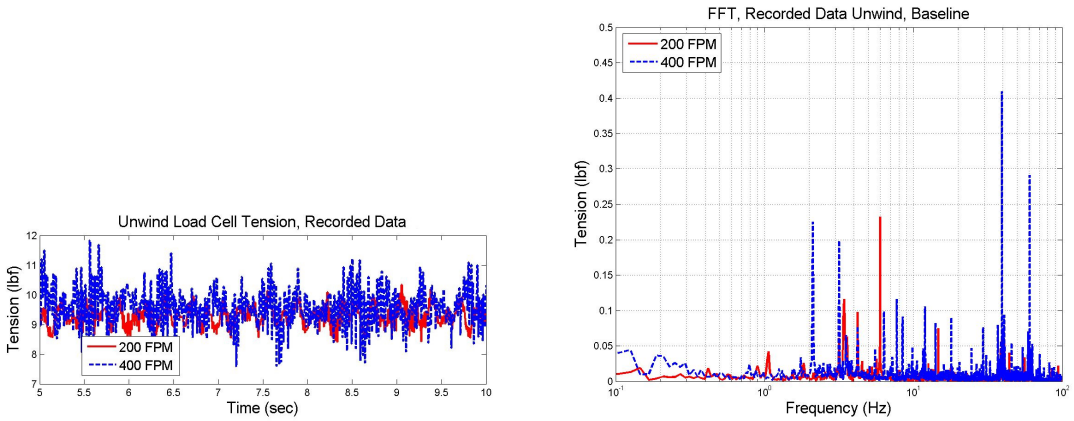



Figure C.1: Baseline Tension, Recorded Tension (left) and FFT (right).

In Figure C.1, the baseline tension, where no added disturbances are present, is shown to indicate that there are non-ideal effects in the web line under the best of circumstances and that the attempt to model the actual system does not include all those effects (and it may be difficult to do so). The load cell at roller 9 is producing a noisy signal, but using the fast-Fourier Transform (FFT) method and Table C.1, the 'noise' is due to several known frequencies: The S-wrap (roller 10) 1-per-rev frequency has a peak at both process speeds, which indicates an eccentricity in the

Table C.1: 1-per-Revolution Frequencies for Given Elements

| Driver | 200FPM | 400FPM |
|---|---|---|
| Unwind | 0.91 Hz | 1.82 Hz |
| Idlers | 4.24 Hz | 8.48 Hz |
| S-wrap | 1.067 Hz | 2.1 Hz |
| Pull Roll | 2.12 Hz | 4.24 Hz |

roller. The idler 1-per-rev frequency is also evident at both process speeds, and for the 400 FPM speed, the pull roll (roller 16) 1-per-rev frequency shows up. This may be because the Pull roll

is the master speed control for the Euclid line in this configuration. The peak at 39 Hz may be a structural resonance or some kind of filtering in the load cell transducer, but that is to be determined. The peak at 60 Hz indicates the load cell is not isolated well from the 60-cycle frequency of electricity. The 39 Hz and 60 Hz peaks will appear in all the measured tension plots. The 3 Hz peak in the 400 FPM data is a control system natural frequency. The 6 Hz peak in the 200 FPM data is a span natural frequency. The model predicts oscillation (very small in magnitude) for the baseline conditions which makes sense because it is a linear model with no disturbances.

Time domain and FFT results from experimentation and simulation are presented. The baseline is the situation describing the default web line. For the EWL, the line speeds are 200 FPM and 400 FPM, with a 14 inch unwind diameter. These two situations are the baselines (see Figure C.1). The plots following them are of disturbed web lines where either a bump is introduced to the unwind roll or an eccentricity is introduced on an idler. The following plots have been scaled to the baseline by multiplying the magnitude of the response of the S-wrap 1-per-rev frequency in the baseline data and dividing the magnitude of the S-wrap response in the disturbed data since the S-wrap peak is not affected by the addition of disturbances but is probably caused by an eccentricity. The peak caused by the S-wrap eccentricity should be the same magnitude in all the plots relative to process speed. The scaling is accomplished to ensure that comparisons from one run to the next are not biased.

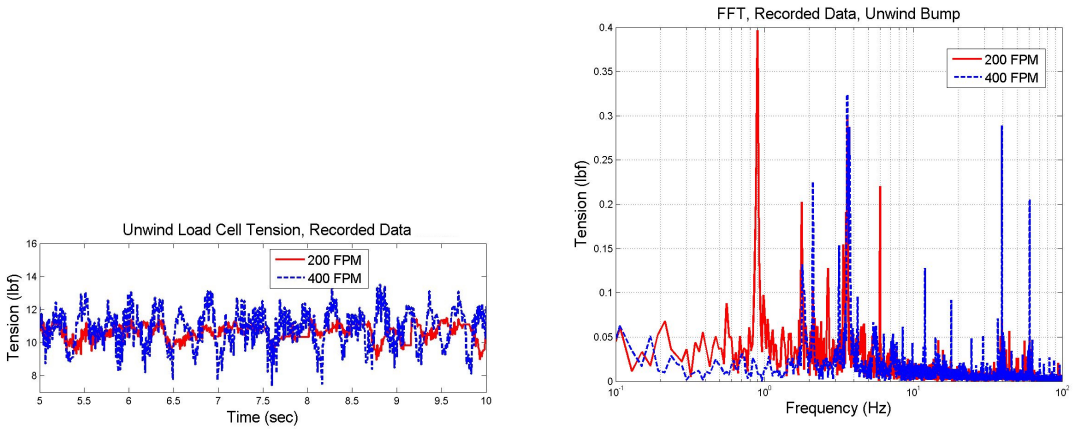



Figure C.2: Unwind Roll Bump Disturbance, Recorded (left) Tension and Frequency Response (right). The Unwind 1-per-rev is a driver, but span natural frequencies show up.

## C.2 Bump on the Unwind Roll

After introducing a $\frac{3}{4}$ inch bar into the unwind roll and laying on web up to the 14 inch diameter, the bump transforms to an $\frac{11}{32}$ inch bump over $54^{\circ}$ of arc. Tension is recorded using the unwind section load cell, t1. The time domain and FFT responses are shown in Figure C.2. The model is also set up with the initial conditions for a bump on the unwind roll. The peak for the 1-per-rev frequency of the unwind bump at 200 FPM in the simulation is about 0.17 lbf while in the recorded data in Figure C.2, it is almost 0.4 lbf. The FFT also shows a large peak at about 1.9 Hz for the 200 FPM data that is not just the first harmonic of the bump frequency, but is also the natural frequency of the control system. The recorded 200 FPM data does not show that at all, but it does have a prominent peak at the first harmonic, 1.8 Hz. Figure C.2 shows peaks in the 200 FPM data at 0.91 Hz (unwind), 1.8 Hz(first harmonic), 2.7 Hz(second harmonic), 3.5 Hz (span natural frequency), and 6 Hz (span natural frequency). For 400 FPM, peaks are located at 1.8 Hz (unwind), 2.1 Hz (pull roll), and 3.5 Hz (span nat. freq.).

## C.3 Idler Eccentricity Upstream of the Dancer

Then an eccentricity is introduced at roller 2, an idler, which gives a higher frequency driver for the dancer to absorb. The eccentricity is 0.1 inch on a 3 inch diameter roller. Thus, for the 200 FPM process speed, the 4.24 Hz frequency should be dominant and for the 400 FPM process speed, the 8.48 Hz frequency should become dominant. The idler position at roller 2 (see Figure 1.5) allows the dancer to affect the disturbance. Figure C.3 illustrates the recorded tension for the upstream eccentric idler and indicates the idler frequency is a driver for both process speeds. Figure C.3 shows peaks for 200 FPM at 4.2Hz (idler), 8.4 Hz (first harmonic), 12.6 Hz, 16.8 Hz, 21 Hz, 25.4 Hz (second-fifth harmonics), and 6 Hz (span nat. freq.). For 400 FPM, the peaks are 2.1 Hz (S-wrap), 8.4 Hz (idler), 16.8 Hz (first harmonic). The linearized model simulation only produced one frequency for each process speed when the FFT was calculated while the recorded data shows evidence of many more frequencies.

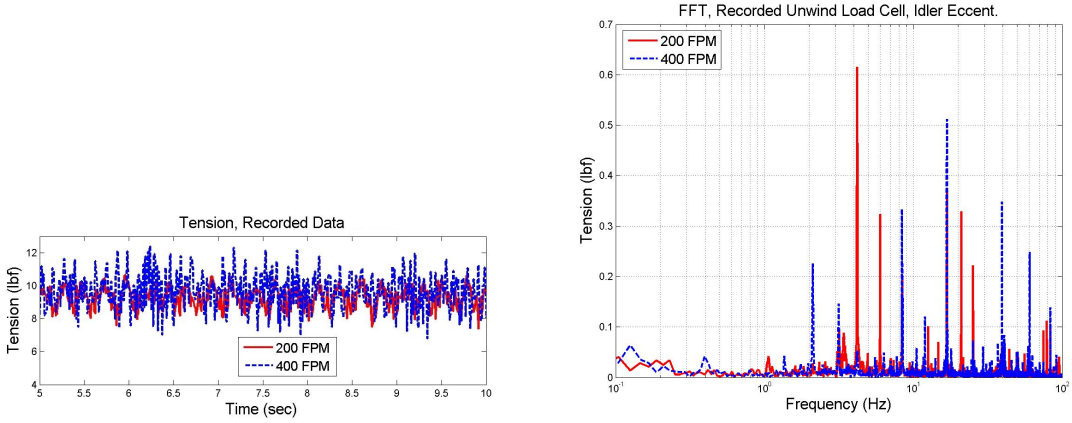



Figure C.3: Idler #2 Eccentricity, Recorded Tension for 200 and 400 FPM process speeds (left) and FFT of Recorded Data (right). The idler frequency is the first large magnitude for both speeds.

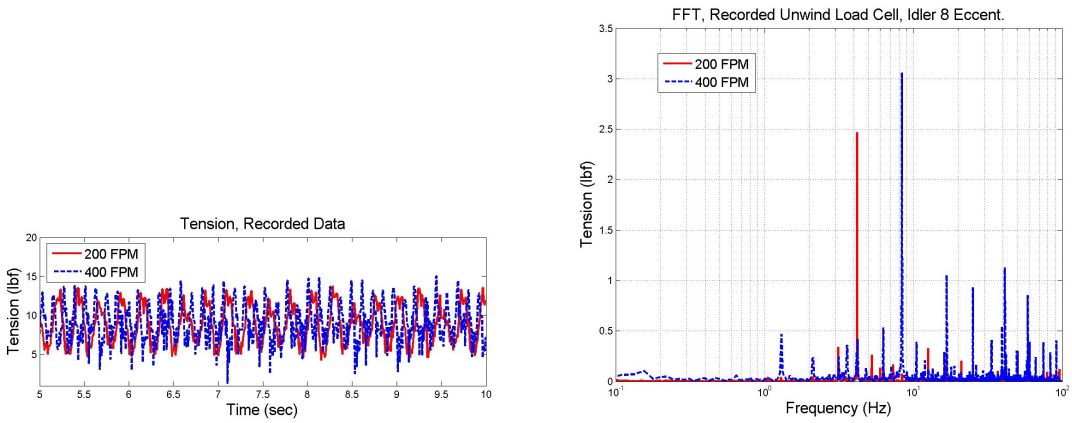



Figure C.4: Eccentric Idler Downstream of the Dancer, Recorded Tension (left) and FFT (right). Recorded data show a 10 lbf peak-to-peak oscillation. The FFT shows the largest peak moving from about 4 Hz to about 8 Hz with the doubling of the line speed.

## C.4 Eccentric Idler Downstream of the Dancer

Previously, the eccentric idler was placed before the dancer so that the dancer could affect the disturbance. Now, the eccentric idler is downstream of the dancer at roller 8 (see Figure 1.5). The disturbance immediately hits the load cell and Figure C.4 shows that the tension variation is large and the FFT shows more noise than previous situations. The 200 FPM peaks are 3.1 Hz (unknown), 4.2 Hz (idler), 5.1 Hz (span nat. freq.), 12 Hz (unknown), 16.8 Hz (fourth harmonic of idler). The 400 FPM peaks from Figure C.4 are 1.2 Hz (unknown), 2.1 Hz (S-wrap), 3.1 Hz (unknown), 3.5Hz (span nat. freq.), 4.2Hz (pull roll), 6.1Hz (span nat. freq.), 8.4 Hz (idler), 11 Hz (span nat. freq.), 16.8 Hz, 25.5 Hz, 33.6 Hz, 42 Hz, 50.4 Hz, 58.8 Hz, 67.2 Hz, 75.6 Hz, 84 Hz, and 92.4 Hz (first-tenth harmonics of idler). Figure C.5 compares the two idler bump scenarios relative to process speed. Comparing a disturbance upstream of the dancer to downstream of the dancer, there is an 83% reduction in the magnitude of the idler 1-per-rev frequency for 200 FPM and 80% reduction for the 400 FPM.

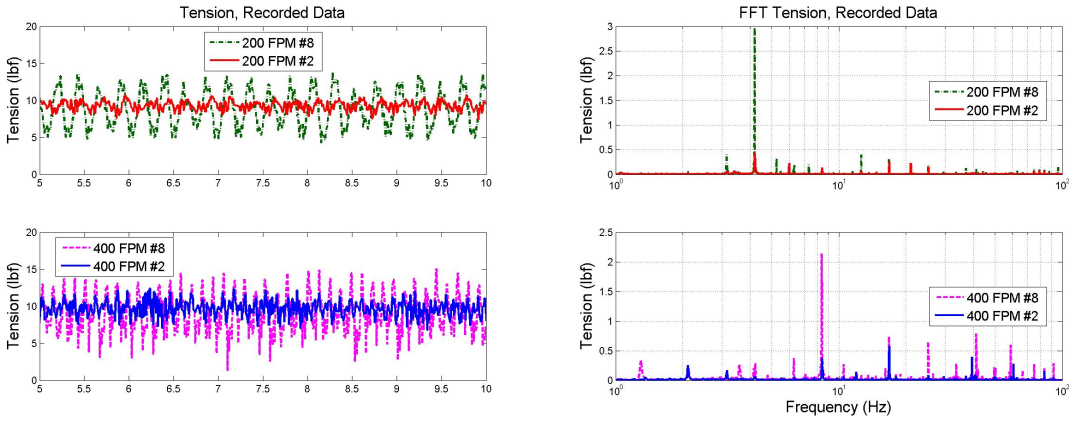



Figure C.5: Eccentric Idler Upstream (#2) of the Dancer vs. Downstream (#8), 200 FPM (left, top), 400 FPM (left, bottom), FFT of 200 FPM (right, top), FFT 400 FPM (right, bottom). The dancer has a large impact on the load cell sensed tension. It removes 83% of the magnitude at the 1-per-rev frequency at 200 FPM and 80% at 400 FPM.

## C.5 Load Cell Feedback

Removing the dancer from the line by locking it out affective makes it just another idler. This leaves the Euclid line to be controlled with load cell feedback. The load cell used for measuring in

the cases with a dancer is selected for control (roller 9). Using the same bump in the unwind as before, data was recorded with load cell control. The recorded data takes on a different appearance compared to runs with a dancer (see Figure C.6). From the FFT, the disturbance is from the unwind

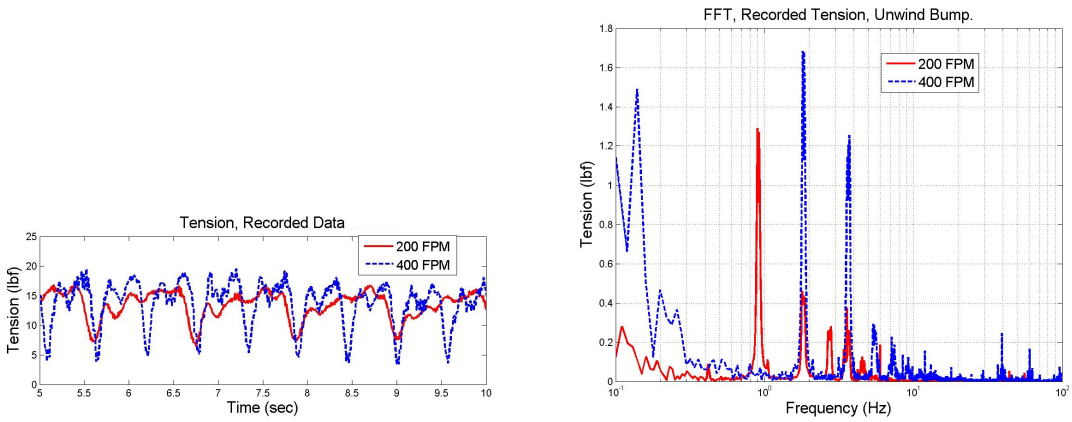



Figure C.6: Unwind Bump Disturbance with No Dancer, Recorded Tension (left) and FFT (right). The unwind 1-per-rev frequency (0.91Hz for 200 FPM and 1.8 Hz for 400 FPM) is clearly present as are several harmonics. The time domain data shows a slow oscillation at 400 FPM that shows up as low frequency peaks in the FFT.

which is as expected and is the case for both operating speeds. Except for the low frequency oscillations, the other peaks in the FFT are the harmonics of the unwind bump 1-per-rev frequency. The harmonics can be seen for each speed. The load cell control is allowing tension oscillations in the 10 lbf range. Compare that with the same unwind bump using a dancer (Figure C.2), where the tension oscillations were in the 6-7 lbf range. Figure C.7 looks at the unwind bump disturbance at 200 FPM and 400 FPM with and without the dancer. The time domain is zoomed in on a 5 second section to show differences in the response with and without the dancer. The FFT shows the same information, but it is easier to appreciate that the dancer removes 83% of the 1-per-rev frequency magnitude at 200 FPM and 82% at 400 FPM looking at the FFT. Figure C.8 compares the simulation results for the unwind bump disturbance and the simulation also show an 84% reduction in the magnitude of the 1-per-rev frequency for 200 FPM and 61% reduction for the 400 FPM case. At both speeds, the natural frequency of the dancer (0.42Hz) is evident (not so in the recorded data of Figure C.2). The simulated no-dancer situation has a control system natural frequency at 0.48Hz and the other tall peaks for that situation are harmonics of the controller natural frequency. The 200 FPM with-dancer simulation has a control system natural frequency at 1.72Hz which hits a harmonic of the dancer natural frequency and a resonant frequency of the idler subsystem. The

400 FPM with-dancer plot shows several frequencies around the dancer natural frequency and then a fourth harmonic at 1.68Hz which is much larger than the unwind 1-per-rev at 1.82Hz. The 400 FPM without-dancer plot shows peaks at 0.47Hz (controller nat. freq.), 1.3Hz (unknown), and 2.6Hz (first harmonic).

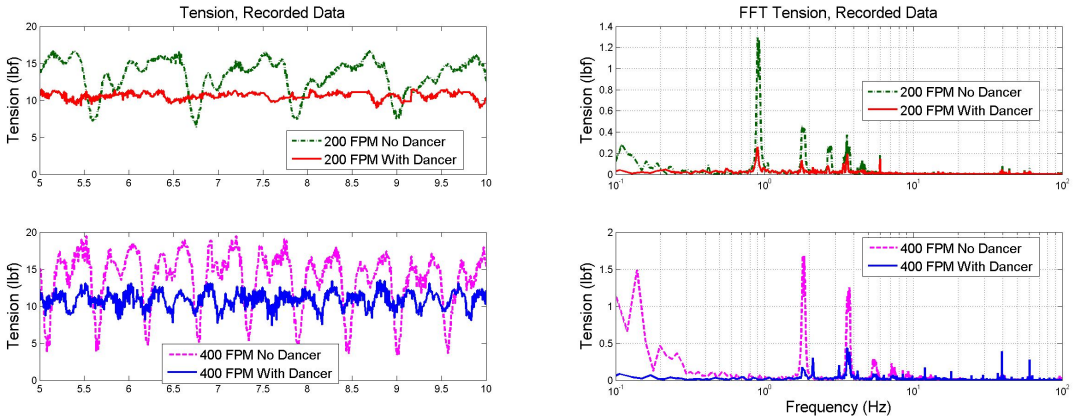



Figure C.7: Comparing the Unwind Bump Disturbance without and with a Dancer. Without a dancer, the oscillations are much larger and sharper (note the zoomed in time scale). The dancer removes 83% of the 1-per-rev frequency magnitude at 200 FPM and 82% at 400 FPM.

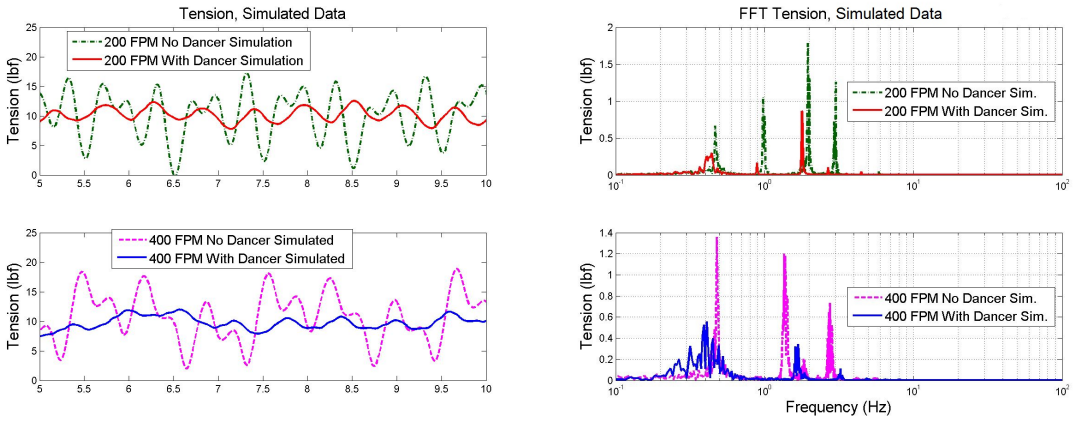



Figure C.8: Comparing Simulations of the Unwind Bump Disturbance without and with a Dancer. The 200 FPM simulation shows an 84% reduction in magnitude of the Unwind 1-per-rev frequency. The 400 FPM simulation shows a 61% reduction at the unwind 1-per-rev frequency.

## C.6 Conclusion

The Fast Fourier Transform (FFT) is a tool that allowed sources of disturbance to be extracted from the recorded load cell tension data. This tool was used to compare several different circumstances

occurring on the EWL. Then, a simulation of the same web line was created from 'primitive elements'. The simulation results were shown at the end to compare with experimental results.

Qualitatively, the simulations do not have the same shape, but that is to be expected when using linearized models. They do agree with the recorded data that the dancer has a magnitude reducing effect on disturbances that occur upstream of the dancer. Quantitatively, the linear model does not mimic the real system. It is often smoother in response and of smaller magnitude than the recording. The limiting case may be the linearized models chosen for the simulations. Nonlinear models would allow for more variability and thereby give a more accurate result.

The experimental results showed that the dancer was highly effective in removing more than 80% of the magnitude of the oscillation at the 1-per-rev frequency of the unwind bump disturbance for both process speeds compared to not having one at all. The dancer also reduced the upstream eccentric idler 1-per-rev frequency oscillations by 80% when compared to the eccentric idler disturbance downstream of the dancer. The dancer has an 80% rejection ratio for reducing disturbances that enter the line upstream of it in measured data and a 60% rejection ratio for the same in simulation. [76]

# APPENDIX D

# THE WEB SPAN DIFFERENTIAL EQUATION

## D.1 Tension in a Web Span

This chapter is describes the derivation of the differential equation for tension in a web span. The process begins by defining a control volume for the web as shown in Figure D.1. Once the control

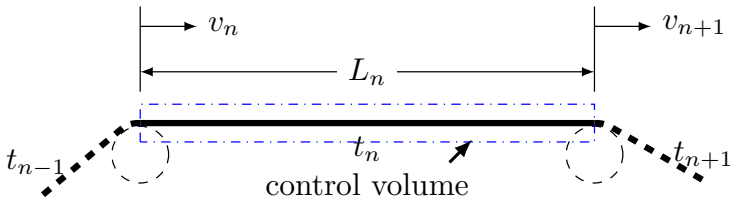



Figure D.1: Control volume defined for the span between rollers

volume is defined, the mass of the web in the span is calculated.

$$\int_{x_1(t)}^{x_2(t)} \rho(x,t)A(x,t)\mathrm{d}x \tag{D.1}$$

The rate of change of the mass in the control volume is what is important.

$$\frac{\mathrm{d}}{\mathrm{d}t}\left[\int_{x_1(t)}^{x_2(t)} \rho(x,t)A(x,t)\mathrm{d}x\right] \tag{D.2}$$

At this point, a differential unit of web needs to be described. It is $\mathrm{d}x$ long by $\mathrm{d}y$ wide by $\mathrm{d}z$ thick in its stretched state. The unstretched state is denoted by a subscript $u$.

$$\mathrm{d}m = \rho(x,t)\mathrm{d}x\mathrm{d}y\mathrm{d}z \tag{D.3}$$

$$\mathrm{d}m_u = \rho_u(x,t)\mathrm{d}x_u\mathrm{d}y_u\mathrm{d}z_u \tag{D.4}$$

The differential mass element of web is shown in Figure D.2 and $\hat{b}_n$ and $\hat{b}_{n+1}$ are the directions the

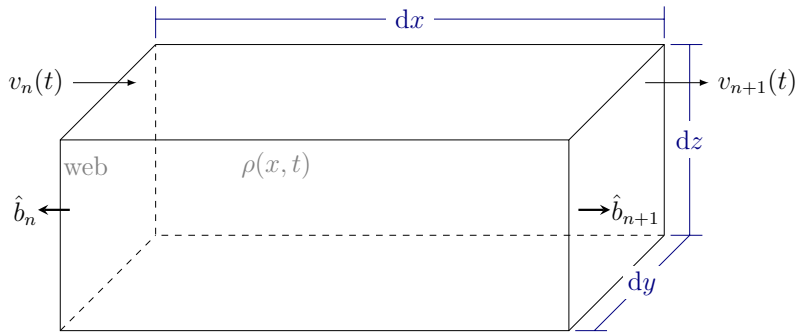



Figure D.2: The differential mass element of web. The boundary movements are shown with $\hat{b}$'s and the velocities are shown for reference.

boundaries of the control volume are moving. The web velocity at the beginning is $v_n(t)$ and at the end is $v_{n+1}(t)$ and are shown for reference only. The velocities would not apply directly to the differential unit. The stretched length $\mathrm{d}x$ is equal to the unstretched length times the $1+\epsilon$, the strain. Assuming very small changes in width of the web and thickness of the web, the differential mass element of the web can be described in terms of the width, $w$, and thickness, $t_{web}$, which are both assumed constants.

$$\mathrm{d}m = \rho(x,t)\,\mathrm{d}x\,w\,t_{web} \tag{D.5}$$

$$\mathrm{d}x = \mathrm{d}x_u(1+\epsilon(x,t)) \tag{D.6}$$

$$\mathrm{d}m = \rho(x,t)\,\mathrm{d}x_u(1+\epsilon(x,t))\,w_u\,h_u \tag{D.7}$$

Assuming that the stretched differential mass unit and the unstretched differential mass unit have the same mass, the two can be equated (think of this differential mass element being out in the middle of the span, not the first or last element of the span). Cross-sectional area is defined as $A = w\,t_{web}$.

$$\mathrm{d}m = \mathrm{d}m_u \tag{D.8}$$

$$\rho(x,t)\,\mathrm{d}x\,w\,t_{web} = \rho_u(x,t)\mathrm{d}x_u\,w\,t_{web} \tag{D.9}$$

$$\rho(x,t)\,\mathrm{d}x_u(1+\epsilon(x,t))\,w\,t_{web} = \rho_u(x,t)\mathrm{d}x_u\,w_u\,t_{webu} \tag{D.10}$$

$$\frac{\rho(x,t)\,(1+\epsilon(x,t))\,w\,t_{web}}{\rho_u(x,t)\,w_u\,t_{webu}} = 1 \tag{D.11}$$

$$\frac{\rho(x,t)\,A}{\rho_u(x,t)\,A_u} = \frac{1}{(1+\epsilon(x,t))} \tag{D.12}$$

Plugging (D.12) into (D.2) gives:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left[\int_{x_1(t)}^{x_2(t)} \frac{\rho_u(x,t)A_u(x,t)}{(1+\epsilon(x,t))}\mathrm{d}x\right] = \frac{\mathrm{d}}{\mathrm{d}t}\left[\int_{x_1(t)}^{x_2(t)} \frac{\rho_u A_u}{(1+\epsilon(x,t))}\mathrm{d}x\right] \tag{D.13}$$

assuming that the density and cross-sectional area are constant which is enforced by earlier assumptions. The density and cross-sectional area can be moved outside the integral and the derivative (from chain rule).

$$\rho_u A_u \frac{\mathrm{d}}{\mathrm{d}t}\left[\int_{x_1(t)}^{x_2(t)} \frac{1}{(1+\epsilon(x,t))}\mathrm{d}x\right] \tag{D.14}$$

Now $(1+\epsilon)$ in the denominator is a very difficult function to integrate, but an approximation can be made by following these steps and assuming that the strain ($\epsilon(x,t)$) is small, $\epsilon(x,t) << 1$. The area and density drop out because they are not important to this derivation.

$$\frac{\mathrm{d}}{\mathrm{d}t}\left[\int_{x_1(t)}^{x_2(t)} \frac{1}{(1+\epsilon(x,t))}\mathrm{d}x\right] = \frac{\mathrm{d}}{\mathrm{d}t}\left[\int_{x_1(t)}^{x_2(t)} \frac{1}{(1+\epsilon(x,t))}\frac{(1-\epsilon(x,t))}{(1-\epsilon(x,t))}\mathrm{d}x\right] \tag{D.15}$$

$$= \frac{\mathrm{d}}{\mathrm{d}t}\left[\int_{x_1(t)}^{x_2(t)} \frac{(1-\epsilon(x,t))}{1-(\epsilon(x,t))^2}\mathrm{d}x\right] \tag{D.16}$$

$$\approx \frac{\mathrm{d}}{\mathrm{d}t}\left[\int_{x_1(t)}^{x_2(t)} (1-\epsilon(x,t))\mathrm{d}x\right] \tag{D.17}$$

Executing the Leibniz's integral rule[1] on (D.17) yields the following:

$$\left[\int_{x_1(t)}^{x_2(t)} \mathrm{d}x\right] \frac{\mathrm{d}}{\mathrm{d}t}\left(1-\epsilon(x,t)\right) + \frac{\mathrm{d}}{\mathrm{d}t}\left[\int_{x_1(t)}^{x_2(t)} \mathrm{d}x\right] \left(1-\epsilon(x,t)\right) \tag{D.18}$$

$$\left[x_2(t)-x_1(t)\right] \frac{\mathrm{d}}{\mathrm{d}t}\left(-\epsilon(x,t)\right) + \frac{\mathrm{d}}{\mathrm{d}t}\left[\int_{x_1(t)}^{x_2(t)} \mathrm{d}x\right] \left(1-\epsilon(x,t)\right) \tag{D.19}$$

$$\left[x_2(t)-x_1(t)\right] \frac{\mathrm{d}}{\mathrm{d}t}\left(-\epsilon(x,t)\right) + \left[\frac{\mathrm{d}}{\mathrm{d}t}x_2(t) - \frac{\mathrm{d}}{\mathrm{d}t}x_1(t)\right] \left(1-\epsilon(x,t)\right) \tag{D.20}$$

From Figures D.1 and D.2, the definitions of $x_1(t)$ and $x_2(t)$ are the beginning and ending points of the control volume in the longitudinal direction and these boundaries are allowed to move ($\hat{b}_n$ and $\hat{b}_{n+1}$). So, at the entering boundary, the derivative of $x_1(t)$ includes both the speed of the web entering the control volume and the speed of the boundary moving, if any. The same is true of the

[1] Leibniz's integral rule is

$$\frac{\mathrm{d}}{\mathrm{d}t}\left[\int_{\phi(t)}^{\psi(t)} g(x,t)\mathrm{d}x\right] = \int_{\phi(t)}^{\psi(t)} \frac{\partial}{\partial t}g(x,t)\mathrm{d}x + \frac{\mathrm{d}\psi(t)}{\mathrm{d}t}g(\psi(t),t) - \frac{\mathrm{d}\phi(t)}{\mathrm{d}t}g(\phi(t),t)$$

exiting boundary at $x_2(t)$.

$$x_1(t) = \int_0^t v_n(\tau)\mathrm{d}\tau - \int_0^t \hat{b}_n(\tau)\mathrm{d}\tau \tag{D.21}$$

$$x_2(t) = \int_0^t v_{n+1}(\tau)\mathrm{d}\tau + \int_0^t \hat{b}_{n+1}(\tau)\mathrm{d}\tau \tag{D.22}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}x_1(t) = v_n(t) - \hat{b}_n(t) \tag{D.23}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}x_2(t) = v_{n+1}(t) + \hat{b}_{n+1}(t) \tag{D.24}$$

Now $\hat{b}_n$ could be equal to zero for all time if the web span is not length varying. In the case of an accumulator, dancer, or non-circular unwind roll, the boundary of the span is moving, $\hat{b}_n(t) \neq 0$. In this example, $\hat{b}_n(t)$ could be either end of the span. Usually the span is only allowed to change in length based on the which roller is physically moving. The other $\hat{b}$ term is zero. The practical side of the moving boundary is that the value of $\hat{b}_n$ is usually governed by the movement of a dancer or an accumulator carriage. The boundary movement is not chaotic. Equation (D.20) then becomes:

$$[x_2(t) - x_1(t)]\,\frac{\mathrm{d}}{\mathrm{d}t}\left(-\epsilon(x,t)\right) + \left[v_{n+1}(t) + \hat{b}_{n+1}(t) - \left(v_n(t) - \hat{b}_n(t)\right)\right](1 - \epsilon(x,t)) \tag{D.25}$$

$$x_2(t) - x_1(t) = L_n \tag{D.26}$$

$$\begin{aligned} L_n\,\frac{\mathrm{d}}{\mathrm{d}t}\left(-\epsilon(x,t)\right) + \left[v_{n+1}(t) + \hat{b}_{n+1}(t) - \left(v_n(t) - \hat{b}_n(t)\right)\right] + \cdots \\ -\epsilon(x,t)\left[v_{n+1}(t) + \hat{b}_{n+1}(t) - \left(v_n(t) - \hat{b}_n(t)\right)\right] \end{aligned} \tag{D.27}$$

$$\begin{aligned} L_n\,\frac{\mathrm{d}}{\mathrm{d}t}\left(-\epsilon_n(t)\right) + \left[v_{n+1}(t) + \hat{b}_{n+1}(t) - \left(v_n(t) - \hat{b}_n(t)\right)\right] + \cdots \\ +v_n(t)\epsilon_n(t) - \hat{b}_n(t)\epsilon_n(t) - v_{n+1}(t)\epsilon_{n+1}(t) - \hat{b}_{n+1}(t)\epsilon_{n+1}(t) \end{aligned} \tag{D.28}$$

Equation (D.28) is just a restatement of an approximation of the rate of change of the length of span (of span mass (D.14)). If the rate of change of span length is assumed to be zero, then (D.28) can be the differential equation of the rate of change of strain in the web span.

$$\begin{aligned} L_n\,\frac{\mathrm{d}}{\mathrm{d}t}\epsilon_n(t) = \left[v_{n+1}(t) + \hat{b}_{n+1}(t) - \left(v_n(t) - \hat{b}_n(t)\right)\right] + \cdots \\ + v_n(t)\epsilon_n(t) - \hat{b}_n(t)\epsilon_n(t) - v_{n+1}(t)\epsilon_{n+1}(t) - \hat{b}_{n+1}(t)\epsilon_{n+1}(t) \end{aligned} \tag{D.29}$$

Assuming that the web is a linear elastic material allows the usage of Hooke's Law: $t = EA\epsilon$. If (D.29) is multiplied by the cross-sectional area of the web and Young's Modulus for the material of

the web, then (D.29) becomes the differential equation for tension in the span.

$$L_n \frac{\mathrm{d}}{\mathrm{d}t} t_n = AE \left[ v_{n+1}(t) + \hat{b}_{n+1}(t) - \left( v_n(t) - \hat{b}_n(t) \right) \right] + \cdots$$
$$+ v_n(t) t_n(t) - \hat{b}_n(t) t_n(t) - v_{n+1}(t) t_{n+1}(t) - \hat{b}_{n+1}(t) t_{n+1}(t) \tag{D.30}$$

If the $\hat{b}$ terms are set to 0 as in a constant length span, the equation reduces to (2.2) which [4,19,24] and more use. Shelton points out that (D.30) is nonlinear even without the $\hat{b}$ terms in [18]. Analysis of the web line at an operating point allows the linearization of the nonlinear terms, if desired.

## D.2 Accounting for Changes in Length of Span

Looking back through the previous section, many of the steps are only the right-hand side of an equals sign that is not shown. The left-hand-side is (D.14) which is the rate of change of mass in the web span. That is why the density and cross-sectional area drop out shortly after (D.14): they are divided across the equals sign. The remaining term is the rate of change of the length of the span. Maintaining an accounting of the quantity of the length of span allows a cross check on the span tension.

Many sources assume the change in the span length to be zero, unless a dancer or accumulator is being studied. That was done in the previous section between (D.28) and (D.29). If, instead, the quantity of length is accounted, then it can be used as a check for when slack occurs in the span. Slack is the situation where there is more length of web between two rollers than the shortest path. The web loses tension and the strain in the span goes to near zero. In fact, the tension in the web is that due to its own weight which is usually small, but nonzero.

$$\frac{\mathrm{d}}{\mathrm{d}t} \left[ \int_{x_1(t)}^{x_2(t)} \frac{1}{(1+\epsilon(x,t))} \mathrm{d}x \right] = - L_n \frac{\mathrm{d}}{\mathrm{d}t} \frac{t_n}{AE} + \left[ v_{n+1}(t) + \hat{b}_{n+1}(t) - \left( v_n(t) - \hat{b}_n(t) \right) \right] + \cdots$$
$$+ v_n(t) \frac{t_n}{AE} - \hat{b}_n(t) \frac{t_n}{AE} - v_{n+1}(t) \frac{t_{n+1}}{AE} - \hat{b}_{n+1}(t) \frac{t_{n+1}}{AE} \tag{D.31}$$

If (D.31) is integrated over time, the result is the span length which based on the assumptions from earlier, should remain constant. However, there are two situations where it will not: when the span is part of a variable length subsystem like a dancer or accumulator, or when slack occurs. Slack occurs when the difference in $v_n$ and $v_{n+1}$ is too great while the other situation is covered

by $\hat{b}_n \neq 0$ or $\hat{b}_{n+1} \neq 0$. There is a possibility of the combination of the two situations in the case of a dancer or accumulator.

Looking at (D.31) in each situation above may shed light on what is happening. The first situation was when the span is part of a variable length subsystem. Then, depending on which end of the span is moving, either $\hat{b}_n \neq 0$ or $\hat{b}_{n+1} \neq 0$. Assuming that roller $n+1$ is moving, then it is $\hat{b}_{n+1} \neq 0$. Let the left-hand-side, inside the derivative, be defined as $L_n(t)$.

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}\left[L_n(t)\right] = &- L_n(t)\frac{\mathrm{d}}{\mathrm{d}t}\left[\frac{t_n}{AE}\right] + \left[v_{n+1}(t) + \hat{b}_{n+1}(t) - v_n(t)\right] + \cdots \\ &+ v_n(t)\frac{t_n}{AE} - v_{n+1}(t)\frac{t_{n+1}}{AE} - \hat{b}_{n+1}(t)\frac{t_{n+1}}{AE}\end{aligned} \tag{D.32}$$

The signs in (D.32) assume that the boundary of the span is moving in the same direction as the web exiting the span control volume (see Figure D.2). Also, the length of the span multiplied by the derivative of tension in span $n$ is now a time varying quantity. Other authors ( [9,11,13,19,35]) have assumed that continuity of mass prevails and the derivative of tension in span $n$ equates to everything else, but $\hat{b}_{n+1}$. It is like assuming that (2.2) is true and therefore can fall out of (D.32) leaving (D.33)[2].

$$\frac{\mathrm{d}}{\mathrm{d}t}\left[L_n(t)\right] = \hat{b}_{n+1}(t) - \hat{b}_{n+1}(t)\frac{t_{n+1}}{AE} \tag{D.33}$$

The second situation, slack, can happen when neither supporting roller moves. In this case, both $\hat{b}_n = 0$ and $\hat{b}_{n+1} = 0$. Equation (D.31) then becomes the following.

$$\frac{\mathrm{d}}{\mathrm{d}t}\left[L_n(t)\right] = - L_n(t)\frac{\mathrm{d}}{\mathrm{d}t}\left[\frac{t_n}{AE}\right] + \left[v_{n+1}(t) - v_n(t)\right] + v_n(t)\frac{t_n}{AE} - v_{n+1}(t)\frac{t_{n+1}}{AE} \tag{D.34}$$

The main driver in this situation is the difference in $v_n$ and $v_{n+1}$. Slack can occur in a variable length span subsystem, too. Where $L_n(t) = L_n(0)$, the shortest path length between the rollers, the derivative of tension changes. It becomes a function of the accumulated length, i.e. mass.

$$\frac{\mathrm{d}}{\mathrm{d}t}t_n = \frac{\mathrm{d}}{\mathrm{d}t}\left[\mathsf{w}_{web}t_{web}\rho_{web}g\int_{L_n(0)}^{L_n(t)}\mathrm{d}\lambda\right] \tag{D.35}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}t_n = \mathsf{w}_{web}t_{web}\rho_{web}g\frac{\mathrm{d}}{\mathrm{d}t}L_n(t) \tag{D.36}$$

[2]Similar to the situation in Fluids where the conservation of mass equation is found inside the conservation of momentum equation and since no mass is generated, the conservation of mass equation equals 0. Therefore all its terms can be summed together, which equals 0, and they are gone from the conservation of momentum equation.

A nonzero limit for the lowest tension in the span can be found from (D.36). Integrating both sides with respect to time and then evaluating the remaining integral from $0$ to $L_n(0)$ will yield the weight of the span which is the minimum tension the span will see. It has to carry its own weight between rollers.

The usefulness of the equations above is that they may be used to check for slack. There is a default length of each span, $L_n$, found at the steady-state of the system. That value is used as the starting length. Each length may have to be adjusted for the strain in the span due to tension. That means that during simulation, the unstretched length of span between two rollers is shorter than the default distance between the rollers. The check for slack would entail comparing the calculated length for each span to the steady-state length found at the beginning of the simulation. Another option would be to test the calculated span tension against the minimum span tension. The speed difference between the incoming roller and the outgoing roller does not change, the span is slack.

If the check showed that the calculated length of span was longer than the default length, that indicates that the tension in the span went to zero and that mass is accumulating between the rollers. The tension of the span could be replaced with only the weight of the web mass between the rollers and its derivative set to basically zero. The effect is that tension in the span does not become negative, which is a continual problem in other implementations of the span tension equation. Tension can not be negative, by definition. Compression of the web would crumple it up. Web can not sustain compression.

Also, this implementation of the span tension equation with length check could be used to simulate a slack condition and test the default measures that are programmed into the control to attend to the situation. Options include braking the unwind roll and accelerating the line speed downstream of the slacked span to pull the excess material out of the span.

# APPENDIX E

# TENSION IN THE REGION OF SLIP ON A ROLLER

A web of material wrapped over a roller was divided into two regions by Brandenburg in [17, 44]. The two regions are the adhesive region and the slip region. The tension is assumed constant in the adhesive region and variable in the slip region. The tension follows a differential equation of the form:

$$\frac{\mathrm{d}t_n(x')}{\mathrm{d}x'} \mp \frac{\mu t_n(x')}{R_n} = 0 \tag{E.1}$$

The $x'$ term is distance around the periphery of the roller and $\mu$ is the coefficient of friction (static and dynamic are assumed equal). The roller's inertial effects can be included.

$$\rho_A t_{web} \frac{\mathrm{d}v_n}{\mathrm{d}t} - \frac{\partial t_n(x',t)}{\partial x'} \pm \frac{\mu t_n(x',t)}{R_n} = 0 \tag{E.2}$$

The value of $\rho_A$ is the area density (not the normal volume density) that is common in industry for webs and $t_{web}$ is the web thickness. For the area density, the area in question is the plan area, not the cross-sectional area due to the inconvenience of being able to obtain a repeatable thickness measurement of webs. Industry has chosen this measure for repeatability because of the compressibility (crushableness) of many webs in use today. Web thickness is usually very small and the area density is small, so the first term can usually be neglected.

$$\frac{\partial t_n(x')}{\partial x'} \mp \frac{\mu t_n(x')}{R_n} = 0 \tag{E.3}$$

Moving the friction term across the equals sign and integrating both sides yields:

$$\frac{\partial t_n(x')}{\partial x'}\frac{1}{t_n(x')} = \pm\frac{\mu}{R_n} \tag{E.4}$$

$$\ln\left(t_n(x')\right) = \pm\frac{\mu}{R_n}x' + C(t) \tag{E.5}$$

$$\ln\left(t_n(0)\right) = \pm\frac{\mu}{R_n}0 + C(t) \tag{E.6}$$

$$t_n(0) = e^{C(t)} = t_{n-1} \tag{E.7}$$

$$\ln\left(t_n(x'_o)\right) - \ln(t_{n-1}) = e^{\pm\mu\frac{x'}{R_n}} \tag{E.8}$$

$$\ln\left(\frac{t_n(x'_o)}{t_{n-1}}\right) = e^{\pm\mu\frac{x'}{R_n}} \tag{E.9}$$

Looking at $x'$ from another perspective, it is the arc length of web-roller contact. So the quantity $x'/R_n$ is the wrap angle, $\theta_{wn}$, when $x'$ goes from 0 to its fullest extent. At the beginning of the wrap angle, $x' = 0$, the tension has to be $t_{n-1}$. Thus, $e^{C(t)} = t_{n-1}$, as shown in (E.7). Moving to the other end of the region of contact, where $x'/R_n = \theta_{wn}$, the tension in the span is that of the free span $n$ or $t_n$.

$$t_n = (t_{n-1})e^{\pm\mu\theta_{wn}} \tag{E.10}$$

The sign of the exponent in (E.10) is determined by the sign of $\left(-\frac{\mathrm{d}t_{n-1}}{\mathrm{d}t}\right)$. Usually, $t_{n-1}$ is assumed to be less than $t_n$.

Whitworth, in [45] cites Brandenburg as having divided up the region of contact between the web and roller into three regions. Neither [44] nor [17] shows this, only that Brandenburg divided the contact area into two regions. Brandenburg's dissertation may be the source Whitworth is referring to, but Brandenburg's dissertation is only available in German. Whitworth uses the three region approach shown in Figure E.1. The angle between 0 and $\xi$, in the direction of roller rotation, is the entry region of slip. The region between $\xi$ and $\beta$ is the active region of adhesion, and the region between angles $\beta$ and $\theta_{wn}$ is the exit region of slip. The tension in the contact region(s) is called $t_n^c(x', t)$ where $x'$ is the circumferential distance from the web's initial contact with the roller to where it exits the roller.

The three regions Whitworth describes are (shown in Figure E.1) delineated by boundaries called $\xi(t)$ and $\beta(t)$.

Region I: The entry slip region between $0 \leq x' \leq \xi R_n$ has tension that satisfies $t_n^c(x', t) = t_{n-1}e^{(\pm\mu x'/R)}$ and the sign of the exponent is determined by $-\frac{\mathrm{d}t_{n-1}}{\mathrm{d}t}$

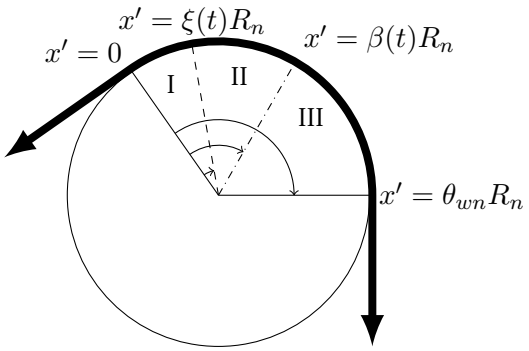



Figure E.1: Roller showing three regions of contact described by Whitworth. Region I is the entry slip region. Region II is the adhesion region. Region III is the exit slip region.

Region II: The active adhesion region between $\xi R_n \leq x' \leq \beta R_n$ where the tension satisfies $\frac{\mathrm{d}t_n^c}{\mathrm{d}t} + V\frac{\partial t_n^c}{\partial x'} = 0$ and general solutions of that equation are of the form $t_n^c(x', t) = f(t - x'/V)$ where $f$ is any differential function

Region III: The exit slip region between $\beta R_n \leq x' \leq \theta_{wn}R_n$ where the tension satisfies $t_n^c(x', t) = t_n e^{(\pm\mu(x'/R_n - \theta_{wn}))}$

In each case Whitworth described how the boundary could move and what the tension in the contact region is. The values of $\xi(t)$ were shown to move in the forward machine direction or vanish to $0$. Whitworth shows that material that moves into the region of adhesion (II) cannot move backwards into the region of entry slip (I). The boundary, $\beta(t)$, was found to always move forward at a rate slower than the rotation rate of the roller when $\left|\frac{\mathrm{d}t_{n-1}}{\mathrm{d}t}\right| < \frac{\mu v_n}{R_n}t_n$ is true (slow change in upstream tension). [45]

Fast changing upstream tension is the case when $\xi(t)$ is non-zero. But, Whitworth shows that it is a discontinuous function and can only be approximated, not found definitely. Error in the approximation is found to be the limiting factor in numerical integration of the differential equations with a limit of the form $\max_n \left(\frac{\mu v_n}{R_n}\right)\delta t << 1$ where $n$ refers to the index of every roller in the system being modeled. [45]

Whitworth notes that the entry slip region and the exit slip region are formed in independent ways. The entry slip region depends on the rate of change of the tension in the upstream span while the exit region slip depends on the relationship between the upstream and downstream span tensions. Whitworth shows that there are cases where the two slip regions seem to meet even when the relation between the upstream and downstream tension are not enough to cause

total slip over the roller. When this happens the values of tension in the contact region are

$$0 \leq x' \leq R_n\xi \qquad t^c(t) = t_{n-1}e^{\pm(\mu x'/R_n)} \tag{E.11}$$

$$R_n\xi \leq x' \leq R_n\theta_{wn} \qquad t^c(t) = t_n e^{\pm\mu(\theta_{wn}-x'/R_n)} \tag{E.12}$$

where the signs are either both positive or both negative and $\beta = \xi$. The value of $\xi$ can be found by equating the two equations for tension in the contact region at the boundary $x' = R_n\xi$.

$$t_{n-1}e^{\pm(\mu x'/R_n)} = t_n e^{\pm\mu(\theta_{wn}-x'/R_n)} \tag{E.13}$$

$$e^{\pm(\mu\xi)} = \frac{t_n}{t_{n-1}} e^{\pm\mu(\theta_{wn}-\xi)} \tag{E.14}$$

$$\frac{e^{\pm(\mu\xi)}}{e^{\pm\mu(\theta_{wn}-\xi)}} = \frac{t_n}{t_{n-1}} \tag{E.15}$$

$$e^{(\pm\mu\xi\mp\mu\theta_{wn}\pm\mu\xi)} = \frac{t_n}{t_{n-1}} \tag{E.16}$$

$$(\pm 2\mu\xi \mp \mu\theta_{wn}) = \ln\left(\frac{t_n}{t_{n-1}}\right) \tag{E.17}$$

$$\pm 2\mu\xi = \pm\mu\theta_{wn} + \ln\left(\frac{t_n}{t_{n-1}}\right) \tag{E.18}$$

$$\xi = \frac{\theta_{wn}}{2} \pm \frac{1}{2\mu}\ln\left(\frac{t_n}{t_{n-1}}\right) \tag{E.19}$$

where the sign is determined by $-\text{sign}\left(\frac{\mathrm{d}t_{n-1}}{\mathrm{d}t}\right)$. Equation (E.19)[1] is important because it is used to find $\xi$, but it is also used to show that differentiation of $\xi$ is possible. The differentiation shows that if the derivative of the upstream and downstream tensions are well defined, then the derivative of $\xi$ is well defined. The problem Whitworth shows is that $\frac{\partial t^c(t)}{\partial x'}$ is not defined (there is a cusp). He shows that $\frac{\partial t^c(t)}{\partial x'}$ has to continuously change from $-\mu t^c/R_n$ to $+\mu t^c/R_n$ in a finite distance. The finite distance needed for that change in the partial of the tension in the contact region is defined as the region of adhesion, so even though the two slip regions appear to meet, they do not. A small, but finite, region of adhesion exists between them (practically, the distance can be thought of as a point.). [45]

[1] Whitworth left out a 2 in the denominator; compare (E.19) to Whitworth's [2.3.44] in [45].

## E.1 Determining the Length of Web between Rollers

In this section, the total length of web between two rollers is found. The point of adhesion at the end of the last section is important here because it can be used to delineate the starting point and ending point of a span. The point is called $x_n$. This is worst case while still having an individual span. If the point $x_n$ can not be found, the roller has lost all adhesion and the span is undefined.

Assuming the point $x_n$ exists on roller $n$ and the point $x_{n-1}$ exists on roller $n-1$, then the following integral sums up the span length. Let $e_n(t)$ be the position independent strain in the material between rollers, $e^c_n(x,t)$ be the strain in the contact region of roller $n$, and $e^c_{n-1}(x,t)$ be the strain in the contact region of roller $n-1$.

$$L^e_{n-1} = \int_{x_{n-1}}^{\theta_{w,n-1}R_n} \frac{\mathrm{d}x'}{1+e^c_{n-1}(x',t)} + \frac{L_n}{1+e_{n-1}(t)} + \int_0^{x_n} \frac{\mathrm{d}x'}{1+e^c_n(x',t)} \tag{E.20}$$

Assuming linear elastic material, Hooke's Law ($\sigma = E\epsilon$) applies. By multiplying by the cross-sectional area, tension in the span may be obtained which means the the previous equation can be written in terms of tensions.

$$\begin{aligned} L^e_{n-1} = \int_{x_{n-1}}^{\theta_{w,n-1}R_n} \frac{E_{n-1}A_{n-1}\mathrm{d}x'}{E_{n-1}A_{n-1}+t^c_{n-1}(x',t)} + \frac{E_{n-1}A_{n-1}L_{n-1}}{E_{n-1}A_{n-1}+t_{n-1}(t)} \\ + \int_0^{x_n} \frac{E_nA_n\mathrm{d}x'}{E_nA_n+t^c_n(x',t)} \end{aligned} \tag{E.21}$$

The effective length of the span $n-1$ is given by (E.21) and the change in length can be found in two ways: accounting for the incoming and outgoing web material or differentiating (E.21). The incoming and outgoing material equation is:

$$\frac{\mathrm{d}L^e_{n-1}}{\mathrm{d}t} = \frac{v_{n-1}-\frac{\mathrm{d}x_{n-1}}{\mathrm{d}t}}{1+e^c_{n-1}(x_{n-1},t)} - \frac{v_n-\frac{\mathrm{d}x_n}{\mathrm{d}t}}{1+e^c_n(x_n,t)} \tag{E.22}$$

$$\frac{\mathrm{d}L^e_{n-1}}{\mathrm{d}t} = \frac{E_{n-1}A_{n-1}\left(v_{n-1}-\frac{\mathrm{d}x_{n-1}}{\mathrm{d}t}\right)}{E_{n-1}A_{n-1}+t^c_{n-1}(x_{n-1},t)} - \frac{E_nA_n\left(v_n-\frac{\mathrm{d}x_n}{\mathrm{d}t}\right)}{A_nE_n+t^c_n(x_n,t)} \tag{E.23}$$

The differentiation method is involved but Whitworth shows that with substitutions, the equation

is given by the following:

$$\begin{aligned}\frac{\mathrm{d}L^e_{n-1}}{\mathrm{d}t} =& \frac{E_{n-1}A_{n-1}\left(v_{n-1}-\frac{\mathrm{d}x_{n-1}}{\mathrm{d}t}\right)}{E_{n-1}A_{n-1}+t^c_{n-1}(x_{n-1},t)} - \frac{E_nA_n\left(v_n-\frac{\mathrm{d}x_n}{\mathrm{d}t}\right)}{A_nE_n+t^c_n(x_n,t)} - \frac{E_{n-1}A_{n-1}v_{n-1}}{t^\beta_{n-1}+E_{n-1}A_{n-1}} \\ &+ \frac{E_nA_nv_n}{t^\xi_n+E_nA_n} + \frac{A_{n-1}E_{n-1}}{t_{n-1}+A_{n-1}E_{n-1}}\frac{\mathrm{d}L_{n-1}}{\mathrm{d}t} - \frac{A_{n-1}E_{n-1}L_{n-1}}{\left(t_{n-1}+A_{n-1}E_{n-1}\right)^2}\frac{\mathrm{d}t_{n-1}}{\mathrm{d}t} \\ &- \frac{A_{n-1}E_{n-1}R_{n-1}}{\mu t_{n-1}}\frac{\mathrm{d}t_{n-1}}{\mathrm{d}t}\left|\frac{1}{t_{n-1}+A_{n-1}E_{n-1}} - \frac{1}{t^\beta_{n-1}+A_{n-1}E_{n-1}}\right| \\ &- \frac{A_nE_nR_n}{\mu t_{n-1}}\frac{\mathrm{d}t_{n-1}}{\mathrm{d}t}\left|\frac{1}{t^\xi_n+A_nE_n} - \frac{1}{t_{n-1}+A_{n-1}E_{n-1}}\right| \end{aligned} \tag{E.24}$$

Subtracting (E.23) from (E.24) removes the length change effects due to incoming and outgoing velocities and leaves the derivative of the span length in terms of span tension derivative, and velocity in and out.

$$\begin{aligned}\frac{v_n}{t^\xi_n+E_nA_n} - \frac{\phi v_{n-1}}{t^\beta_{n-1}+E_{n-1}A_{n-1}} + \frac{\phi}{t_{n-1}+A_{n-1}E_{n-1}}\frac{\mathrm{d}L_{n-1}}{\mathrm{d}t} =& \\ \left[\frac{\phi L_{n-1}}{\left(t_{n-1}+A_{n-1}E_{n-1}\right)^2} + \frac{\phi R_{n-1}}{\mu_{n-1}t_{n-1}}\left|\frac{1}{t_{n-1}+A_{n-1}E_{n-1}} - \frac{1}{t^\beta_{n-1}+A_{n-1}E_{n-1}}\right|\right.& \\ \left.+\frac{R_n}{\mu_nt_{n-1}}\left|\frac{1}{t^\xi_n+A_nE_n} - \frac{1}{t_{n-1}+A_{n-1}E_{n-1}}\right|\right]\frac{\mathrm{d}t_{n-1}}{\mathrm{d}t}& \end{aligned} \tag{E.25}$$

$$\phi = \frac{A_{n-1}E_{n-1}}{A_nE_n} \tag{E.26}$$

The coefficient of friction between the web and roller in (E.25)[2] is roller specific and hence the addition of subscript. The ratio of web properties from span to span is quantified in $\phi$. Usually, the properties are assumed constant, but cases could be made for non-constant properties with laminations, printing, slitting, compressing, and other industrial processes used in web handling.

If tension is assumed to be very small compared to $A_nE_n$ and the difference in roller speed compared to line speed is very small, then (E.25) becomes:

$$t_n << A_nE_n \tag{E.27}$$

$$\frac{|v_n - U_L|}{U_L} << 1 \tag{E.28}$$

[2] Whitworth has a typographical error in his Equation [2.4.13] where the last term in absolute value bars is negated. See the progression from Eqn. [2.4.11] to [2.4.12].

$$\frac{v_n}{t_n^{\xi}+E_nA_n}-\frac{\phi v_{n-1}}{t_{n-1}^{\beta}+E_{n-1}A_{n-1}}+\frac{\phi}{t_{n-1}+A_{n-1}E_{n-1}}\frac{\mathrm{d}L_{n-1}}{\mathrm{d}t}=$$

$$\left[\frac{\phi L_{n-1}}{\left(t_{n-1}+A_{n-1}E_{n-1}\right)^2}+\frac{\phi R_{n-1}}{\mu_{n-1}t_{n-1}}\left|\frac{1}{t_{n-1}+A_{n-1}E_{n-1}}-\frac{1}{t_{n-1}^{\beta}+A_{n-1}E_{n-1}}\right|\right.$$
$$\left.+\frac{R_n}{\mu_nt_{n-1}}\left|\frac{1}{t_n^{\xi}+A_nE_n}-\frac{1}{t_{n-1}+A_{n-1}E_{n-1}}\right|\right]\frac{\mathrm{d}t_{n-1}}{\mathrm{d}t} \tag{E.29}$$

$$\frac{v_n\left(t_{n-1}+A_{n-1}E_{n-1}\right)}{t_n^{\xi}+E_nA_n}-\frac{\phi v_{n-1}\left(t_{n-1}+A_{n-1}E_{n-1}\right)}{t_{n-1}^{\beta}+E_{n-1}A_{n-1}}+\phi\frac{\mathrm{d}L_{n-1}}{\mathrm{d}t}=$$
$$\left[\frac{\phi L_{n-1}}{\left(t_{n-1}+A_{n-1}E_{n-1}\right)}+\frac{\phi R_{n-1}}{\mu_{n-1}t_{n-1}}\left|1-\frac{\left(t_{n-1}+A_{n-1}E_{n-1}\right)}{t_{n-1}^{\beta}+A_{n-1}E_{n-1}}\right|\right.$$
$$\left.+\frac{R_n}{\mu_nt_{n-1}}\left|\frac{\left(t_{n-1}+A_{n-1}E_{n-1}\right)}{t_n^{\xi}+A_nE_n}-1\right|\right]\frac{\mathrm{d}t_{n-1}}{\mathrm{d}t} \tag{E.30}$$

Applying a common denominator to the right-hand-side within the absolute values obtains:

$$\frac{v_n\left(t_{n-1}+A_{n-1}E_{n-1}\right)}{t_n^{\xi}+E_nA_n}-\frac{\phi v_{n-1}\left(t_{n-1}+A_{n-1}E_{n-1}\right)}{t_{n-1}^{\beta}+E_{n-1}A_{n-1}}+\phi\frac{\mathrm{d}L_{n-1}}{\mathrm{d}t}=$$
$$\left[\frac{\phi L_{n-1}}{\left(t_{n-1}+A_{n-1}E_{n-1}\right)}+\frac{\phi R_{n-1}}{\mu_{n-1}t_{n-1}}\left|\frac{t_{n-1}^{\beta}-t_{n-1}}{t_{n-1}^{\beta}+A_{n-1}E_{n-1}}\right|\right.$$
$$\left.+\frac{R_n}{\mu_nt_{n-1}}\left|\frac{t_{n-1}-t_n^{\xi}}{t_n^{\xi}+A_nE_n}\right|\right]\frac{\mathrm{d}t_{n-1}}{\mathrm{d}t} \tag{E.31}$$

The ratio $\phi$ is applied to convert $A_nE_n$ to $A_{n-1}E_{n-1}$:

$$\frac{v_n\left(A_{n-1}E_{n-1}\right)\left(t_{n-1}^{\beta}+E_{n-1}A_{n-1}\right)-v_{n-1}\left(A_{n-1}E_{n-1}\right)\left(\phi t_n^{\xi}+E_{n-1}A_{n-1}\right)}{\left(E_{n-1}A_{n-1}\right)\left(E_nA_n\right)}+\phi\frac{\mathrm{d}L_{n-1}}{\mathrm{d}t}=$$
$$\left[\frac{\phi L_{n-1}}{\left(A_{n-1}E_{n-1}\right)}+\frac{\phi R_{n-1}}{\mu_{n-1}t_{n-1}}\left|\frac{t_{n-1}^{\beta}-t_{n-1}}{t_{n-1}^{\beta}+A_{n-1}E_{n-1}}\right|+\frac{R_n}{\mu_nt_{n-1}}\left|\frac{t_{n-1}-t_n^{\xi}}{t_n^{\xi}+A_nE_n}\right|\right]\frac{\mathrm{d}t_{n-1}}{\mathrm{d}t} \tag{E.32}$$

Canceling out an $A_{n-1}E_{n-1}$ from the top and bottom of the left-hand-side and applying the as-

sumption from (E.27) obtains:

$$\frac{\phi v_n(t^{\beta}_{n-1} + E_{n-1}A_{n-1}) - \phi v_{n-1}(\phi t^{\xi}_n + E_{n-1}A_{n-1})}{(E_{n-1}A_{n-1})} + \phi\frac{\mathrm{d}L_{n-1}}{\mathrm{d}t} =$$
$$\left[\frac{\phi L_{n-1}}{(A_{n-1}E_{n-1})} + \frac{\phi R_{n-1}}{\mu_{n-1}t_{n-1}}\left|\frac{t^{\beta}_{n-1} - t_{n-1}}{A_{n-1}E_{n-1}}\right| + \frac{\phi R_n}{\mu_n t_{n-1}}\left|\frac{t_{n-1} - t^{\xi}_n}{A_{n-1}E_{n-1}}\right|\right]\frac{\mathrm{d}t_{n-1}}{\mathrm{d}t} \tag{E.33}$$

$$v_n(t^{\beta}_{n-1} + E_{n-1}A_{n-1}) - v_{n-1}(\phi t^{\xi}_n + E_{n-1}A_{n-1}) + (A_{n-1}E_{n-1})\frac{\mathrm{d}L_{n-1}}{\mathrm{d}t} =$$
$$\left[L_{n-1} + \frac{R_{n-1}}{\mu_{n-1}t_{n-1}}\left|t^{\beta}_{n-1} - t_{n-1}\right| + \frac{R_n}{\mu_n t_{n-1}}\left|t_{n-1} - t^{\xi}_n\right|\right]\frac{\mathrm{d}t_{n-1}}{\mathrm{d}t} \tag{E.34}$$

$$A_{n-1}E_{n-1}\left[v_n - v_{n-1} + \frac{\mathrm{d}L_{n-1}}{\mathrm{d}t}\right] + (t^{\beta}_{n-1}v_n) - v_{n-1}\phi t^{\xi}_n =$$
$$\left[L_{n-1} + \frac{R_{n-1}}{\mu_{n-1}t_{n-1}}\left|t^{\beta}_{n-1} - t_{n-1}\right| + \frac{R_n}{\mu_n t_{n-1}}\left|t_{n-1} - t^{\xi}_n\right|\right]\frac{\mathrm{d}t_{n-1}}{\mathrm{d}t} \tag{E.35}$$

Using the assumption in (E.28) to linearize the nonlinear combination of velocities and contact region tensions, (E.35) becomes:

$$L^{e}_{n-1}\frac{\mathrm{d}t_{n-1}}{\mathrm{d}t} = A_{n-1}E_{n-1}\left[v_n - v_{n-1} + \frac{\mathrm{d}L_{n-1}}{\mathrm{d}t}\right] - U_L\left(\phi t^{\xi}_n - t^{\beta}_{n-1}\right) \tag{E.36}$$

$$L^{e}_{n-1} = \left[L_{n-1} + \frac{R_{n-1}}{\mu_{n-1}t_{n-1}}\left|t^{\beta}_{n-1} - t_{n-1}\right| + \frac{R_n}{\mu_n t_{n-1}}\left|t_{n-1} - t^{\xi}_n\right|\right] \tag{E.37}$$

Whitworth uses (E.36) and (E.37) to draw three conclusions:

I. The final equation is independent of the arbitrary locations $x_{n-1}$ and $x_n$

II. The inclusion of $\frac{\mathrm{d}}{\mathrm{d}t}L_{n-1}$ in the final equation allows for effects such as eccentricity and roll "flats" to affect the tension

III. Though nonlinear, the final equation is homogeneous

## E.2 Temporary Loss of Adhesion on a Roller

In this section, assume that roller $n$ in Figure E.2 is slipping and that rollers $n-1$ and $n+1$ have at least a point of adhesion at all times. When roller $n$ slips, the derivative of span tension is

not defined because that equation was derived using a control volume approach and simplifying. When slip occurs at roller $n$, the boundary of the control volume no longer exists. Effectively, the two spans are made into one plus the web wrapped over the slipping roller.

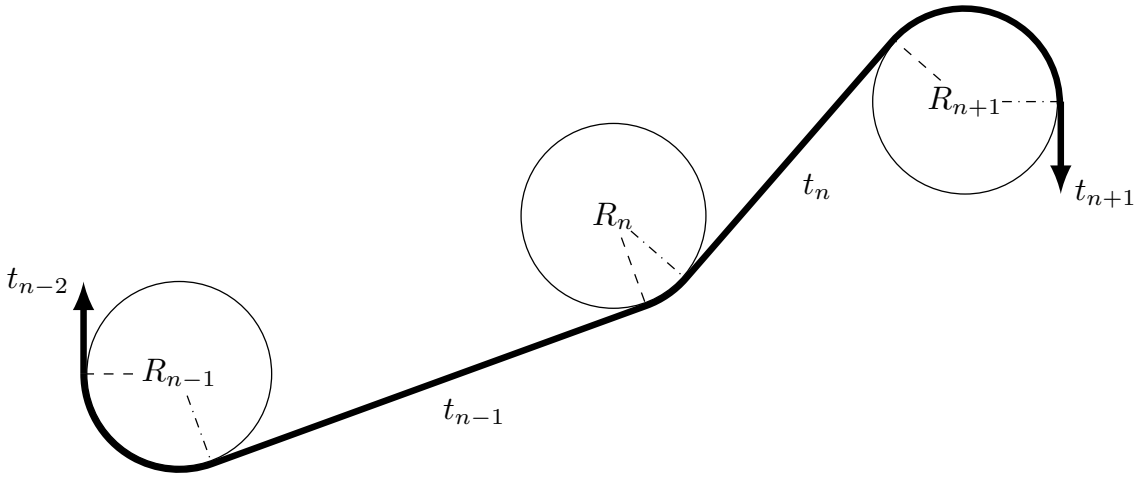



Figure E.2: Example web path through three rollers with wrap angles indicated by dashed and dash-dotted lines.

Let

$$\chi_n = e^{\pm \mu_n \theta_{wn}} = \frac{t_n}{t_{n-1}} \tag{E.38}$$

when slipping is occurring.

Whitworth shows that similar equations to (E.36) and (E.37) can be derived for the total span, called ${}_sL_o$, between some point on roller $n-1$, $x_{n-1}$, to another arbitrary point on roller $n+1$, $x_{n+1}$. Since the derivative of span tension is not defined, the analogous equation to (E.36) can not be used for a span ending at roller $n$. Another way to think of this is that the system of differential equations becomes very *stiff* when slip occurs. The problem remains that the derivative of tension is undefined across a slipping roller. Whitworth circumvents the problem by looking at variables that are continuous. He focused on the strain in the web.

Let $e_n^a$ be the strain in span $n$ assuming adhesion. Let $e_n^s$ be the strain in span $n$ assuming slip occurred at roller $n$. Then sum up the total material in the system between roller $n-1$ and roller

$n+1$. For the adhesion case, the following statement results:

$$\int_{\beta R_{n-1}}^{\theta_{wn-1}R_{n-1}} e^a_{n-1} e^{(\pm\mu x'/R_{n-1})}\mathrm{d}x' + L_{n-1}e^a_{n-1} + \int_0^{x_n} e^a_{n-1} e^{(\pm\mu x'/R_n)}\mathrm{d}x' + \int_{x_n}^{\theta_{wn}R_n} e^a_n e^{(\pm\mu(x'/R_n-\theta_{wn}))}\mathrm{d}x' + L_n e^a_n + \int_0^{\xi R_{n+1}} e^a_n e^{(\pm\mu x'/R_{n+1})}\mathrm{d}x' \tag{E.39}$$

And for the slipping case, the following statement results:

$$\int_{\beta R_{n-1}}^{\theta_{wn-1}R_{n-1}} e^s_{n-1} e^{(\pm\mu x'/R_{n-1})}\mathrm{d}x' + L_{n-1}e^s_{n-1} + \int_0^{\theta_{wn}R_n} e^s_{n-1} e^{(\pm\mu x'/R_n)}\mathrm{d}x' + L_n\chi_n e^s_{n-1} + \int_0^{\xi R_{n+1}} e^s_{n-1}\chi_n e^{(\pm\mu x'/R_{n+1})}\mathrm{d}x' \tag{E.40}$$

Evaluating (E.39) and (E.40) shows that (E.39) equals $e^a_{n-1}L^e_{n-1} + e^a_n L^e_n$ and (E.40) equals $e^s_{n-1}\left(L^e_{n-1} + \chi_n L^e_n\right)$. Whitworth's assumption is that these two quantities are equal. Hooke's Law is applied to put the equation in terms of tensions.

$$e^a_{n-1}L^e_{n-1} + e^a_n L^e_n = e^s_{n-1}\left(L^e_{n-1} + \chi_n L^e_n\right) \tag{E.41}$$

$$t^a_{n-1}L^e_{n-1} + t^a_n L^e_n = t^s_{n-1}\left(L^e_{n-1} + \chi_n L^e_n\right) \tag{E.42}$$

$$t^s_{n-1} = \frac{t^a_{n-1}L^e_{n-1} + t^a_n L^e_n}{\left(L^e_{n-1} + \chi_n L^e_n\right)} \tag{E.43}$$

Using (E.38), the tension in span $n$ when roller $n$ is slipping can be found:

$$t^s_n = \chi_n t^s_{n-1} \tag{E.44}$$

which is because Whitworth assumes that $t^a_n/t^a_{n-1} = \chi_n(1+\delta)$ where $\delta$ is small. The $\delta$ is controllable by reducing the time step in the integration routine used to solve the above equations.

Much later in [45, pg. 151-2], Whitworth states that during the numerical solving of the differential equations he had laid out, the value of $L^e_n$ could reasonably be equated to the largest value of the average span length. That is the last time Whitworth mentions $L^e_n$. He does not use the derived equation for it, (E.21). The journal version of Whitworth's dissertation, [46], does not mention the estimate of $L^e_n$ at all.

# APPENDIX F

# NORMAL FORCE FOR SLIDING-FRICTION DRIVEN ROLLER MODEL

The Sliding Friction Driven Roller (SFDR) model requires a friction force to be applied at the periphery of the idle roller. Coulomb friction was assumed.

$$F_f = \mu_n F_n \tag{F.1}$$

A simple static force balance was applied to an idle roller to define an average resultant normal force for calculating the friction force. The normal force is assumed to bisect the wrap angle of the roller giving $\delta = \theta_{on} + \theta_{wn}/2 + \pi$.

Referencing Figure F.1, the summation of forces in the horizontal direction is

$$-t_{n-1}\sin(\theta_{in}) + t_n \sin(\theta_{on}) - F_n \cos(\delta) = 0 \tag{F.2}$$

$$F_n = \frac{-t_{n-1}\sin(\theta_{in} + t_n \sin(\theta_{on})}{\cos(\delta)} \tag{F.3}$$

where $\delta \neq (2n-1)\frac{\pi}{2}$ and as a check, $\theta_{in} > \theta_{on}$ for a clockwise rotating roller. If $\delta = (2n-1)\frac{\pi}{2}$, the equation for $F_n$ has to change to (F.4) which is the vertical direction summation of forces. The resultant force $F_n$ is assumed to be approximately the normal force on the roller due to the web.

$$F_n = \frac{t_{n-1}\cos(\theta_{in}) - t_n \sin(\theta_{on})}{\sin(\delta)} \tag{F.4}$$

If the roller is rotating counter clockwise, the equation for the normal force is (F.5) for $\delta \neq n\pi$.

$$F_n = \frac{-t_{n-1}\cos(\theta_{in}) - t_n \cos(\theta_{on})}{\sin(\delta)} \tag{F.5}$$

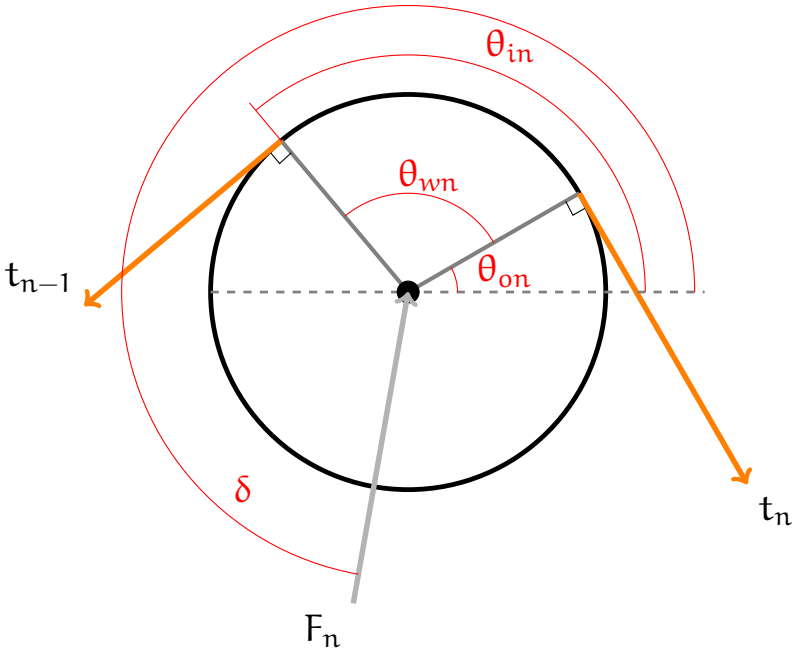



Figure F.1: Force Balance on a Roller Rotating Clockwise

for $\theta_{in} < \theta_{on}$. If $\delta = n\pi$, use (F.6).

$$F_n = \frac{t_{n-1}\sin(\theta_{in}) - t_n \sin(\theta_{on})}{\cos(\delta)} \tag{F.6}$$

# APPENDIX G

# TEST PLAN FOR THE HIGH-SPEED WEB LINE

Test plans for the High-Speed Web Line (HSWL) experiments are recorded in this chapter. The speed profile for the test is a start-up which means the line begins at rest with tension applied to the spans and accelerates to the speed for the test remains at that speed for approximately ten (10) seconds and then deccelerates to rest. Tension levels on the HSWL are in the units of PLI. The actual tension force in the web varies based on different web widths. The test plan will maintain the PLI tension measurement between the different runs and different materials.

Experimentation on the HSWL originally followed the plan shown in Table G.1 in the Appendix. On or about test number 9, the test plan fell apart. The PET material could not be accurately handled at speeds above 800 FPM due to air entrainment and loss of traction on the lateral web guide rollers. Then 24 inch wide Tyvek was used and tests 17 through 24 were accomplished followed by the same tests with 6 inch wide Tyvek. All of these runs were accomplished using the Rockwell gains.

Reviewing the run data showed that the dancer in the unwind section was not operating correctly. The load cells and dancer were calibrated using hanging weights on a rope that followed the web path. The calibration showed that the dancer had a large stiction which, when overcome, would push the dancer to the far end of the track. The pressure in the cylinder was about 60 psi when the stiction was overcome. Programmatic band-aids did not fix the problem and eventually the dancer as a feedback device was abandoned as spare parts and instructions on disassembly are

not available. The unwind section was left with only load cell feedback control.

Gain values for both the Rockwell method and the RA method for nine experiments with three materials all at 24 lbf tension in the web are shown in Table G.2. These come from a separate test matrix that followed the one below. The process laid out in section G.2 was followed for each trial. Only a load cell was used for feedback on the unwind section as noted above.

## G.1 Test Matrix

The Table G.1 shows the list of experiments considered in this document. The matrix will be executed for the HSWL with two controller gain calculation methods: the current method that Rockwell Automation installed and the gain calculation method described in chapter 3.

Table G.1: Test Plan Matrix

| Test Number | Web Material | | Speed Range (FPM) | Control Feedback | | | |
|---|---|---|---|---|---|---|---|
| | | | | Unwind | | Rewind | |
| | PET | Tyvek | | Dancer[1] | LC[2] | Dancer[3] | LC[2] |
| 1 | X | | 400 | X | | | X |
| 2 | X | | 400 | | X | | X |
| 3 | X | | 400 | X | | X | |
| 4 | X | | 400 | | X | X | |
| 5 | X | | 800 | X | | | X |
| 6 | X | | 800 | | X | | X |
| 7 | X | | 800 | X | | X | |
| 8 | X | | 800 | | X | X | |
| 9 | X | | 1600 | X | | | X |
| 10 | X | | 1600 | | X | | X |
| 11 | X | | 1600 | X | | X | |
| 12 | X | | 1600 | | X | X | |
| 13 | X | | 2400 | X | | | X |
| 14 | X | | 2400 | | X | | X |
| 15 | X | | 2400 | X | | X | |
| 16 | X | | 2400 | | X | X | |
| 17 | | X | 400 | X | | | X |
| 18 | | X | 400 | | X | | X |
| 19 | | X | 400 | X | | X | |
| 20 | | X | 400 | | X | X | |
| 21 | | X | 800 | X | | | X |
| 22 | | X | 800 | | X | | X |
| 23 | | X | 800 | X | | X | |
| 24 | | X | 800 | | X | X | |
| 25 | | X | 1600 | X | | | X |
| 26 | | X | 1600 | | X | | X |
| 27 | | X | 1600 | X | | X | |
| 28 | | X | 1600 | | X | X | |
| 29 | | X | 2400 | X | | | X |
| 30 | | X | 2400 | | X | | X |
| 31 | | X | 2400 | X | | X | |
| 32 | | X | 2400 | | X | X | |

[1] Linearly translating dancer

[2] LC means load cell

[3] S-Wrap dancer

## G.2 Test Plan for an Individual Experiment

I. Start RSLogix if it is not already running

II. In RSLogix, bring up the Trend that will be used with the specific control method

III. Ensure the web material is the one assigned in the Test Plan Matrix, Table G.1, for this experiment

Follow these steps to change materials:

(a) Run the existing material almost entirely to one of the unwind or rewind rolls

(b) Cut the web free from the empty roll

(c) Cut the web free from the full roll, leaving the web in the machine to help pull the new material through the Bypass path

(d) Use the hoist to remove the empty roll and the full roll

(e) Bring in the new parent roll and place it on the unwind motor using the hoist

(f) Attach the loose end of the new material to the strip of old material that is threaded through the machine with tape

(g) Install an empty core and spindle on the rewind motor

(h) Attach the old web end to the empty core with tape

(i) Use 'Maintenance Mode' to slowly pull the new material through the machine

(j) Once the new material reaches the rewind, stop the machine

(k) Remove the tape holding the new material to the old material

(l) Unwind the old material from the core

(m) Tape the new material to the empty core
Wind a wrap or two by hand if there is material to help ensure the connection holds.

IV. Ensure the air pressure driven clamps on the unwind and rewind rolls are pressurized

V. Measure the diameters of the unwind and rewind rolls with a $\pi$-tape and record the measurements.

VI. Inspect the HSWL for continuous web and ensure no debris is in contact with the web

VII. Ensure the web path is the Bypass path which skips the wrinkle module, nip station 2, and several rollers over those two modules

The bypass path is depicted in Figure G.1

VIII. Bring up the HMI screen on the computer

The HMI is the program on the start bar called 'RSVeiw32 Works 32K', if it is minimized.

IX. Ensure the unwind and rewind rolls' motors are setup correctly with the wrap direction and the current diameter of the rolls

X. Set the tension level in both the unwind and rewind section to be the same value: 3 PLI

XI. Ensure only the unwind, master speed control, and rewind motors are enabled

The motor control screen describes them as DM01, DM03, and DM09, respectively. The 'Drive Setup' screen contains the enable/disable switches for the motors as well as the line direction and material properties input.

XII. Ensure the machine operating direction is set to FORWARD

The HSWL defines FORWARD as web moving from the DM01 motor to the DM09 motor. In Figure G.1, FORWARD is from the Unwind stand to the Rewind Stand.

XIII. For the unwind and rewind motors, select the control mode that is assigned by the Test Plan Matrix, Table G.1

(a) Dancer means select the 'Dancer - Speed' option for the unwind motor

(b) LC means select the 'Tension - Speed' option for the unwind motor

(c) Dancer means select the 'Dancer - Speed' option for the rewind motor

(d) LC means select the 'Tension - Speed' option for the rewind motor

XIV. Ensure the web material properties are correct for the material type selected in the Test Plan Matrix, Table G.1

XV. On the 'Nip Control' screen, ensure the Nip Stand 1, Nip 1 is CLOSED

The HSWL will not operate if this nip roller is open. Click the word, OPEN, with the mouse to close the nip.

XVI. On the 'Unwind Section' screen, enter the measured diameter of the unwind by clicking the diameter set point. Enter the diameter on the on-screen keypad that appears. Click enter. Then click the 'Set Diameter' push button.

XVII. On the 'Rewind Section' screen, enter the measured diameter of the rewind by clicking the diameter set point. Enter the diameter on the on-screen keypad that appears. Click enter. Then click the 'Set Diameter' push button.

XVIII. On the 'Running' screen, ensure 'Line Speed 1 Setpoint' in the upper left corner of the screen is set to the value of the speed range in Table G.1 for this experiment

XIX. Ensure the speed setpoint selector button just to the right of the Line Speed 1&2 Setpoints says 'Speed Setpoint 1'

Click the button with the mouse if it does not say 'Speed Setpoint 1'.

XX. Click the START button at the top of the screen

A horn will sound 5 times.

XXI. Within 5 seconds of the last horn blast, click the START button again

The line applies tension to the web at this point. The motors will slowly rotate and, if dancers are used, the dancers will move to their middle points of travel.

XXII. Bring up RSLogix and click the 'Run' button on the Trend to start data acquisition

XXIII. Bring up the HMI screen again

XXIV. Click START once more to begin the acceleration

Click the STOPbutton at any time to decelerate the line to rest. Any of the Emergency Stop buttons distributed around the HSWL will also stop the line but with larger deceleration rates.

XXV. Wait while the HSWL accelerates to full speed

XXVI. Continue to allow RSLogix to record data for 10 seconds

XXVII. Click the STOPbutton to decelerate the line to rest (tension is removed, as well)

XXVIII. Bring up RSLogix once the line has stopped

XXIX. Click the Stop button on the Trend screen to stop recording data

XXX. Click the 'Log' button on the Trend screen and select 'Save Trend As…'

A Save dialog box appears and the name of the trend should be displayed in the file name box. Add a run number to the end of the file name to discriminate it from other runs using this Trend. Add comments about the run conditions and any failures noted.

XXXI. In the file type option of the dialog box, select '*.CSV' so that Excel can easily open the file

XXXII. Ensure the directory is where the Trends are saved

XXXIII. Click Save

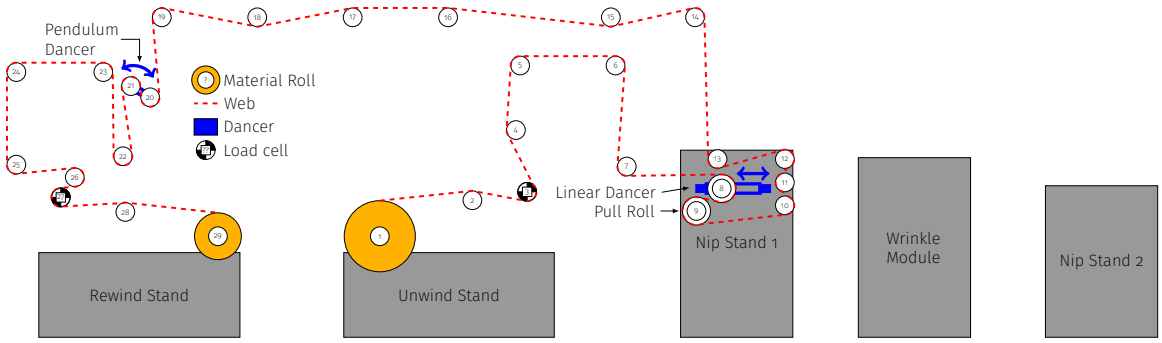



Figure G.1: High-Speed Web Line with stands showing the Bypass path for the web as viewed from the computer

Table G.2: Experimental Beginning and Ending of Run Gains for the HSWL Unwind Motor, 0-800 FPM Start-up

| Gain Calc. Method | Material | Width (in) | Speed $K_p$ Beg. | Speed $K_p$ End | Speed $K_i$ Beg. | Speed $K_i$ End | Tension $K_p$ Beg. | Tension $K_p$ End | Tension $K_i$ Beg. | Tension $K_i$ End | Radius (ft) Beg. | Radius (ft) End |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Rockwell | Tyvek | 6 | 10.77 | 10.00 | 4.31 | 4.00 | 100.83 | 81.93 | 1008.27 | 819.29 | 0.53 | 0.46 |
| RA | Tyvek | 6 | 10.91 | 9.86 | 111.34 | 100.66 | 86.96 | 73.75 | 335.72 | 284.72 | 0.68 | 0.62 |
| RA | Tyvek | 6 | 9.73 | 8.68 | 99.30 | 88.60 | 71.91 | 55.18 | 277.59 | 213.01 | 0.61 | 0.54 |
| Rockwell | PET | 6 | 12.15 | 11.67 | 4.86 | 4.67 | 123.16 | 118.08 | 1231.61 | 1180.81 | 0.62 | 0.60 |
| RA | PET | 6 | 14.16 | 13.08 | 144.55 | 133.51 | 19.05 | 17.52 | 73.55 | 67.65 | 0.60 | 0.57 |
| RA | PET | 6 | 13.22 | 11.93 | 134.90 | 121.80 | 17.72 | 15.74 | 68.41 | 60.75 | 0.58 | 0.54 |
| Rockwell | Tyvek | 24 | 39.05 | 32.64 | 15.62 | 13.06 | 56.68 | 60.15 | 566.76 | 601.47 | 0.76 | 0.72 |
| RA | Tyvek | 24 | 25.34 | 20.78 | 258.61 | 212.12 | 23.76 | 20.57 | 91.72 | 79.39 | 0.71 | 0.66 |
| RA | Tyvek | 24 | 12.34 | 9.74 | 125.90 | 99.39 | 12.66 | 8.92 | 48.88 | 34.43 | 0.52 | 0.43 |

# APPENDIX H

# EXPERIMENTS USING THE EUCLID WEB LINE

## H.1 Setup

After receiving an EncoderOutlet.com TR1 encoder, the job of installing it into the Rockwell Automation control system was ahead. Once complete, the data from the encoder was collected coincidentally with other data from the control system. This avoided the problem of multiple data acquisition systems having to be aligned. Rockwell Automation donated the Allen-Bradley High-Speed Counter (HSC) module to the WHRC to allow the encoder acquisition.

The encoder was initially set up as an 'Encoder 1X' format in the Control Logix program for the EWL. There is a problem in the Control Logix network that the data from the HSC is not updated at every 10ms query. Rockwell has looked at this without success. This means that there are zeros in the data stream which play havoc with numerical differentiation when attempting to find the speed from position data. The encoder was switched to 'Continuous Rate Frequency Mode' which describes how many 4MHz pulses are counted within the pulse of the encoder. Based on knowing how much arc length is described by one pulse, the speed can be found. The mode does not leave a zero in the data stream if it is not updated. Instead, it reuses the last value which is much better for calculations.

One last change to the setup of the HSC for the TR1 encoder was to modify how many encoder pulses were counted before the number of 4MHz pulses was saved to the data stream. Through

calculations, 46 encoder pulses were found to fit within 5 ms. Since that system is limited to multiples of 2 for the number of counts, 32 encoder pulses was the selected value. That means that the HSC counts the number of 4MHz pulses required for 32 encoder pulses to register. Since the TR1 encoder has 4096 pulses per revolution, this happens 128 times in a revolution of the wheel and provides a more accurate estimation of the wheel speed. Also, the update rate on the EWL is 10 ms so at speed, the Rockwell control software will have new data every half millisecond. The conversion from counts to $ft/s$ is given in (H.1).

$$S_{meas}(n) = \frac{1}{2}\frac{32[Pulses]}{\frac{250\cdot10^{-9}[s]}{[Pulse]}\cdot n[Pulses]}\frac{\frac{1}{2}[ft/rev.]}{4096[Pulses/rev.]} \tag{H.1}$$

The calculated value of $S_{meas}$ is the tangential speed in $ft/s$ the edge of the encoder wheel is traveling and $n$ is the number of pulses recorded by the HSC. The value of $1/2ft/rev.$ is the circumference of the encoder wheel.

## H.2 Initial Measurements

The TR1 encoder was placed on idler 9, which is also the load cell for the unwind section. Data was recorded (28 May 2018) at three line tensions (8.6, 12.7, and 19.4 lbf) and at five line speeds (150, 200, 300, 400, 450 FPM). The data recorded were the TR1 counts, S-Wrap speed, Pull roll speed, IR sensor counts, Dancer position, Wheel Stored Value, and Unwind section tension. All recorded runs were in the Forward direction and the Unwind diameter was 14 inches.

### H.2.1 Plots

The data was plotted by tension level and line speed. This showed the wheel encoder was consistently slower on average than the S-wrap. The FFT showed a 1-per-rev at the S-wrap speed in the encoder data. The encoder speed when placed on the web instead of the idler was found to be faster than the idler speed, but not as fast as the S-wrap on average. There were peaks in the data above the S-wrap speed. These findings indicate that slip is occurring.

## H.3 Torque Addition Effect on Web Speed

Ron Lynch gave Dr. Reid the idea of applying an additional drag torque to an idler to induce slip. This guarantees slip occurs which was questionable in previous tests. His method involved draping a rope or thin web over the idler in question and fixing one end while hanging weights from the other to allow for varying the applied torque.

### H.3.1 Setup

This trial started with setting up the addition torque application device. Cloth, a large clothes pin, zip-ties, and string were acquired for the project. Dr. Reid wanted to have tension measurements on both sided of the wheel so a tension staged to the left of the idler and the cloth was attached to its hood via string and a zip-tie. The other end was secured by passing the clothes pin through the cloth (See Figure 5.2). Weights were added to the clothes pin in the case of the washers and hung on it in the case of the 0.5 lbf weights. The TR1 encoder was positioned on the web for this set of trials to capture the web speed.

### H.3.2 Test Plan

The following procedure was used to acquire data for this trial. The data collected during the trial by the trend function of the RSLogix 5000® software was the TR1 encoder counts, the IR sensor counts, the Unwind, S-wrap, and Pull roll speeds, the unwind section tension, and the dancer position. The following data were acquired by hand before, during, or after the run: Unwind and rewind diameters, amount of weights on the cloth, number of washers on the cloth, pre-run tension indicated on the tension meter after hanging the weight on the cloth, the tension after the line has settled into it 400 FPM setpoint, and the tension meter reading just after the line stops moving after the stop command. These are tabulated in Table H.1 with notes in Table H.3.

I. Starting with the unwind diameter on the PanelView at just indicating 15 inches, set the web guide on the unwind to center

II. Hang the weights for the run

III. Record the pre-run tension reading

IV. Initiate the data collection

V. Start the web guide for the rewind section

VI. Engage the line to perform a start-up to 400 FPM at ∼5 lbf line tension (the air pressure regulator was set at 7)

VII. Wait for the line to settle in at 400 FPM, then record the dynamic tension reading from the tension meter

VIII. Monitor the unwind diameter on the PanelView. When it indicates 13 inches, apply the stop command

IX. Record the post-run tension just after the line comes to rest

X. Stop the data collection

XI. Save the trend data as a `*.csv` file type

XII. Remove the cloth and weight from the hook on the meter

XIII. Set the unwind web guide to auto

XIV. Turn off the rewind web guide

XV. Set the line direction to reverse

XVI. Press start to initiate a start-up to 400 FPM

XVII. Monitor the unwind diameter while allowing the line to run in reverse until the unwind diameter reads 15 inches

XVIII. Stop the line

XIX. Set the line direction to forward

XX. Repeat steps 1–20 for all hanging weight increments

### H.3.3 Discussion

The original cloth broke during this battery of trials. See Note 6. A second cloth was obtained with rings sewn into the ends to make connections. Runs 8–11 used the second cloth.

### H.3.4 Plots

The data is compared by how much hanging weight is used. The S-wrap speed and web speed are compared. This information will facilitate understanding how to calculate web speed as compared to roller speed in simulations.

Table H.1: Slip Trial Hand Collected Data

| Run | Unwind Diameter (in) | Rewind Diameter (in) | Weights (lbf) | Washers | Static Tension (lbf) | Dynamic Tension (lbf) | Post Static Tension (lbf) | Notes |
|---|---|---|---|---|---|---|---|---|
| 1 | 14 | 13 | 0.5 | 8 | 0.57 | 1.2 | 1.04 | 1,3,12 |
| 2 | 14 | 13 | 1 | 8 | 0.84 | 2.005 | 1.8 | 1,3,12 |
| 3 | 15 | 12 | 1.5 | 8 | 1.23 | 2.82 | 2.51 | 1,3,12 |
| 4 | 15 | 12 | 2 | 8 | 1.4 | 3.68 | 3.25 | 1,3,12 |
| 5 | 15 | 12 | 2.5 | 8 | 2.18 | 4.32 | 3.7 | 1,3,12 |
| 6 | 15 | 12 | 5 | 8 | 3.73 | 7.88 | 6.8 | 1,2,3,12 |
| 7 | 15 | 12 | 2.5 | 33 | 2.72 | 5.85 | 4.93 | 1,4,12 |
| 8a | 15 | 12 | 2.5 | 58 | | | | 2,4,5,6,12 |
| 8 | 15 | 12 | 2.5 | 58 | 2.99 | 6.45 | 5.32 | 2,4,5,7,12 |
| 9 | 15 | 12 | 2 | 33 | 2.12 | 5.34 | 4.43 | 5,7,12 |
| 10 | 15 | 12 | 2 | 25 | 1.95 | 4.6 | 4.13 | 7,12 |
| 11 | 15 | 12 | 2 | 50 | 2.47 | 5.9 | 5.08 | 4,7,8,12 |

Table H.2: Slip Trial Hand Collected Data

| Run | Unwind Diameter (in) | Rewind Diameter (in) | Weights (lbf) | Washers | Static Tension (lbf) | Dynamic Tension (lbf) | Post Static Tension (lbf) | Notes |
|---|---|---|---|---|---|---|---|---|
| 12 | 15 | 12 | 2 | 50 | 2.46 | 5.41 | 4.5 | 4,7,9 |
| 13 | 15 | 12 | 2 | 45 | 2.59 | 5.5 | 4.67 | 4,7,9 |
| 14 | 15 | 12 | 2 | 40 | 2.37 | 5.07 | 4.35 | 4,7,9,10 |
| 15 | 15 | 12 | 2 | 35 | 2.13 | 5.05 | 4.32 | 4,7,9 |
| 16 | 15 | 12 | 2 | 33 | 2.15 | 4.9 | 3.93 | 4,7,9 |
| 17 | 15 | 12 | 2 | 8 | 1.76 | 3.96 | 3.29 | 7,9 |
| 18 | 15 | 12 | 2 | 10 | 1.46 | 4.13 | 3.64 | 7,9 |
| 19 | 15 | 12 | 2 | 15 | 1.835 | 4.25 | 3.74 | 7,9 |
| 20 | 15 | 12 | 2 | 20 | 1.79 | 4.38 | 3.71 | 7,9 |
| 21 | 15 | 12 | 2 | 18 | 2.09 | 4.39 | 4 | 7,9 |
| 22 | 15 | 12 | 2 | 16 | 2.03 | 4.31 | 3.56 | 7,9 |
| 23 | 15 | 12 | 2 | 14 | 1.39 | 4.19 | 3.55 | 7,9,11 |
| 24 | 15 | 12 | 0 | 0 | 0 | 0 | 0 | 7,9,13 |
| 25 | 15 | 12 | 0 | 8 | 0.19 | 0.4 | 0.26 | 7,9,14 |
| 26 | 15 | 12 | 0.5 | 8 | 0.59 | 1.19 | 1.18 | 7,9 |
| 27 | 15 | 12 | 1 | 8 | 0.83 | 2.06 | 1.97 | 7,9 |
| 28 | 15 | 12 | 1.5 | 8 | 1.35 | 2.88 | 2.55 | 7,9 |
| 29 | 15 | 12 | 2 | 8 | 1.42 | 3.85 | 3.34 | 7,9 |
| 30 | 15 | 12 | 2 | 8 | 1.55 | 4.14 | 3.51 | 7,9,15 |
| 31 | 15 | 12 | 2.5 | 8 | 1.63 | 4.9 | 3.9 | 7,9,16 |

Table H.3: Slip Trial Hand Collected Data Notes

| # | Note |
|---|---|
| 1 | Cloth, zip-tie, safety pin not included in weight. 400FPM top speed. |
| 2 | Roller stopped. |
| 3 | Weight per washer: 0.027lbf |
| 4 | Red string washer set |
| 5 | Blue string washer set |
| 6 | The cloth broke |
| 7 | Second cloth with sewn in rings for attachment used. |
| 8 | Idler came to rest just as the stop button for the end of the test was pressed (reached 13 inches unwind diameter). |
| 9 | Wheel encoder on idler instead of web |
| 10 | Not sure on the value of the Post tension measurement. Waited too long to write it down. |
| 11 | Idler ran without stopping the whole test, but at a visibly slower speed than the line. |
| 12 | Wheel encoder on web at idler 9 |
| 13 | Baseline, no weight, no cloth |
| 14 | Just cloth, clothes pin, and 8 washers needed to keep clothes pin in place |
| 15 | Accidently left off the last weight, so reran the 2 lbf trial a second time |
| 16 | Idler stopped repeatedly during trial |

## H.4 Torque Addition Effect on Roller Speed

This set of trials also used Ron Lynch's idea. His method involved draping a rope or thin web over the idler in question and fixing one end while hanging weights from the other to allow for varying the applied torque. Here, the focus is on the roller speed.

### H.4.1 Setup

The setup is the same as for section H.3.1 except for the placement of the encoder. The TR1 encoder was positioned in contact with the idler for this set of trials to capture the idler speed.

### H.4.2 Test Plan

The procedure in section H.3.2 was used to acquire data for this trial. The hand collected data are tabulated in Table H.2 with notes in Table H.3.

The second cloth was used during all the runs.

### H.4.3 Plots

Plotting idler speed over time by hanging weight. Figure H.1 shows the start up to shutdown nature of the test being performed. The S-wrap roller is speed controlled, so the different hanging weight values do not make a difference in its speed. The same kind of plot of the idler shows that there is a drop in idler speed at around 2.5lbf of hanging weight (see top plot in Figure H.2). The difference between the idler stopping and starting and continually running at a low speed is between 2.89 and 2.83lbf (see middle plot in Figure H.2).

Using the data collected by hand and applying a force balance to the roller free body, the required frictional force can be found. Figure H.3 shows the results of the tabulated friction force versus hanging weight.

The condition of the surface of the rollers in the EWL is not good. Rust spots abound. The previous data was collected with the rust on the rollers. The rust was removed and the surface of the roller was cleaned. Figure H.4 shows the trials of roller 9 after rust removal. (The friction force data is shown on Figure H.3.) Again, an additional torque was applied to the roller. As the torque increased, the roller speed fell off, but with less consistency than before (Figure H.2). Here, the idler stopped at a much lower torque than above, but then run continually with more breaking torque applied (see 1.446 as compared to 1.5 in Figure H.4).

To check other parameter effects, a roller with a very small wrap angle was tested and a roller with a smaller diameter was tested. Figure H.5 shows the time history of the two runs. No additional torque was applied to either roller. Idler 6a with a small wrap angle did not attain a relatively constant speed. It was increasing during the whole run. Idler 6 is part of a web guide and has smaller diameter and a smaller inertia. It ran at 394.46 FPM with a standard deviation of 0.28 FPM.

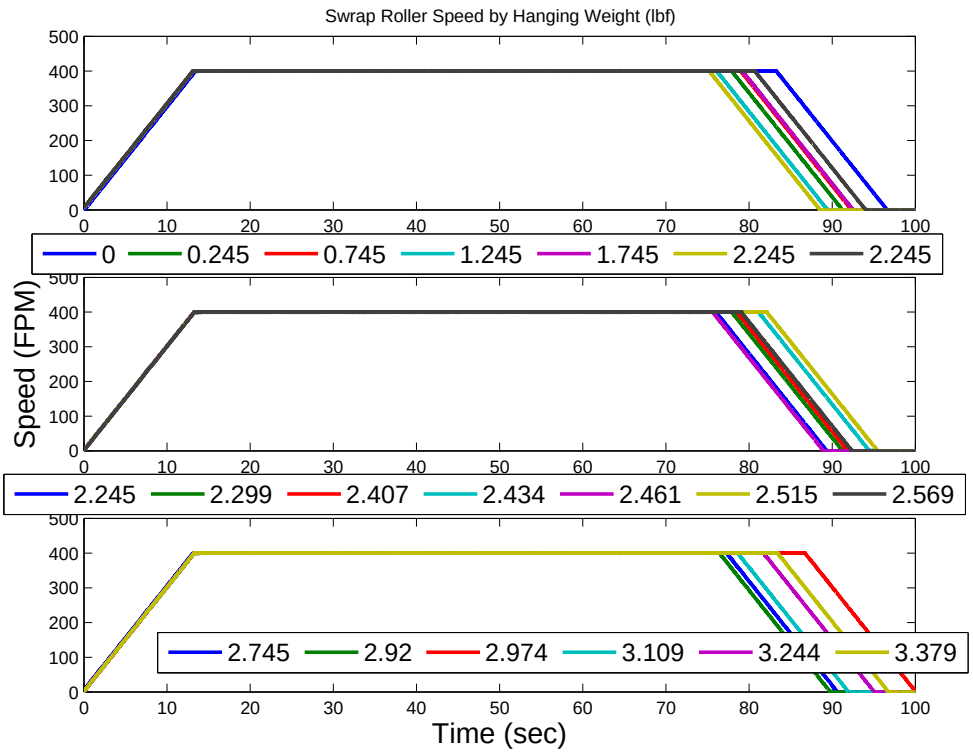



Figure H.1: Time History of S-wrap Roller Speeds for each Hanging Weight. This data is presented to show the duration of the test from rest to 400FPM to rest.

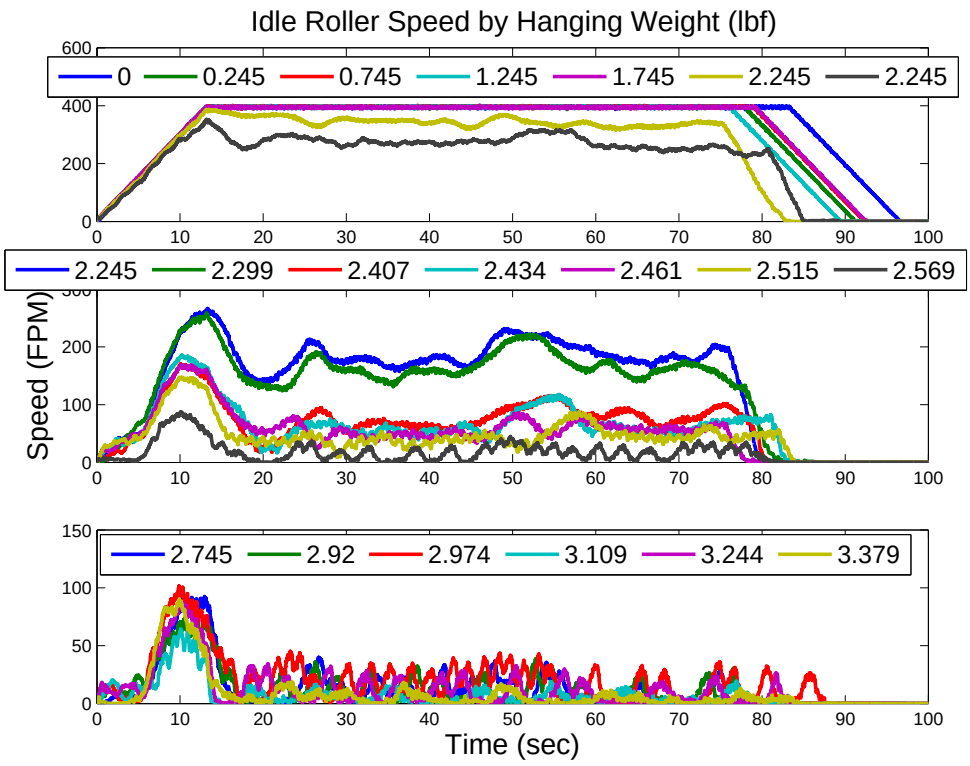



Figure H.2: Time History of the Wheel Encoder Running on the Idler. Note the decrease in speed as the hanging weight increases. At hanging weights greater than 2.89lbf the idler stops and starts (see the black line on the middle plot).

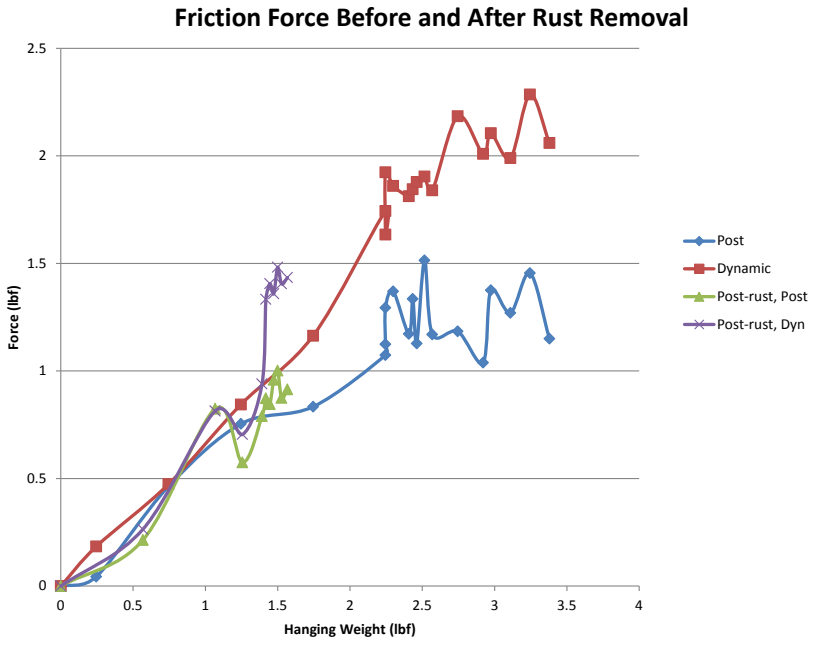



Figure H.3: Friction force from static force balance by hanging weight. The green and purple lines show the post-rust removal friction which is at almost a pound less hanging weight than the pre-rust removal friction forces.

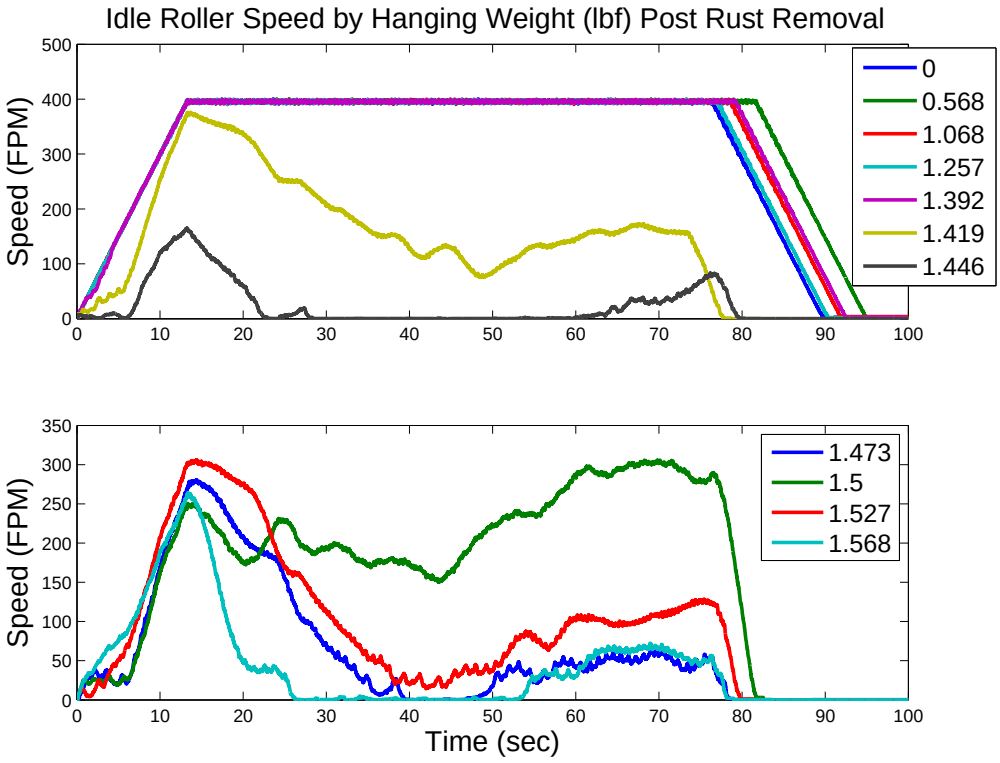



Figure H.4: Time History of the Wheel Encoder Running on the Idler Post Rust Removal. Note the decrease in speed as the hanging weight increases and the reduced weight required to cause slip.

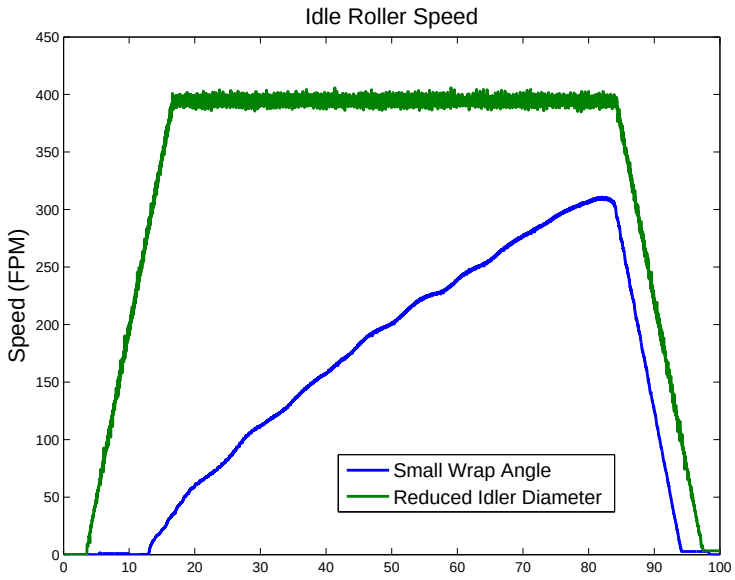



Figure H.5: Time History of the Wheel Encoder Running on Idler 6a and 6. Idler 6a (▬) has a small wrap angle and Idler 6 has a reduced diameter, 2.234 inches as compared to 3 inches for the others tested.

## H.5 Friction Coefficient Measurement

The coefficient of friction can be measured between a roller and a web. The capstan equation is utilized to back out the coefficient of friction based on recorded tension data and wrap angle. Slip is defined as a noticeable to the eye relative movement of the web compared to the roller. Collecting the data over many runs allows averaging out the calculated value and increases the level of confidence in the confidence interval.

### H.5.1 Setup

Obtain a dead weight between 5lbf and 20lbf, a couple metal bars, string, a tension meter or load cell, and a sample of the web. Select a roller that allows easy access, an open space below it in which the dead weight can hang, and mounting of the tension meter, preferably in line with the top edge of the roller to ensure a $90^\circ$ wrap angle. Mount the tension meter so that the web will be perpendicular to the roller edge. Use one metal bar to make a connection to the web for the tension meter and the other bar for hanging the dead weight. Figure H.6 shows an example. The tension meter was moved to the gray bar just behind it for the actual test upon reflection. The meter was clamped to the bar for support.

### H.5.2 Method

The following procedure was used to obtain data on the tension held by the web/roller interface, just before slip occurred. The data collected were the dead weight value, the wrap angle, and the tension on the tension meter just before slip.

I. With the setup complete, hang the dead weight on the web

II. Record the static tension from the meter

III. With a gloved hand, slowly twist the roller in the direction that applies more tension to the tension meter

IV. Watch the tension meter readout

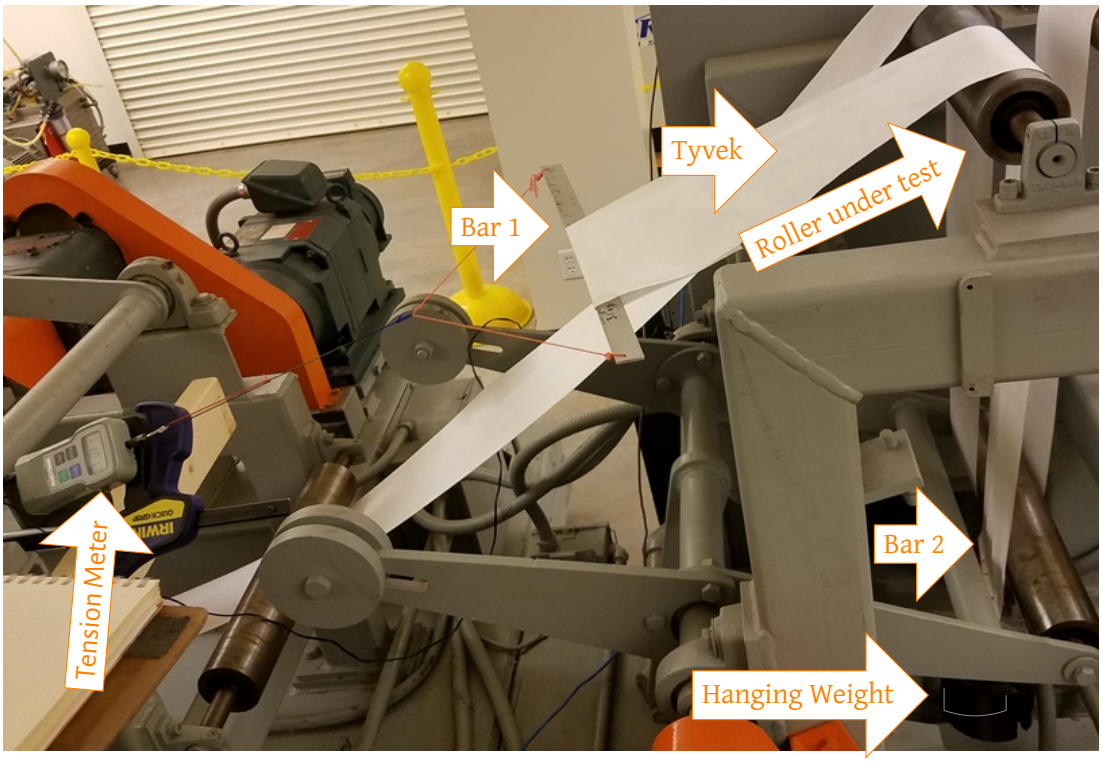



Figure H.6: Hardware Setup for Coefficient of Friction Test. The load cell measures tension on one side of the roller/web interface while the hanging weight applies tension to the other.

V. Apply more twist, checking the meter repeatedly, and watch for slip between the web and roller

VI. Once slip is witnessed, record the tension measurement just before slip occurred

VII. Repeat steps II–VI more than ten times to increase statistical significance of the measurements

### H.5.3 Data Analysis

The recorded tension, hanging weight, and wrap angle are used in the capstan equation to back out the coefficient of friction, $\mu$. The capstan equation is

$$\frac{T_h}{T_l} = e^{\mu\theta} \tag{H.2}$$

where $T_h$ is the high tension, $T_l$ is the low tension, and $\theta$ is the wrap angle in radians. Taking the natural log of both sides and dividing through by $\theta$, the coefficient of friction is obtained.

$$\mu = \frac{1}{\theta} \ln\left(\frac{T_h}{T_l}\right) \tag{H.3}$$

From the data in Table H.4, the average coefficient of friction of roller 13 before rust removal was 0.18 and after rust removal was 0.15. For roller 9, the average coefficient of friction before rust removal was 0.24 and after rust removal, it was 0.12 (see Table H.5).

Table H.4: Coefficient of Friction Data for Roller 13 Before Rust Removal and After Rust Removal. The * is added to the data that was taken statically. The wrap angle used is $105^\circ$.

| Pre-Rust Removal | | | Post Rust Removal | | |
|---|---|---|---|---|---|
| $T_l$ (lbf) | $T_h$ (lbf) | $\mu$ | $T_l$ (lbf) | $T_h$ (lbf) | $\mu$ |
| 11.80* | 10.96* | 0.04* | 11.80* | 11.47* | 0.02* |
| 11.80 | 17.68 | 0.22 | 11.80 | 15.45 | 0.15 |
| 11.80 | 16.68 | 0.19 | 11.80 | 15.38 | 0.14 |
| 11.80 | 16.35 | 0.18 | 11.80 | 15.50 | 0.15 |
| 11.80 | 16.95 | 0.20 | 11.80 | 15.99 | 0.17 |
| 11.80 | 15.86 | 0.16 | 11.80 | 16.00 | 0.17 |
| 11.80 | 15.38 | 0.14 | 11.80 | 15.55 | 0.15 |
| 11.80 | 15.53 | 0.15 | 11.80 | 16.09 | 0.17 |
| 11.80 | 16.50 | 0.18 | 11.80 | 15.84 | 0.16 |
| 11.80 | 16.83 | 0.19 | 11.80 | 15.02 | 0.13 |
| 11.80 | 16.60 | 0.19 | 11.80 | 15.42 | 0.15 |
| 11.80 | 16.28 | 0.18 | 11.80 | 15.09 | 0.13 |
| 11.80 | 16.19 | 0.17 | 11.80 | 15.05 | 0.13 |
| 11.80 | 15.64 | 0.15 | 11.80 | 15.09 | 0.13 |
| 11.80 | 16.70 | 0.19 | 11.80 | 16.20 | 0.17 |
| 11.80 | 15.84 | 0.16 | 11.80 | 16.09 | 0.17 |
| 11.80 | 16.28 | 0.18 | 11.80 | 15.70 | 0.16 |
| 11.80 | 16.17 | 0.17 | 11.80 | 15.32 | 0.14 |
| 11.80 | 15.99 | 0.17 | 11.80 | 15.48 | 0.15 |

Table H.5: Coefficient of Friction between Web and Roller 9 Before and After Rust Removal. The wrap angle used is $88°$.

| Pre-Rust Removal | | | Post-Rust Removal | | |
|---|---|---|---|---|---|
| $T_l$ (lbf) | $T_h$ (lbf) | $\mu$ | $T_l$ (lbf) | $T_h$ (lbf) | $\mu$ |
| 16.70 | 25.86 | 0.2849 | 11.80 | 11.33 | 0.0265 |
| 16.70 | 25.25 | 0.2694 | 11.80 | 14.15 | 0.1182 |
| 16.70 | 25.01 | 0.2631 | 11.80 | 13.8 | 0.1019 |
| 16.70 | 24.37 | 0.2463 | 11.80 | 14.12 | 0.1169 |
| 16.70 | 24.29 | 0.2441 | 11.80 | 14.13 | 0.1173 |
| 16.70 | 24.67 | 0.2542 | 11.80 | 14.09 | 0.1155 |
| 16.70 | 24.88 | 0.2598 | 11.80 | 14.21 | 0.1210 |
| 16.70 | 24.08 | 0.2385 | 11.80 | 14.2 | 0.1205 |
| 16.70 | 24.41 | 0.2473 | 11.80 | 13.98 | 0.1104 |
| 16.70 | 23.92 | 0.2341 | 11.80 | 14.19 | 0.1201 |
| 16.70 | 23.97 | 0.2355 | 11.80 | 14.21 | 0.1210 |
| 16.70 | 24.22 | 0.2423 | 11.80 | 14.38 | 0.1287 |
| 16.70 | 23.33 | 0.2179 | 11.80 | 14.14 | 0.1178 |
| 16.70 | 24.44 | 0.2481 | 11.80 | 14.3 | 0.1251 |
| 16.70 | 24.74 | 0.2561 | 11.80 | 14.2 | 0.1205 |
| 16.70 | 23.98 | 0.2358 | 11.80 | 14.17 | 0.1192 |
| 16.70 | 23.07 | 0.2106 | 11.80 | 14.26 | 0.1233 |
| 16.70 | 23.68 | 0.2276 | 11.80 | 14.08 | 0.1150 |
| 16.70 | 23.63 | 0.2262 | 11.80 | 14.13 | 0.1173 |
| 16.70 | 24.15 | 0.2404 | 11.80 | 14.21 | 0.1210 |
| 16.70 | 24.25 | 0.2431 | | | |
| 16.70 | 23.81 | 0.2311 | | | |
| 16.70 | 23.39 | 0.2195 | | | |
| 16.70 | 23.8 | 0.2309 | | | |
| | Average: | 0.2419 | | Average: | 0.1185 |

## H.6 Matlab ODE45 Events Issue

MATLAB has a capability to check for events defined by a function going to zero during the execution of the ODE45 command. To use this, define the myevents() function to have an element go to zero as the event of interest happens. MATLAB will check this event function every iteration step and when the sign of the the previous event function evaluation times the current event function evaluation is negative, an event happened. MATLAB will then use the two time steps to be the bounds for a bisection method of finding the actual zero point in time of the event function. This method works for things like limit events (pendulum swing, tension going to zero), but for slip, it does not work.

The reason for it not working in the case of slip is unknown. A hypothesis is that in backing up to the point in time where the slip criteria event went to zero, MATLAB restarts the integration at a point where the slip could happen or not. Thus, it seems to generally choose not to happen. Since the zero finding routine can not be shut off in the ODE45 algorithm, another method must be used. If it just found that slip could happen and adjusted the tensions based on the Whitworth equation, then slip would continue to happen. But, since it goes back to the zero point, there is no "momentum" in the slip direction.

To use ODE45 anyway, the slip checking algorithm was placed in the derivative execution. That way, every time the derivative is evaluated, the initial tension set is checked against the slip conditions. Since the derivative is evaluated 7 times per iteration step in ODE45, that means a lot of slip checks. This is problematic when the initial tensions that caused the slip condition are wanted for various reasons. To just save the data every time the slip condition is met will net many thousands of data points. Also the variable time step capability of the ODE45 algorithm causes it to back up and redo iterations based on failing an error check which would add repeated data (possibly) to the set.

A recording function was created to run in the background of MATLAB while the ODE45 algorithm is running which will only save the latest data set for a specific time and only save one set per 0.005 seconds. The function lodges itself in the MATLAB workspace and will only be re-

moved by '`clear`'-ing the function name. Therefore, an internal function mechanism was created to clear its memory (an "initialize" function) and a printing mechanism to return the event data when asked.

The method works for `ODE45` and for a Runge-Kutta $4^{th}$ order integration routine. With the `RK4` method, when the slip condition is checked can be chosen much like WTS (based on the help files of [66]).

## H.7 The Nip Roller Derivation

Adding a nip roll to a driven roller ensures that web does not slip. The following derivation comes from the Web Transport System - V2.1 Help, [66]. This method negates the down force of the nip on the web. The force is only found such that the tangential speed of the nip is equal to the tangential speed of its roller (see (H.9)). It does not find or use the applied down force of the nip onto the driven roller. Important assumptions include no slip between the web and either roller, there is enough down force between the nip and the driven rollers to ensure the no slip condition, and the quantity of the down force is not important.

Driven roller and nip roller primitive elements:

$$\frac{\mathrm{d}}{\mathrm{d}t}v_n = \frac{1}{J_n}\left[-(B_{fn}+C_{mn})v_n + R_n^2(T_n - T_{n-1}) + R_n K_{mn} u_n - R_n^2 f\right] \tag{H.4}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}v_{nip} = \frac{1}{Jn_n}\left[-Bn_{fn}v_{nip} + Rn_n^2 f\right] \tag{H.5}$$

Solving for $f$ and substituting into the driven roller primitive element:

$$f = \frac{1}{Rn_n^2}\left[\frac{\mathrm{d}}{\mathrm{d}t}v_{nip}Jn_n + Bn_{fn}v_{nip}\right] \tag{H.6}$$

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}v_n = \frac{1}{J_n}\Bigg[ & -(B_{fn}+C_{mn})v_n + R_n^2(T_n - T_{n-1}) + R_n K_{mn} u_n \\ & - R_n^2 \frac{1}{Rn_n^2}\left[\frac{\mathrm{d}}{\mathrm{d}t}v_{nip}Jn_n + Bn_{fn}v_{nip}\right]\Bigg]\end{aligned} \tag{H.7}$$

$$Rn_n^2 J_n \frac{\mathrm{d}}{\mathrm{d}t} v_n = - Rn_n^2 (B_{fn} + C_{mn}) v_n + Rn_n^2 R_n^2 (T_n - T_{n-1}) + Rn_n^2 R_n K_{mn} u_n$$
$$- R_n^2 \frac{\mathrm{d}}{\mathrm{d}t} v_{nip} J n_n - R_n^2 B n_{fn} v_{nip} \tag{H.8}$$

Let the nip speed equal the driven speed and the equate the accelerations:

$$v_n = v_{nip}, \quad \frac{\mathrm{d}}{\mathrm{d}t} v_{nip} = \frac{\mathrm{d}}{\mathrm{d}t} v_n \tag{H.9}$$

$$\left(R_n^2 J n_n + Rn_n^2 J_n\right) \frac{\mathrm{d}}{\mathrm{d}t} v_n = - Rn_n^2 (B_{fn} + C_{mn}) v_n + Rn_n^2 R_n^2 (T_n - T_{n-1})$$
$$+ Rn_n^2 R_n K_{mn} u_n - R_n^2 B n_{fn} v_n \tag{H.10}$$

$$\left(R_n^2 J n_n + Rn_n^2 J_n\right) \frac{\mathrm{d}}{\mathrm{d}t} v_n = - \left(Rn_n^2 (B_{fn} + C_{mn}) + R_n^2 B n_{fn}\right) v_n$$
$$+ Rn_n^2 R_n^2 (T_n - T_{n-1}) + Rn_n^2 R_n K_{mn} u_n \tag{H.11}$$

$$\frac{\mathrm{d}}{\mathrm{d}t} v_n = \frac{1}{(R_n^2 J n_n + Rn_n^2 J_n)} \left[ - \left( Rn_n^2 (B_{fn} + C_{mn}) + R_n^2 B n_{fn} \right) v_n \right.$$
$$\left. + Rn_n^2 R_n^2 (T_n - T_{n-1}) + Rn_n^2 R_n K_{mn} u_n \right] \tag{H.12}$$

# APPENDIX I

# CODES USED FOR ANALYSIS AND SIMULATION OF WEB LINES

Analyzing web lines for problems requires the ability to record tensions and speeds along the web line. Once data is acquired, it may be analyzed for frequency content, maximums, minimums, average values, or standard deviations. Also, verifying that the layout of the web line is what is expected needs a visualization method. Importing data from outside sources has to be handled and certain types of data need to be grouped together so that copy errors are minimized.

## I.1 Fast Fourier Transform

A tool called a Fast Fourier Transform [85] can be used to find frequency information from raw data. It is a algorithm that takes a discrete data set and attempts to fit a group of sinusoids to the raw time domain result. It calculates a group of frequencies and magnitudes of the sinusoid at those frequencies and returns the list of frequencies and magnitudes. With a little mathematical manipulation, the output can be plotted and the plot shows the magnitude of oscillations at given frequencies. The MATLAB implementation of the discrete Fast Fourier Transform is the command, `fft`. It returns a vector of complex numbers that are the magnitudes of components at frequencies that have to be calculated based on the time between data samples. Listing I.1 has the MATLAB code to output the frequency and magnitudes of the raw data stored in variable `ff`.

Listing I.1: Fast Fourier Transform Mathematical Manipulation

```
n = size(ff); %ff  has raw data taken at intervals of T seconds
m=mean(ff);
ff=ff-repmat(m,n(1),1);
tstep = T; % Sampling period
Y = fft(ff);
Fs = 1/tstep;           % Sampling frequency
L = length(ff);              % Length of signal
if mod(L,2) == 1
    L = L - 1;
end
P2 = abs(Y/L);
P1 = P2(1:L/2+1,:);
P1(2:end-1,:) = 2*P1(2:end-1,:);  %Magnitudes
f = Fs*(0:(L/2))/L; %frequencies (Hz)
```

## I.2 Industrial S-Curve

Descriptive statistics of the data set can be found using MATLAB 's built in commands: `mean`, `min`, `max`, and `variance`. These commands will return the average, minimum, maximum, and variance (standard deviation squared), respectively. But, before these commands are used, the raw data must be interpreted for its error. If the web line was supposed to start from 0 RPM and accelerate to 235 RPM, then the `min` will be 0 and the `max` will be 235. The expected value must be removed from the data which will leave the error. The expected value would be the value of the reference ramp for that time interval used to go from 0 to 235 RPM.

The reference ramp used on the EWL and HSWL is the Industrial S-Curve which is not really an S-curve (like a log-sigmoid), but a rounded corner, ramp. It uses inputs that control the acceleration rate and the jerk rate used to move between the starting value to the ending value. The jerk rate controls how round the corners are. Once the acceleration rate has been met, it is held constant until the jerk rate requires the reduction of the acceleration rate to plateau at the ending value. See Figure I.1 for an example. Note the transition points indicated with black dashed lines. From time equal 0 to the first black, dashed line, the Industrial S-Curve maintains constant jerk which causes linear increase of the acceleration rate. Between the two black, dashed lines, the Industrial S-Curve maintains constant acceleration (see the green line). Then, the Industrial S-Curve decelerates to the desired ending speed at a constant, negative jerk rate once again.

The advantages of the Industrial S-Curve are that the jerk and acceleration rates are constants or $0$ and the duration of the transition to the desired speed does not need to be known *a priori*. It is set by the jerk and acceleration rates. That makes calculating the equilibrium torque which requires the acceleration easy. The true S-curve has continuously changing acceleration throughout its transition and the time duration of the transition sets the maximum acceleration. Calculating the changing acceleration rate is cumbersome and calculating the jerk rate even more so.

Investigating the EWL and HSWL lines' control programs (RSLogix), the values of the acceleration rates and the jerk rates were found. Using those two inputs and the starting and ending values of the S-Curve, the Industrial S-Curve can be calculated for any instant in time after the start using Listing I.2, `IndScurveInput`. These values may be subtracted out of the raw data at each time step to obtain the error.

The whole point of the Industrial S-Curve is to impose smooth transitions onto the dynamics of the (in this case) speed demand. The industrial S-Curve is providing the dynamics for this transition in a fixed manner. The Industrial S-Curve outputs three things at any given time: the demand speed, the acceleration rate, and the jerk rate. Thus, instantaneous changes in the jerk rate do not cause large changes in the output, similar to the dynamics of a second order system with no overshoot. The rise time of the industrial S-Curve is not called out, but is imposed by the acceleration rate and jerk rate.

Listing I.2: Industrial S-Curve Code

```
function [ref] = IndScurveInput(t,start_val,end_val,accel_rate,jerk_rate,...
    initialize)
persistent local_time local_time_delta local_start_time local_times
persistent local_accel_change_flag
delta = end_val - start_val;  ref = zeros(3,1);
if nargin == 6
    local_time = 0; local_time_delta = 0; local_start_time = t;
    t0 = accel_rate/jerk_rate;
    t1 = (abs(end_val - start_val) - 1*jerk_rate*t0^2)/accel_rate;
    local_times = [t0, t0+t1];
    if delta > 0
        local_accel_change_flag = 1;
    else
        local_accel_change_flag = -1;
    end
else
    local_time = t - local_start_time;
    if local_time_delta == 0
        local_time_delta = local_time;
    end
end
if delta < 0
    jerk_rate = -jerk_rate;
end
if start_val == end_val
    ref = [end_val; 0; 0];
else
    jerk_rate_store = jerk_rate; %positive acceleration
   if local_time < local_times(1)
        ref = [local_time^2*jerk_rate/2; jerk_rate*local_time; jerk_rate];
   elseif local_time < local_times(2)
        ref = [jerk_rate/2*local_times(1)^2+(local_time-local_times(1))...
            *accel_rate*local_accel_change_flag;...
            accel_rate*local_accel_change_flag; 0];
   elseif local_time < local_times*[1 1]'
        ref = [jerk_rate/2*local_times(1)^2+(local_time - local_times(1))...
            *accel_rate*local_accel_change_flag+(-jerk_rate/2*(local_time -...
            local_times(2))^2);...
            accel_rate*local_accel_change_flag - jerk_rate*(local_time - ...
            local_times(2)); -jerk_rate];
   else
        ref = [(end_val - start_val); 0; 0];
   end
ref(1,1) = ref(1,1) + start_val;
end
end
```

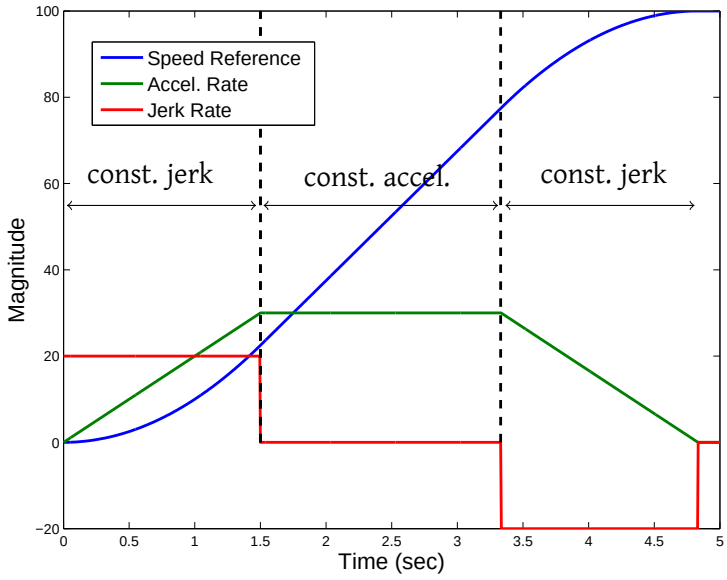



Figure I.1: Example Industrial S-Curve from 0-100 with acceleration rate = 30 and jerk rate = 20

## I.3 Non-Circular Roll Shape Functions

In certain modeling applications, a non-circular roll shape needs to be simulated (discussed in section 2.2.1). A more generalized form of dealing with roll shape and its effects on span tension is covered in [33, 53]. The roll shape impacts a couple of things about the roll: the mass distribution and the radius. A bump on the roll causes a slight eccentricity due to the distribution of mass not being uniform. The `RollShape` function (see Listing I.4) calculates the radius, first and second derivatives of the radius, and is set up to take into account varying radius rolls by updating the `r1i` value when calling the function. Since roll shapes can vary, the function calls another function, `Bump` (see Listing I.3). `Bump` actually calculates the radius and derivatives and hands them back to `RollShape`. In simulation, having a bump on the roll requires an eccentricity to by used. The magnitude of the eccentricity caused by various `RollShape`'s is calculated with the `CGcalc` script (see Listing I.5).

The `CGcalc` script operates by approximating the roll with 360 1° wide, right triangles. The radius is used for one side and the width of 1° at the radius is used for the other side. The centroid is calculated using the two known sides as the base and height. The centroids are summed up to obtain the whole roll centroid which is assumed to be the center of mass of the roll. The center of

Listing I.3: Bump Code

```
function [r, rdot, rddot, rddot2] = Bump(theta,thetadot,r_0,mag,limlo,...
limhi,freq)
%theta (rad), thetadot (radian per sec), r_0 (radius of perfect
%circle in feet), mag (bump height in inches), limlo (polar
%coordinate lower limit of start of bump in rad), limhi (polar
%coordinate upper limit of end of bump in rad), r (radius in ft at
%that theta), rdot (rate of change of radius at that theta in ft per
%sec)
if nargin < 7
    freq = 1;
end
thta = mod(theta,2*pi);
if thta >=  limlo && thta <= limhi
    rdot = (freq*2*pi/(limhi-limlo))*mag/12/2*sin(freq*(thta-limlo)*...
        (2*pi/(limhi-limlo)))*thetadot;  %*(1-exp(-500*(theta-limlo)))
    rddot = (freq*2*pi/(limhi-limlo))^2*mag/12/2*cos(freq*(thta-limlo)*...
        (2*pi/(limhi-limlo)))*thetadot^2*sign(thetadot);
    rddot2 = (freq*2*pi/(limhi-limlo))*mag/12/2*sin(freq*(thta-limlo)*...
        (2*pi/(limhi-limlo)));
else
    rdot = 0;
    rddot = 0;
    rddot2 = 0;
end
r = r_0 +(heaviside(thta-limlo)-heaviside(thta-limhi))*(mag/12/2*...
    (1-cos(freq*(thta-limlo)*(2*pi/(limhi-limlo)))));
end
```

mass is reported in inches from the perfect circle center and degrees as in polar coordinates.

Listing I.4: Roll Shape Selection Code

```
function [r1, r1dot, r1ddot, r1ddot2] = Rollshape(theta,thetadot,r1i,type)
% Returns a radius and rate of change of the radius  for different shapes:
%  type = 1 : bump between 30 and 60 degrees (as measured on Euclid Line)
%  type = 2 : smaller bump
%  type = 3 : flat-ish shape only
%  type = 4 : Use lots of sinusoids to approximate a flat... no
%  otherwise : perfect circle
%
thta = mod(theta,2*pi);
switch type
    case 1
        % bump on circle
        [t1,t2,t3,t4] = Bump(thta,thetadot,r1i,11/32,pi/6,...
            pi*84.5539/180);
        r1 = t1;
        r1dot = t2;
        r1ddot = t3;
        r1ddot2 = t4;
    case 2
        % create a smaller bump
        [t1,t2, t3] = Bump(thta,thetadot,r1i,.25,pi/6,pi/3);
        t4 = 0;
        r1 = t1;
        r1dot = t2+t4;
        r1ddot = t3;
    case 3
        % create a flat on the surface of a roll
        [t1,t2,t3] = Bump(thta,thetadot,r1i,-2.05,pi/4,9*pi/12);
        r1 = t1;
        r1dot = t2;
        r1ddot = t3;
    case 4
        % create a flat on the surface of a roll with a bump
        [t1,t2] = Bump(thta,thetadot,r1i*0,-2.05/3,pi/4,9*pi/12,3);
        [t3, t4] = Bump(thta,thetadot,r1i,-2.05,pi/4,9*pi/12);
        [t5,t6] = Bump(thta,thetadot,r1i*0,-2.05/5,pi/4,9*pi/12,5);
        [t7,t8] = Bump(thta,thetadot,r1i*0,-2.05/7,pi/4,9*pi/12,7);
        [t9,t10] = Bump(thta,thetadot,r1i*0,-2.05/9,pi/4,9*pi/12,9);
        r1 = t1+t3+t5+t7+t9;
        r1dot = t2+t4+t6+t8+t10;
    otherwise
        r1 = r1i;
        r1dot = 0;
        r1ddot = 0;
        r1ddot2 = 0;
end
```

Listing I.5: Center of Gravity Calculator Code

```
phi=pi*1/180:pi*1/180:2*pi;
x=zeros(size(phi,2),2);
clear a r1

for i=1:numel(phi)
    r1(i)=Rollshape(phi(i),0,7/12,1); % Calls Rollshape for given params
    a(i) = pi/360*(r1(i)^2-1.5^2);%1/2*r1(i)^2*cos(1*pi/180)*sin(1*pi/180);
    x(i,1:2) = [phi(i)+.5*pi/180, 2/3*r1(i)];%*cos(.5*pi/180)];
end
c = a*diag(x(:,2))*sin(x(:,1))/(sum(a));
d = (a*diag(x(:,2))*cos(x(:,1)))/sum(a);
phi_c = atan(c/d);
r = sqrt(c^2+d^2);
str = ['\tY Comp.\tX Comp.\tPhase angle\tRadius\n\t(in)\t(in)\t\t(rad)'...
    '\t(in)\n'];
fprintf(1,str)
fprintf(1,'\t%3.5f\t%3.5f\t%3.5f\t\t%3.5f\n',c,d,phi_c,r)
str = 'Calculated center of mass: %3.5f inches at %3.5f degrees\n';
fprintf(1,str,r,phi_c*180/pi)
polar(phi,r1*12)
hold on
polar(x(:,1),x(:,2)*12,'r.')
polar(phi_c,r*12,'kx')
text(.5,3.4,'(in)')
xlabel('(deg)')
hold off
```

## I.4 Pendulum Dancer Functions

If a pendulum dancer is used in the web line (discussed in section 2.2.3), the span lengths between the dancer roller and its previous and next rollers change as the dancer swings. Initial attempts at the pendulum dancer primitive element assumed small angle approximation and linearized the span length change about the nominal value of 0° swing [66]. Including nonlinear effects in the models was an important part of the research. The span lengths had to be solved for online using the `PendDancerLeng` function (see Listing I.6). The tension of the spans around the pendulum dancer on the Euclid Web Line (EWL) was offset by an air cylinder. While the cylinder was perpendicular to the pendulum arm, all the force from the cylinder is applied to countering the tension force from the spans, but as the dancer swings, some of the force is directed along the arm of the dancer so less force is available from the air cylinder to counter the span tension. This phenomena is accounted for with the `TorqueComponent` function (see Listing I.7).

Listing I.6: Pendulum Dancer Span Length Calculation Code

```
function [L1,L1_dot] = PendDancerLeng(theta,theta_dot,L1c,c,K,R1,R2)
% Solves by geometry for the span length and span length rate of change
%
% Syntax:
%   [L1,L1_dot] = PendDancerLeng(theta,theta_dot,L1c,c,K,R1,R2)
%     where
%       theta: angle of pendulum (0 is assumed to be when spans leading
%              into and out of the pendulum are perpendicular to the
%              pendulum arm.
%       theta_dot: angular velocity of the pedulum arm.
%       L1c: length of the span when pendulum arm is at 0 degress.
%       c: length of pendulum arm.
%       K: distance parallel to the pendulum arm between the centers of the
%           rollers on the pendulum and the other end of the span.
%       R1: radius of roller on pendulum.
%       R2: radius of roller at the other end of the span.
%   Output
%       L1: current length of the span.
%       L1_dot: current rate of change of the length of the span.
%

L1 = sqrt((L1c-c*sin(theta)).^2+(K-R1-R2+c*(1-cos(theta))).^2);
L1_dot = 1/2*(-2*c*(L1c - c*sin(theta)).*cos(theta) + 2*c*(K - R2 - R1 +...
    c*(1 - cos(theta))).*sin(theta)).*theta_dot./(L1);
end
```

Listing I.7: Pendulum Dancer Span Length Calculation Code

```
function phi = TorqueComponent(theta,c,d,type,g,R1,L1)
%calculates angle to use cosine of for finding the moment component
%that is available to the pendulum
%
% theta : pendulum angle
% c :  pendulum length
% d :  perpendicular length of the force to its point of origin
%         (from the pivot of the air cylinder to the pendulum arm)
% type : 1 = air cylinder, 2 = span, lower, 3 = span, upper
%
        if type == 1
            phi  = theta + atan(d*(1-cos(theta))/(c - d*sin(theta)));
        elseif type == 2
            psi = theta/2;
            i=0;
            tol =1;
            while tol > 1e-8 && i<10
                gn = sqrt(R1^2+L1^2-2*R1*L1*cos(psi));
                gdot = g-gn;
                psidot = g*gdot/(R1*L1*sin(psi));
                psi = psi+psidot*.001;
                g = gn;
                i = i+1;
                tol = abs(psidot*0.001);
            end
            chi = atan((c-g*cos(theta-psi))/(d-g*sin(theta-psi)));
            phi = cos(theta)*cos(chi)-sin(theta)*sin(chi);
    end
```

## I.5 Plotting the Web Line

One way to known that the web line data being used is the desired one is to plot the layout of the rollers and spans. Initially, the layout is found by inspection of an existing line, use of another tool like WTS [66], or by imagining a line and placing the roller centers. The visual work area of WTS is very helpful in the creation of a layout and the functions in this section expect data from WTS. The `ElementPropReadIn` function (see Listing I.8) takes an export file from WTS and reads it into MATLAB . Table I.1 shows the list of property names which are the rows of the data read in from WTS. Table I.2 is split over two pages because of its length and shows the WTS export file. Unfortunately, these cannot be copied to text file and used because the real WTS export files are fixed-width columns and the `ElementPropReadIn` function is designed to work with the fixed-width columns. Table I.3 is also split over two pages due to its length and it contains the export file from WTS for the Euclid web line. The `ElementPropReadIn` function is used in the web line simulation functions in Appendix J to incorporated the export file from WTS. The `SpanData` function (see Listing I.9) requires the $X$ (horizontal) and $Y$ (vertical) coordinates of the centers of the rollers and the radius of the rollers as well as the rotational direction for each roller. The `SpanData` function returns a matrix of span data which is used by the `WrapAngle` function (see Listing I.10) to calculate angles of wrap on each of the interior rollers of a web line. The `PrintLine` function (see Listing I.11) takes a string containing the path to a file which was exported from WTS. It uses the `SpanData` and `WrapAngle` functions internally to gather the information required to plot the web line layout. Figures I.2 and I.3 are examples of the output of the `PrintLine` function.

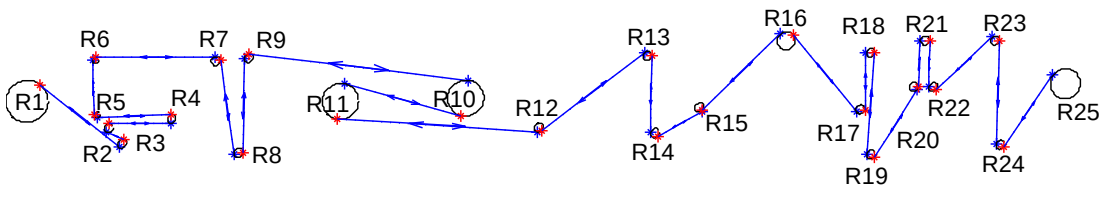



Figure I.2: The Euclid Web Line Schematic Diagram Produced with the `PrintLine` Function.

Table I.1: Element Property Row Labels from WTS Export

| # | Roll Data Rows | Span Data Rows |
|---|---|---|
| 1 | X Position (feet) | Young's Modulus (kpsi) |
| 2 | Y Position (feet) | Web Thickness (mil) |
| 3 | Roller Diameter (inches) | Web Width (inches) |
| 4 | Core Diameter (inches) | Span Length (inches) |
| 5 | Nip Roller Diameter (inches) | Web Density, lbm/($in^3$) |
| 6 | Nip Mass (lbm) | Initial Web Tension (lbf) |
| 7 | Nip Bearing Friction (lbf-ft-sec) | Web RMS Roughness (micro-in) |
| 8 | Nip Inertia (lbf-ft-$sec^2$) | Permeability ($in^3$/s)/($in^2$-psi) |
| 9 | Nip Roller Inertia (ft-lbf-$s^2$) | |
| 10 | Roller Mass (lbm) | |
| 11 | Nip/Dancer Arm Placement (degrees) | |
| 12 | Length of Dancer Arm (inches) | |
| 13 | Roller Inertia (ft-lbf-$s^2$) | |
| 14 | Mass of Core (lbm) | |
| 15 | Roller Core Inertia (lbf-ft-$sec^2$) | |
| 16 | Dancer Drag (lbf-sec/ft)\|(lbf-ft-sec/rad) | |
| 17 | Spring Constant (lbf/ft) | |
| 18 | Motor Constant (lbf-ft/amp) | |
| 19 | Torque/Speed Ratio (lbf-ft-sec) | |
| 20 | Bearing Friction (lbf-ft-sec) | |
| 21 | Static Coefficient of Friction | |
| 22 | Initial Angle of Eccentricity (deg) | |
| 23 | Amount of Eccentricity (inches) | |
| 24 | Initial Velocity (fpm) | |
| 25 | Initial Current (Amp) | |
| 26 | Initial External Force\|Torque (lbf)\|(lbf-in) | |
| 27 | Roller RMS Roughness (micro-in) | |
| 28 | Angle of Wrap (degrees) | |

Table I.2: WTS Export for HSWL.

| | Unw Roll 1 | Idl Roll 2 | Idl Roll 3 | Idl Roll 4 | Idl Roll 5 | Idl Roll 6 | Idl Roll 7 | Idl Roll 8 | Drv Roll 9 | Idl Roll 10 | Idl Roll 11 | Idl Roll 12 | Idl Roll 13 | Idl Roll 14 | Idl Roll 15 | Idl Roll 16 | Idl Roll 17 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| X Position (feet) | 1.28E+01 | 1.55E+01 | 1.72E+01 | 1.68E+01 | 1.70E+01 | 1.98E+01 | 2.01E+01 | 2.29E+01 | 2.22E+01 | 2.48E+01 | 2.48E+01 | 2.48E+01 | 2.28E+01 | 2.21E+01 | 1.96E+01 | 1.48E+01 | 1.20E+01 |
| Y Position (feet) | 3.00E+00 | 4.08E+00 | 4.31E+00 | 6.15E+00 | 8.07E+00 | 8.07E+00 | 5.07E+00 | 4.42E+00 | 3.75E+00 | 3.92E+00 | 4.60E+00 | 5.29E+00 | 5.29E+00 | 9.50E+00 | 9.50E+00 | 9.50E+00 | 9.50E+00 |
| Roller Diameter (inches) | 2.40E+01 | 4.00E+00 | 4.00E+00 | 4.00E+00 | 4.00E+00 | 4.00E+00 | 4.00E+00 | 8.00E+00 | 8.00E+00 | 4.00E+00 | 4.00E+00 | 4.00E+00 | 4.00E+00 | 4.00E+00 | 4.00E+00 | 4.00E+00 | 4.00E+00 |
| Core Diameter (inches) | 4.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Nip Roller Diameter (inches) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 4.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Nip Mass (lbm) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 1.50E+01 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Nip Bearing Friction (lbf-ft-sec) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 2.00E-04 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Nip Inertia (lbf-ft-sec^2) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 1.14E-02 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Nip Roller Inertia (ft-lbf-s^2) * | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 1.14E-02 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Roller Mass (lbm) * | 4.28E+02 | 1.10E+01 | 1.10E+01 | 1.10E+01 | 1.10E+01 | 1.10E+01 | 1.10E+01 | 2.63E+01 | 3.20E+01 | 1.10E+01 | 1.10E+01 | 1.10E+01 | 1.10E+01 | 1.10E+01 | 1.10E+01 | 1.10E+01 | 1.10E+01 |
| Nip/Dancer Arm Placement (degrees) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 1.35E+02 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Length of Dancer Arm (inches) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Roller Inertia (ft-lbf-s^2) * | 5.32E+00 | 8.39E-03 | 8.39E-03 | 8.39E-03 | 8.50E-03 | 8.50E-03 | 8.39E-03 | 8.39E-03 | 1.04E-01 | 8.39E-03 | 8.39E-03 | 8.39E-03 | 8.39E-03 | 8.39E-03 | 8.39E-03 | 8.39E-03 | 8.39E-03 |
| Mass of Core (lbm) | 8.00E+01 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Roller Core Inertia (lbf-ft-se^2) | 5.39E-02 | 8.39E-03 | 8.39E-03 | 8.39E-03 | 8.39E-03 | 8.39E-03 | 8.39E-03 | 8.39E-03 | 1.04E-01 | 8.39E-03 | 8.39E-03 | 8.39E-03 | 8.39E-03 | 8.39E-03 | 8.39E-03 | 8.39E-03 | 8.39E-03 |
| Dancer Drag (lbf-sec/ft)—(lbf-ft-sec/rad) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Spring Constant (lbf/ft) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Motor Constant (lbf-ft/amp) | 1.00E+02 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 1.00E+02 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Torque/Speed Ratio (lbf-ft-sec) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Bearing Friction (lbf-ft-sec) | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 |
| Static Coefficient of Friction | 0.00E+00 | 3.00E-01 | 3.00E-01 | 3.00E-01 | 3.00E-01 | 3.00E-01 | 3.00E-01 | 3.00E-01 | 4.00E-01 | 3.00E-01 | 3.00E-01 | 3.00E-01 | 3.00E-01 | 3.00E-01 | 3.00E-01 | 3.00E-01 | 3.00E-01 |
| Initial Angle of Eccentricity (deg) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Amount of Eccentricity (inches) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Initial Velocity (fpm) * | 5.98E-07 | 6.00E-07 | 6.00E-07 | 6.00E-07 | 6.00E-07 | 6.00E-07 | 6.00E-07 | 6.00E-07 | 6.00E-07 | 6.00E-07 | 6.00E-07 | 6.00E-07 | 6.00E-07 | 6.00E-07 | 6.00E-07 | 6.00E-07 | 6.00E-07 |
| Initial Current (Amp) * | 3.00E-01 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 5.64E-12 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Initial External Force—Torque (lbf)—(lbf-in) | 0.00E+00 | 0.00E+00 | 3.00E+01 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Roller RMS Roughness (micro-in) | 0.00E+00 | 3.90E+01 | 4.00E+01 | 3.90E+01 | 3.90E+01 | 3.90E+01 | 3.90E+01 | 4.00E+01 | 4.00E+01 | 3.90E+01 | 3.90E+01 | 3.90E+01 | 3.90E+01 | 3.90E+01 | 3.90E+01 | 3.90E+01 | 3.90E+01 |
| Angle of Wrap (degrees) * | 0.00E+00 | 8.51E+00 | 1.14E+02 | 2.45E+01 | 8.63E+01 | 9.00E+01 | 8.69E+01 | 1.77E+02 | 1.87E+02 | 1.12E+02 | 5.80E+01 | 1.29E+02 | 9.51E+01 | 9.31E+01 | 1.16E+01 | 3.97E+00 | 6.73E+00 |

Computed from Analysis.

Dancer Damping Coefficient:
1. (lbf-sec/ft) If Translational
2. (lbf-ft-sec/rad) If Pivoting

Initial External Force/Torque:
1. (lbf) If Load Cell
2. (lbf) If Translational
3. (lbf-ft) If Pivoting

| | Span 1 | Span 2 | Span 3 | Span 4 | Span 5 | Span 6 | Span 7 | Span 8 | Span 9 | Span 10 | Span 11 | Span 12 | Span 13 | Span 14 | Span 15 | Span 16 | Span 17 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Youngs Modulus (kpsi) | 2.75E+02 | 2.75E+02 | 2.75E+02 | 2.75E+02 | 2.75E+02 | 2.75E+02 | 2.75E+02 | 2.75E+02 | 2.75E+02 | 2.75E+02 | 2.75E+02 | 2.75E+02 | 2.75E+02 | 2.75E+02 | 2.75E+02 | 2.75E+02 | 2.75E+02 |
| Web Thickness (mil) | 1.00E+00 | 1.00E+00 | 1.00E+00 | 1.00E+00 | 1.00E+00 | 1.00E+00 | 1.00E+00 | 1.00E+00 | 1.00E+00 | 1.00E+00 | 1.00E+00 | 1.00E+00 | 1.00E+00 | 1.00E+00 | 1.00E+00 | 1.00E+00 | 1.00E+00 |
| Web Width (inches) | 2.40E+01 | 2.40E+01 | 2.40E+01 | 2.40E+01 | 2.40E+01 | 2.40E+01 | 2.40E+01 | 2.40E+01 | 2.40E+01 | 2.40E+01 | 2.40E+01 | 2.40E+01 | 2.40E+01 | 2.40E+01 | 2.40E+01 | 2.40E+01 | 2.40E+01 |
| Span Length (inches) | 3.41E+01 | 1.93E+01 | 2.21E+01 | 2.31E+01 | 3.40E+01 | 3.60E+01 | 3.41E+01 | 9.01E+00 | 3.15E+01 | 7.21E+00 | 7.21E+00 | 2.37E+01 | 5.10E+01 | 2.97E+01 | 5.79E+01 | 3.40E+01 | 3.38E+01 |
| Web Density, lbm/(in^3) | 3.30E-02 | 3.30E-02 | 3.30E-02 | 3.30E-02 | 3.30E-02 | 3.30E-02 | 3.30E-02 | 3.30E-02 | 3.30E-02 | 3.30E-02 | 3.30E-02 | 3.30E-02 | 3.30E-02 | 3.30E-02 | 3.30E-02 | 3.30E-02 | 3.30E-02 |
| Initial Web Tension (lbf) * | 3.00E+01 | 3.00E+01 | 3.00E+01 | 3.00E+01 | 3.00E+01 | 3.00E+01 | 3.00E+01 | 3.00E+01 | 3.00E+01 | 3.00E+01 | 3.00E+01 | 3.00E+01 | 3.00E+01 | 3.00E+01 | 3.00E+01 | 3.00E+01 | 3.00E+01 |
| Web RMS Roughness (micro-in) | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 |
| Permeability (in^3/s)/(in^2-psi) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |

Computed from Analysis.

| | Idl Roll 18 | Idl Roll 19 | Idl Roll 20 | Idl Roll 21 | Idl Roll 22 | Idl Roll 23 | Idl Roll 24 | Idl Roll 25 | Idl Roll 26 | Idl Roll 27 | Idl Roll 28 | Wnd Roll 29 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| X Position (feet) | 9.15E+00 | 6.31E+00 | 5.97E+00 | 5.40E+00 | 5.17E+00 | 4.83E+00 | 2.00E+00 | 2.00E+00 | 3.73E+00 | 3.32E+00 | 5.23E+00 | 7.98E+00 |
| Y Position (feet) | 9.50E+00 | 9.50E+00 | 7.13E+00 | 7.46E+00 | 5.36E+00 | 7.88E+00 | 7.88E+00 | 5.13E+00 | 4.75E+00 | 4.14E+00 | 3.72E+00 | 3.00E+00 |
| Roller Diameter (inches) | 4.00E+00 | 4.00E+00 | 4.00E+00 | 4.00E+00 | 4.00E+00 | 4.00E+00 | 4.00E+00 | 4.00E+00 | 4.00E+00 | 4.00E+00 | 4.00E+00 | 8.00E+00 |
| Core Diameter (inches) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 4.00E+00 |
| Nip Roller Diameter (inches) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Nip Mass (lbm) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Nip Bearing Friction (lbf-ft-sec) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Nip Inertia (lbf-ft-sec^2) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Nip Roller Inertia (ft-lbf-s^2) * | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Roller Mass (lbm) * | 1.10E+01 | 1.10E+01 | 1.10E+01 | 1.10E+01 | 1.10E+01 | 1.10E+01 | 1.10E+01 | 1.10E+01 | 1.10E+01 | 1.10E+01 | 1.10E+01 | 1.10E+02 |
| Nip/Dancer Arm Placement (degrees) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Length of Dancer Arm (inches) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Roller Inertia (ft-lbf-s^2) * | 8.39E-03 | 8.39E-03 | 8.39E-03 | 8.39E-03 | 8.39E-03 | 8.50E-03 | 8.50E-03 | 8.39E-03 | 8.39E-03 | 8.39E-03 | 8.39E-03 | 9.26E-02 |
| Mass of Core (lbm) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 8.00E+01 |
| Roller Core Inertia (lbf-ft-sec^2) | 8.39E-03 | 8.39E-03 | 8.39E-03 | 8.39E-03 | 8.39E-03 | 8.39E-03 | 8.39E-03 | 8.39E-03 | 8.39E-03 | 8.39E-03 | 8.39E-03 | 5.39E-02 |
| Dancer Drag (lbf-sec/ft)—(lbf-ft-sec/rad) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Spring Constant (lbf/ft) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Motor Constant (lbf-ft/amp) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 1.00E+02 |
| Torque/Speed Ratio (lbf-ft-sec) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Bearing Friction (lbf-ft-sec) | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 |
| Static Coefficient of Friction | 3.00E-01 | 3.00E-01 | 3.00E-01 | 3.00E-01 | 3.00E-01 | 3.00E-01 | 3.00E-01 | 3.00E-01 | 3.00E-01 | 3.00E-01 | 3.00E-01 | 0.00E+00 |
| Initial Angle of Eccentricity (deg) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Amount of Eccentricity (inches) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Initial Velocity (fpm) * | 6.00E-07 | 6.00E-07 | 6.00E-07 | 6.00E-07 | 6.00E-07 | 6.00E-07 | 6.00E-07 | 6.00E-07 | 6.00E-07 | 6.00E-07 | 6.00E-07 | 6.00E-07 |
| Initial Current (Amp) * | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 1.00E-01 |
| Initial External Force—Torque (lbf)—(lbf-in) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 3.00E+01 | 0.00E+00 | 0.00E+00 |
| Roller RMS Roughness (micro-in) | 3.90E+01 | 3.90E+01 | 3.90E+01 | 3.90E+01 | 3.90E+01 | 3.90E+01 | 3.90E+01 | 3.90E+01 | 3.90E+01 | 4.00E+01 | 3.90E+01 | 0.00E+00 |
| Angle of Wrap (degrees) * | 1.35E+01 | 9.66E+01 | 1.50E+02 | 1.53E+02 | 1.83E+02 | 9.00E+01 | 9.00E+01 | 8.86E+01 | 1.50E+02 | 1.49E+02 | 8.81E+00 | 0.00E+00 |

Computed from Analysis.

Dancer Damping Coefficient:
1. (lbf-sec/ft) If Translational
2. (lbf-ft-sec/rad) If Pivoting

Initial External Force/Torque:
1. (lbf) If Load Cell
2. (lbf) If Translational
3. (lbf-ft) If Pivoting

| | Span 18 | Span 19 | Span 20 | Span 21 | Span 22 | Span 23 | Span 24 | Span 25 | Span 26 | Span 27 | Span 28 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Youngs Modulus (kpsi) | 2.75E+02 | 2.75E+02 | 2.75E+02 | 2.75E+02 | 2.75E+02 | 2.75E+02 | 2.75E+02 | 2.75E+02 | 2.75E+02 | 2.75E+02 | 2.75E+02 |
| Web Thickness (mil) | 1.00E+00 | 1.00E+00 | 1.00E+00 | 1.00E+00 | 1.00E+00 | 1.00E+00 | 1.00E+00 | 1.00E+00 | 1.00E+00 | 1.00E+00 | 1.00E+00 |
| Web Width (inches) | 2.40E+01 | 2.40E+01 | 2.40E+01 | 2.40E+01 | 2.40E+01 | 2.40E+01 | 2.40E+01 | 2.40E+01 | 2.40E+01 | 2.40E+01 | 2.40E+01 |
| Span Length (inches) | 3.38E+01 | 2.85E+01 | 6.92E+00 | 2.50E+01 | 3.02E+01 | 3.40E+01 | 3.30E+01 | 2.09E+01 | 7.82E+00 | 2.31E+01 | 3.41E+01 |
| Web Density, lbm/(in^3) | 3.30E-02 | 3.30E-02 | 3.30E-02 | 3.30E-02 | 3.30E-02 | 3.30E-02 | 3.30E-02 | 3.30E-02 | 3.30E-02 | 3.30E-02 | 3.30E-02 |
| Initial Web Tension (lbf) * | 3.00E+01 | 3.00E+01 | 3.00E+01 | 3.00E+01 | 3.00E+01 | 3.00E+01 | 3.00E+01 | 3.00E+01 | 3.00E+01 | 3.00E+01 | 3.00E+01 |
| Web RMS Roughness (micro-in) | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 |
| Permeability (in^3/s)/(in^2-psi) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |

Computed from Analysis.

| | Unw Roll 1 | Idl Roll 2 | Idl Roll 3 | Dan Roll 4 | Idl Roll 5 | Idl Roll 6 | Idl Roll 7 | Idl Roll 8 | Idl Roll 9 | Drv Roll 10 | Drv Roll 11 | Idl Roll 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| X Position (feet) | 7.62E-01 | 7.62E-01 | 3.39E+00 | 3.02E+00 | 4.76E+00 | 2.71E+00 | 2.68E+00 | 5.16E+00 | 5.85E+00 | 5.60E+00 | 1.06E+01 | 7.08E+00 |
| Y Position (feet) | 1.80E+01 | 1.68E+01 | 1.72E+01 | 1.75E+01 | 1.76E+01 | 1.91E+01 | 1.91E+01 | 1.65E+01 | 1.91E+01 | 1.79E+01 | 1.79E+01 | 1.72E+01 |
| Roller Diameter (inches) | 1.40E+01 | 3.00E+00 | 3.00E+00 | 3.00E+00 | 3.00E+00 | 3.00E+00 | 3.00E+00 | 3.00E+00 | 3.00E+00 | 1.20E+01 | 1.20E+01 | 3.00E+00 |
| Core Diameter (inches) | 3.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Nip Roller Diameter (inches) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 4.00E+00 | 0.00E+00 | 0.00E+00 |
| Nip Mass (lbm) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 1.50E+01 | 0.00E+00 | 0.00E+00 |
| Nip Bearing Friction (lbf-ft-sec) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 2.00E-04 | 0.00E+00 | 0.00E+00 |
| Nip Inertia (lbf-ft-sec^2) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 1.14E-02 | 0.00E+00 | 0.00E+00 |
| Nip Roller Inertia (ft-lbf-s^2) * | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 1.14E-02 | 0.00E+00 | 0.00E+00 |
| Roller Mass (lbm) * | 3.74E+01 | 1.82E+01 | 1.82E+01 | 1.19E+00 | 8.76E-01 | 8.76E-01 | 1.82E+01 | 1.82E+01 | 1.82E+01 | 7.85E+01 | 7.85E+01 | 1.82E+01 |
| Nip/Dancer Arm Placement (degrees) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 9.00E+01 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 9.00E+01 | 0.00E+00 | 0.00E+00 |
| Length of Dancer Arm (inches) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 1.50E+01 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Roller Inertia (ft-lbf-s^2) * | 1.11E+00 | 4.64E-03 | 4.64E-03 | 5.47E-04 | 2.20E-04 | 2.20E-04 | 4.64E-03 | 4.64E-03 | 4.64E-03 | 9.24E+00 | 9.24E+00 | 4.64E-03 |
| Mass of Core (lbm) | 2.84E+01 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Roller Core Inertia (lbf-ft-sec^2) | 1.53E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 9.24E+00 | 9.24E+00 | 0.00E+00 |
| Dancer Drag (lbf-sec/ft)—(lbf-ft-sec/rad) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 1.50E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Spring Constant (lbf/ft) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Motor Constant (lbf-ft/amp) | 1.27E+01 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 6.25E+01 | 6.25E+01 | 0.00E+00 |
| Torque/Speed Ratio (lbf-ft-sec) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Bearing Friction (lbf-ft-sec) | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 |
| Static Coefficient of Friction | 0.00E+00 | 3.00E-01 | 3.00E-01 | 3.00E-01 | 3.00E-01 | 3.00E-01 | 3.00E-01 | 3.00E-01 | 3.00E-01 | 4.00E-01 | 4.00E-01 | 3.00E-01 |
| Initial Angle of Eccentricity (deg) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Amount of Eccentricity (inches) | 1.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 1.00E-01 | 0.00E+00 | 0.00E+00 |
| Initial Velocity (fpm) * | 4.00E+02 | 4.00E+02 | 4.00E+02 | 4.00E+02 | 4.00E+02 | 4.00E+02 | 4.00E+02 | 4.00E+02 | 4.00E+02 | 4.00E+02 | 4.00E+02 | 4.00E+02 |
| Initial Current (Amp) * | 6.93E-02 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 3.01E-05 | 2.44E-04 | 0.00E+00 |
| Initial External Force—Torque (lbf)—(lbf-in) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 3.00E+02 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 1.52E+02 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Roller RMS Roughness (micro-in) | 0.00E+00 | 4.00E+01 | 4.00E+01 | 4.00E+01 | 4.00E+01 | 4.00E+01 | 4.00E+01 | 4.00E+01 | 4.00E+01 | 4.00E+01 | 4.00E+01 | 4.00E+01 |
| Angle of Wrap (degrees) * | 0.00E+00 | 1.96E+02 | 1.57E+02 | 1.83E+02 | 9.23E+01 | 9.10E+01 | 8.10E+01 | 1.71E+02 | 9.68E+01 | 1.89E+02 | 1.92E+02 | 4.06E+01 |

Computed from Analysis.

Dancer Damping Coefficient:
1. (lbf-sec/ft) If Translational
2. (lbf-ft-sec/rad) If Pivoting

Initial External Force/Torque:
1. (lbf) If Load Cell
2. (lbf) If Translational
3. (lbf-ft) If Pivoting Density of Tyvek: 1.073698E-0002

| | Span 1 | Span 2 | Span 3 | Span 4 | Span 5 | Span 6 | Span 7 | Span 8 | Span 9 | Span 10 | Span 11 | Span 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Youngs Modulus (kpsi) | 4.63E+01 | 4.63E+01 | 4.63E+01 | 4.63E+01 | 4.63E+01 | 4.63E+01 | 4.63E+01 | 4.63E+01 | 4.63E+01 | 4.63E+01 | 4.63E+01 | 4.63E+01 |
| Web Thickness (mil) | 5.36E+00 | 5.36E+00 | 5.36E+00 | 5.36E+00 | 5.36E+00 | 5.36E+00 | 5.36E+00 | 5.36E+00 | 5.36E+00 | 5.36E+00 | 5.36E+00 | 5.36E+00 |
| Web Width (inches) | 6.08E+00 | 6.08E+00 | 6.08E+00 | 6.08E+00 | 6.08E+00 | 6.08E+00 | 6.08E+00 | 6.08E+00 | 6.08E+00 | 6.08E+00 | 6.08E+00 | 6.08E+00 |
| Span Length (inches) | 3.41E+01 | 6.00E+00 | 2.08E+01 | 2.45E+01 | 1.77E+01 | 3.94E+01 | 3.15E+01 | 3.16E+01 | 7.34E+01 | 3.99E+01 | 6.60E+01 | 4.35E+01 |
| Web Density, lbm/(in^3) | 1.14E-02 | 1.14E-02 | 1.14E-02 | 1.14E-02 | 1.14E-02 | 1.14E-02 | 1.14E-02 | 1.14E-02 | 1.14E-02 | 1.14E-02 | 1.14E-02 | 1.14E-02 |
| Initial Web Tension (lbf) * | 9.80E+00 | 9.88E+00 | 9.97E+00 | 1.01E+01 | 1.01E+01 | 1.02E+01 | 1.03E+01 | 1.04E+01 | 1.05E+01 | 1.05E+01 | 1.01E+01 | 1.01E+01 |
| Web RMS Roughness (micro-in) | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 |
| Permeability (in^3/s)/(in^2-psi) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |

Computed from Analysis.

| | Idl Roll 13 | Idl Roll 14 | Idl Roll 15 | Idl Roll 16 | Drv Roll 17 | Idl Roll 18 | Idl Roll 19 | Idl Roll 20 | Idl Roll 21 | Idl Roll 22 | Idl Roll 23 | Idl Roll 24 | Wnd Roll 25 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| X Position (feet) 7.6222898E-0001 | 1.80E+01 | 1.82E+01 | 1.85E+01 | 1.91E+01 | 2.18E+01 | 2.41E+01 | 2.41E+01 | 2.53E+01 | 2.56E+01 | 2.58E+01 | 2.76E+01 | 2.77E+01 | 2.95E+01 |
| Y Position (feet) | 1.92E+01 | 1.81E+01 | 1.92E+01 | 1.80E+01 | 1.96E+01 | 1.67E+01 | 1.93E+01 | 1.84E+01 | 1.96E+01 | 1.84E+01 | 1.96E+01 | 1.68E+01 | 1.84E+01 |
| Roller Diameter (inches) | 3.00E+00 | 3.00E+00 | 3.00E+00 | 3.00E+00 | 6.00E+00 | 3.00E+00 | 3.00E+00 | 3.00E+00 | 3.00E+00 | 3.00E+00 | 3.00E+00 | 3.00E+00 | 1.00E+01 |
| Core Diameter (inches) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 3.00E+00 |
| Nip Roller Diameter (inches) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 4.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Nip Mass (lbm) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 1.50E+01 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Nip Bearing Friction (lbf-ft-sec) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 2.00E-04 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Nip Inertia (lbf-ft-sec^2) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 1.14E-02 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Nip Roller Inertia (ft-lbf-s^2) * | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 1.14E-02 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Roller Mass (lbm) * | 1.82E+01 | 1.82E+01 | 1.82E+01 | 1.82E+01 | 4.25E+01 | 1.82E+01 | 1.82E+01 | 1.82E+01 | 1.82E+01 | 1.82E+01 | 1.82E+01 | 1.82E+01 | 3.32E+01 |
| Nip/Dancer Arm Placement (degrees) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 9.00E+01 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Length of Dancer Arm (inches) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Roller Inertia (ft-lbf-s^2) * | 4.64E-03 | 4.64E-03 | 4.64E-03 | 4.64E-03 | 2.47E+00 | 4.64E-03 | 4.64E-03 | 4.64E-03 | 4.64E-03 | 4.64E-03 | 4.64E-03 | 4.64E-03 | 1.11E+00 |
| Mass of Core (lbm) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 2.84E+01 |
| Roller Core Inertia (lbf-ft-sec^2) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 1.04E-01 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 1.53E+00 |
| Dancer Drag (lbf-sec/ft)—(lbf-ft-sec/rad) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Spring Constant (lbf/ft) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Motor Constant (lbf-ft/amp) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 3.09E+01 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 1.27E+01 |
| Torque/Speed Ratio (lbf-ft-sec) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Bearing Friction (lbf-ft-sec) | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 | 2.00E-04 |
| Static Coefficient of Friction | 3.00E-01 | 3.00E-01 | 3.00E-01 | 3.00E-01 | 4.00E-01 | 3.00E-01 | 3.00E-01 | 3.00E-01 | 3.00E-01 | 3.00E-01 | 3.00E-01 | 3.00E-01 | 0.00E+00 |
| Initial Angle of Eccentricity (deg) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Amount of Eccentricity (inches) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Initial Velocity (fpm) * | 4.00E+02 | 4.00E+02 | 4.00E+02 | 4.00E+02 | 4.00E+02 | 4.00E+02 | 4.00E+02 | 4.00E+02 | 4.00E+02 | 4.00E+02 | 4.00E+02 | 4.00E+02 | 4.00E+02 |
| Initial Current (Amp) * | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 1.69E-03 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 5.30E-02 |
| Initial External Force—Torque (lbf)—(lbf-in) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 1.00E+01 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| Roller RMS Roughness (micro-in) | 4.00E+01 | 4.00E+01 | 4.00E+01 | 4.00E+01 | 4.00E+01 | 4.00E+01 | 4.00E+01 | 4.00E+01 | 4.00E+01 | 4.00E+01 | 4.00E+01 | 4.00E+01 | 0.00E+00 |
| Angle of Wrap (degrees) * | 1.28E+02 | 1.77E+02 | 1.62E+02 | 1.12E+02 | 9.54E+01 | 1.53E+02 | 1.41E+02 | 1.34E+02 | 1.80E+02 | 1.35E+02 | 1.35E+02 | 1.48E+02 | 0.00E+00 |

Computed from Analysis.

Dancer Damping Coefficient:
1. (lbf-sec/ft) If Translational
2. (lbf-ft-sec/rad) If Pivoting

Initial External Force/Torque:
1. (lbf) If Load Cell
2. (lbf) If Translational
3. (lbf-ft) If Pivoting Density of Tyvek: 1.073698E-0002

| | Span 13 | Span 14 | Span 15 | Span 16 | Span 17 | Span 18 | Span 19 | Span 20 | Span 21 | Span 22 | Span 23 | Span 24 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Youngs Modulus (kpsi) | 4.63E+01 | 4.63E+01 | 4.63E+01 | 4.63E+01 | 4.63E+01 | 4.63E+01 | 4.63E+01 | 4.63E+01 | 4.63E+01 | 4.63E+01 | 4.63E+01 | 4.63E+01 |
| Web Thickness (mil) | 5.36E+00 | 5.36E+00 | 5.36E+00 | 5.36E+00 | 5.36E+00 | 5.36E+00 | 5.36E+00 | 5.36E+00 | 5.36E+00 | 5.36E+00 | 5.36E+00 | 5.36E+00 |
| Web Width (inches) | 6.08E+00 | 6.08E+00 | 6.08E+00 | 6.08E+00 | 6.08E+00 | 6.08E+00 | 6.08E+00 | 6.08E+00 | 6.08E+00 | 6.08E+00 | 6.08E+00 | 6.08E+00 |
| Span Length (inches) | 1.38E+01 | 1.41E+01 | 1.55E+01 | 3.75E+01 | 4.45E+01 | 3.11E+01 | 1.82E+01 | 1.52E+01 | 1.51E+01 | 2.56E+01 | 3.43E+01 | 2.85E+01 |
| Web Density, lbm/(in^3) | 1.14E-02 | 1.14E-02 | 1.14E-02 | 1.14E-02 | 1.14E-02 | 1.14E-02 | 1.14E-02 | 1.14E-02 | 1.14E-02 | 1.14E-02 | 1.14E-02 | 1.14E-02 |
| Initial Web Tension (lbf) * | 1.02E+01 | 1.03E+01 | 1.04E+01 | 1.05E+01 | 9.87E+00 | 9.96E+00 | 1.00E+01 | 1.01E+01 | 1.02E+01 | 1.03E+01 | 1.04E+01 | 1.05E+01 |
| Web RMS Roughness (micro-in) | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 | 1.50E+01 |
| Permeability (in^3/s)/(in^2-psi) | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |

Computed from Analysis.

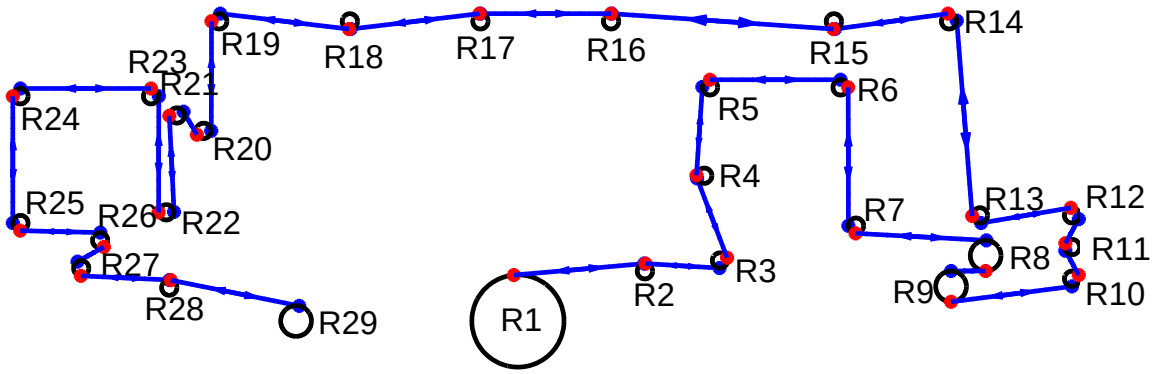



Figure I.3: The High-Speed Web Line Schematic Diagram Produced with the `PrintLine` Function.

Listing I.8: Element Property Read-In Code

```
function [Rolls, Spans, Colnames] = ElementPropReadIn(dat,rolls,spans)
% ElementPropReadIn is a function which returns the Roller and span
% elements properties from a WTS V2.1 element property export file.
% By Ben Reish (c) 2016
%
% Syntax: [Rolls, Spans, Colnames] = ElementPropReadIn(dat,rolls,spans)
%   where
%       dat:    name (and path to) export data file
%       rolls:  number of rollers in system
%       spans:  number of spans in system
%       Rolls:  matrix with roller element properties.  The order is given
%               below.
%       Spans:  matrix with span element properties.  The order is given
%               below.
%
FID = fopen(dat,'r');
Colnames = textscan(FID,'%8c  %d',rolls);
Rolls = textscan(FID,['%*47c',repmat('%f',[1,rolls])],28,'CollectOutput',...
    1,'BufSize',8191);
Spans = textscan(FID,['%*47c',repmat('%f',[1,spans])],8,'HeaderLines',...
    16,'CollectOutput',1);
fclose(FID);
```

Listing I.9: Span Data Calculation Code

```
function [ data,theta_in,theta_on ] = SpanData(X,Y,R,rot,theta_arm)
%returns a matrix of data (rows for each span) of span length, left-hand-
% endpoint, right-hand-endpoint, angle from horizontal to direction of span
% from left-hand-endpoint, angle from horizontal to direction of span from
% right-hand-endpoint.
%   Inputs:
%       X:      Cartesian x coordinate of center of roller (ft)
%       Y:      Cartesian y coordinate of center of roller (ft)
%       R:      Radius (ft) of roller
%       rot:    rotation of roller, 1 CCW, -1 CW
```

```
%
%   Output:
%       data:   [span length, left-hand- endpoint, right-hand-endpoint,
%                   angle from horizontal to direction of span from left-
%                   hand-endpoint, angle from horizontal to direction of
%                   span from right-hand-endpoint]
%       theta_in:   The angle from the direction of the theta_arm vector
%                   for each roller to the incoming span connection point
%       theta_on:   The angle from the direction of the theta_arm vector
%                   for each roller to the outgoing span connection point
%
n=size(X);
if max(n)>1
    %vector
    c = [X Y];
    d1 = zeros(max(n),1);
    data = zeros(max(n)-1,7);
    for i = 1:max(n)-1
        d1(i) = sqrt((c(i+1,:)-c(i,:))*(c(i+1,:)-c(i,:))');
        c1 = c(i,:);
        c2 = c(i+1,:);
            r1=R(i);
            r2=R(i+1);
        if rot(i)*rot(i+1)>0
            %same directions of rotation
            gamma = atan2(c2(2)-c1(2),c2(1)-c1(1));
            L = sqrt(d1(i)^2-(r1-r2)^2);
            if rot(i) == -1
                %if 1st roller CW
                theta = acos((r1-r2)/d1(i));
                phi = pi-theta;
                c3 = c1+R(i)*[cos(theta+gamma) sin(theta+gamma)];
                c4 = c2+R(i+1)*[cos(gamma+(pi-phi)) sin(gamma+(pi-phi))];
                d3 = [sin(theta+gamma) -cos(theta+gamma)];
                d4 = -[sin(gamma+(pi-phi)) -cos(gamma+(pi-phi))];
            else
                %1st roller CCW
                theta = acos((r1-r2)/d1(i));
                phi = pi-theta;
                c3 = c1-R(i)*[cos(theta+gamma) sin(theta+gamma)];
                c4 = c2-R(i+1)*[cos(gamma+(pi-phi)) sin(gamma+(pi-phi))];

                d3 = [sin(phi+gamma) -cos(phi+gamma)];
                d4 = -[sin(gamma+(phi)) -cos(gamma+(phi))];

            end
        else
            %opposite directions of rotation
            gamma = 2*pi+atan2(c2(2)-c1(2),c2(1)-c1(1));
```

```
60              if gamma > 2*pi
                    gamma = gamma - 2*pi;
62              end
                L = sqrt(d1(i)^2-(r1+r2)^2);
64              if rot(i) == -1
                    %if 1st roller CW
66                  theta = acos((r1+r2)/d1(i));
                    phi = pi/2-theta;
68                  if c2(2)>c1(2)
                        c3 = c1+R(i)*[cos(theta+gamma) sin(theta+gamma)];
70                  c4 = c2+R(i+1)*[cos(-pi+gamma+theta) sin(-pi+gamma+theta)];
                    d3 = [sin(theta+gamma) -cos(theta+gamma)];
72                  d4 = [sin(-pi+gamma+theta) -cos(-pi+gamma+theta)];
                    else
74
                    c3 = c1+R(i)*[cos(theta+gamma) sin(theta+gamma)];
76                  c4 = c2+R(i+1)*[cos(-pi+gamma+theta) sin(-pi+gamma+theta)];
                    d3 = [sin(theta+gamma) -cos(theta+gamma)];
78                  d4 = [sin(-pi+gamma+theta) -cos(-pi+gamma+theta)];
                    end
80              else
                    %1st rotating CCW
82                  theta = acos((r1+r2)/d1(i));
                    phi = pi/2-theta;
84                  c3 = c1-R(i)*[-cos(theta-gamma) sin(theta-gamma)];
                    c4 = c2+R(i+1)*[cos(gamma+pi-theta) sin(gamma+pi-theta)];
86                  d3 = [sin(theta-gamma) cos(theta-gamma)];
                    d4 = -[sin(pi-theta+gamma) -cos(pi-theta+gamma)];
88              end
            end
90          data(i,1:7) = [L c3(1) c3(2) c4(1) c4(2) atan2(d3(2),d3(1)) ...
                atan2(d4(2),d4(1))];
92      end
    else
94      disp('not enough data points in X -- SpanData')
    end
96      if nargin > 4
            theta_in = zeros(1,max(n));
98          theta_on = theta_in;
            ep2 = data(1,[2 3]);
100         varm = [cos(theta_arm(1)) sin(theta_arm(1))]*R(1)+[X(1) Y(1)];
            D = (2*R(1)^2-((ep2(1)-varm(1))^2+(ep2(2)-varm(2))^2))/2/R(1)^2;
102         theta_on(1) = atan2(sqrt(1-D^2),D);
            for i = 2:max(n)-1
104             ep1 = data(i-1,[4 5]);
                ep2 = data(i,[2 3]);
106             varm = [cos(theta_arm(i)) sin(theta_arm(i))]*R(i)+[X(i) Y(i)];
                D = (2*R(i)^2-((ep1(1)-varm(1))^2+(ep1(2)-varm(2))^2))/2/R(i)^2;
108             theta_in(i) = atan2(sqrt(1-D^2),D);
```

```
            D = (2*R(i)^2-((ep2(1)-varm(1))^2+(ep2(2)-varm(2))^2))/2/R(i)^2;
110             theta_on(i) = atan2(sqrt(1-D^2),D);
        end
112         ep1 = data(max(n)-1,[4 5]);
        varm = [cos(theta_arm(max(n))) sin(theta_arm(max(n)))]*R(max(n))...
114             +[X(max(n)) Y(max(n))];
        D = (2*R(max(n))^2-((ep1(1)-varm(1))^2+(ep1(2)-varm(2))^2))/2/...
116             R(max(n))^2;
        theta_in(max(n)) = atan2(sqrt(1-D^2),D);
118     end
    function x = Intersects(n)
120         ep1 = data(n-1,[4 5]); ep2 = data(n,[2 3]); t3 = data(n-1,6);
        t4 = data(n,7);
122         t = sin(t3)/(sin(t3)*cos(t4)-sin(t4)*cos(t3))*((ep2(2)-ep1(2))...
            /sin(t3)*cos(t3)+ep1(1)-ep2(1));
124         if t>=0
            s = (sin(t4)*t+ep2(2)-ep1(2))/sin(t3);
126             if s>=0
                x = 1;
128             else
                x = 0;
130             end
        else
132             x = 0; disp('t had to go backwards -- SpanData')
        end
134     end
end
```

Listing I.10: Wrap Angle Calculation Code

```
function [ wraps ] = WrapAngle(SData, Rn )
%WrapAngle calculates the angle of wrap of a web over a roller given the
%output of the SpanData function and the radii.  Created by Ben Reish (c)
%2018
%   Inputs:
%       SData:      The matrix of data output from the SpanData function
%       Rn:         The vector of radii of the rollers
%
m = size(Rn);
spans = max(m)-1;
wraps = zeros(m);
for i=2:spans
    D = (2*Rn(i)^2-((SData(i-1,4)-SData(i,2))^2+(SData(i-1,5)-...
        SData(i,3))^2))/2/Rn(i)^2;
    wraps(i) = atan2(sqrt(1-D^2),D);
    D3s = (SData(i-1,5)+sin(SData(i-1,6))*((SData(i,2)-...
        SData(i-1,4))/cos(SData(i-1,6))) - SData(i,3))/sin(SData(i,7))/(1-...
        (sin(SData(i-1,6))*cos(SData(i,7)))/sin(SData(i,7))/...
        cos(SData(i-1,6)));
    D3t = (SData(i,2) + cos(SData(i,7))*D3s - SData(i-1,4))/...
        cos(SData(i-1,6));
    if D3s < 0 || D3t < 0
        wraps(i) = 2*pi - wraps(i);
    end
end
end
```

Listing I.11: Print Line Code

```
function [sdata,thtain,thtaout] = PrintLine(file,rolls,rot,fig,theta_arm)
%Creates a figure that shows the the layout of the web line
%   Inputs:
%       file:   file name to element data file
%       rolls:  number of roller in the data file
%       rot:    row vector of -1 (CW) or 1 (CCW) for each roller
%       theta_arm:  angle from the horizontal to the pendulum arm (defaults
%                   to 0)
%   Outputs:
%       Figure of the plotted webline
%       sdata:  Matrix from SpanData function
%       thtain: angle between theta_arm direction and incoming span
%       connection point
%       thtaout:    angle between theta_arm direction and outgoing span
%       connection point
[Rolls, Spans, Colnames] = ElementPropReadIn(file,rolls,rolls-1);
Rn = Rolls{1}(3,:)/24;
C = [Rolls{1}(1,:)' Rolls{1}(2,:)'];
if nargin<4
    fig = 1; theta_arm = zeros(1,rolls);
elseif nargin<5
    theta_arm = zeros(1,rolls);
end
[data,thtain,thtaout] = SpanData(Rolls{1}(1,:)',Rolls{1}(2,:)',...
    Rolls{1}(3,:)'/24,rot',theta_arm);
i=[0:360]/180*pi;  circle =[cos(i') sin(i')];
h=figure(fig); clf
hold on
for i = 1:rolls-1
    plot(Rn(i)*circle(:,1)+C(i,1),Rn(i)*circle(:,2)+C(i,2),'k',..
        data(i,[2 4]),data(i,[3 5]),'b:',data(i,2),data(i,3),'r*',...
        data(i,4),data(i,5),'b*')
    text(C(i,1),C(i,2),['R' num2str(i)],'fontsize',7)
    quiver(data(i,[2 4]),data(i,[3 5]),[cos(data(i,6)) ...
        cos(data(i,7))],[sin(data(i,6)) sin(data(i,7))])
end
plot(Rn(i+1)*circle(:,1)+C(i+1,1),Rn(i+1)*circle(:,2)+C(i+1,2),'k')
text(C(i+1,1),C(i+1,2),['R' num2str(i+1)],'fontsize',7)
hold off
title('Web Line Visualization','fontsize',16); axis equal
h1=get(h,'Children');
set(h1,'Visible','Off','Position',[0.05 .05 .9 .8])
set(get(h1,'title'),'Visible','On')
if nargout > 0
    sdata = data;
end
end
```

## I.6 Recording Events

Most of the integration in this dissertation was accomplished with the MATLAB function `ODE45` which has a variable time step. Partly due to the problem discussed in section H.6 and partly due to needing to know when rollers slipped, a data recording function was created that persists until MATLAB is closed or the function is cleared from memory called `RecordEvents` (see Listing I.12). Since `ODE45` can back up and re-iterate over a time period with a smaller time step to meet integration tolerances, the `RecordEvents` function will compare the current time with the stored times it has in memory and if it is smaller or equal to one already in memory, the previously stored data point it thrown out and the new one is stored. The `RecordEvents` function needs to be initialized before being used. Once resident in memory, calls to the function will include the current time, the state vector of the system, and the reason vector. After the integration is finished, the `RecordEvents` function can be asked to produce a cell array containing the times of events, the states of the system at those times, and the reason vectors that caused the events. The user can then search the reason vector for rows that correspond to the reason of interest. Once that is known, the time of the event and the state of the system may be investigated.

Listing I.12: Record Events Code

```
function [ Event_struct ] = RecordEvents(t,y,indices,initialize )
%Created by Ben Reish (c) 2018
%   The RecordEvents function takes data and compares it with existing data
%   to see if an event at that time already exists.  If so it compares the
%   indices to see if it is the same event.  If so it discards it; if not,
%   it adds the event to a history stack.  The stack has to be initialized
%   so that the function can repeatedly work with it.
%
%   Syntax:
%       t:          calling time (when the function is called
%       y:          vector to be stored (data)
%       indices:    vector of -1,0,1 's to indicate what triggered the
%                       event (this is not limited to ones and zeros, but
%                       that's what is used initially
%       initialize: a value of 1 initializes the function, 0 or not
%                       included lets the function run normally, 2 returns
%                       the Event_struct to the calling function.  The
%                       function must be called first with a 1, then called
%                       one or more times with a zero, and then a value of
%                       2 may be used.
%       Event_struct:   cell array of a (1) vector of times data was
%                       recorded based on t, (2) matrix of stacked y
%                       vectors, (3) matrix of stacked indices vectors
persistent times retained_data retained_indices
if nargin < 4
    initialize = 0;
end
n = size(y,2);
if n == 1
    y = y';
end
n = size(indices,2);
if n == 1
    indices = indices';
end
if initialize == 0
    m = find(abs(times - t) <= 0.004,1);
    if isempty(m)
        %time t not previously stored, store it
        times = [times t];
        retained_data = [retained_data; y];
        retained_indices = [retained_indices; indices];
    else
        if size(retained_indices(end,:),2)~= n
            disp([t,n])
        end
```

Listing I.13: Record Events Code (cont.)

```
            test = retained_indices(end,:)-indices;
            if sum(test==0) == numel(indices)
                %previously stored
            else
                times = [times t];
            retained_data = [retained_data; y];
            retained_indices = [retained_indices; indices];
            end
        end
    elseif initialize == 1
        times = [];
        retained_data = [];
        retained_indices = [];
    elseif initialize == 2
        Event_struct = {times,retained_data,retained_indices};
    else
        error('RecordEvents:InvalidInitialize',...
            'Value of initialize input is not 0,1,2')
    end
end
```

## I.7 The Runge-Kutta 4 Solver

When simulating with the SFDR model, a fixed-time-step integrator was required for the method used to check for slip and switch to the SFDR model. The the Runge-Kutta 4 method came from [86] but that is not the only integration method available. Through the `type` variable, the function can be told to use Eulerian, Huen, Simpson, Runge-Kutta 4, or Runge-Kutta 5 methods. A set of options has to be set for the function to call a slip check function at the end of each integration step. The slip check function name is given to `odeBR4` as an option and it is user defined. It is suggested that the Whitworth criteria, (5.2) and (5.3) be used.

Listing I.14: Runge-Kutta 4 Solver Code

```
function [time,z] = odeBR4(fname,timelimits,x0,options)
%   Created by Ben Reish (c) 2018
%   odeBR4 is a 4th order Runge-Kutta method with fixed time step.
%
%   odeBR4 can take options.  The options variable is a cell array of 2
%   columns and multiple rows.  If you use the options variable, you have
%   to assign 'TimeStep' a value; otherwise, the function will fail.  The
%   other options are 'SlipTrue' which is boolean 1 for true, 0 for false.
%   'EventFunction' takes an argument in the 2nd column that is the name of
```

```
%   the function you want called at the end of the integration step.  Both
%   'SlipTrue' and 'EventFunction' have to be assigned for the event
%   function to be processed.  If you want to turn off the event function
%   evaluation, set 'SlipTrue' to 0.
%
%   Though not a function parameter (it could be made an option; I just
%   didn't), 'type' can take values from 1 to 5 in order to make the
%   integrator use Eulerian (1), Huen (2), Simpson's rule (3), RK4 (4,
%   default), or RK5 (5) methods.
%
%
type = 4; %1: Euler, 2: Huen, 3: simpson's rule, 4: RK4, 5: RK5
if nargin > 3
    m=size(options);
    for i = 1:max(m)
        switch options{i,1}
            case 'TimeStep'
                dt = options{i,2};
            case 'SlipTrue'
                slip_true = 1;
            case 'EventFunction'
                eventfcn = options{i,2};
            otherwise
                warning('Unused option encountered')
        end
    end
else
    dt = 0.005; %default time step
    slip_true = 0; %default to no slip consideration
end
timecheck = rem(timelimits*[-1 1]',dt);
odd_lot = 0; %boolean for noting a nonuniform timeseries
if timecheck == 0
    timeseries = timelimits(1):dt:timelimits(2);
else
    timeseries = [timelimits(1):dt:timelimits(2)-mod(timelimits(2),dt),...
        timelimits(2)];
    odd_lot = 1;
end
neqs = max(size(x0));
yout = zeros(numel(timeseries),neqs);
t=timeseries(1);
x = x0;
for i = 1:numel(timeseries)
    if i == numel(timeseries) && odd_lot
        dt = timeseries(end)-timeseries(i-1);
    end
    if type == 4
    k1 = dt*feval(fname,t,x0);
```

```
        k2 = dt*feval(fname,t+dt/2,x0+k1/2);
60      k3 = dt*feval(fname,t+dt/2,x0+k2/2);
        k4 = dt*feval(fname,t+dt,x0+k3);
62      elseif type == 5
            k1 = dt*feval(fname,t,x0);
64      k2 = dt*feval(fname,t+dt/4,x0+k1/4);
        k3 = dt*feval(fname,t+dt/4,x0+k1/8+k2/8);
66      k4 = dt*feval(fname,t+dt/2,x0-k2/2+k3);
        k5 = dt*feval(fname,t+3*dt/4,x0+3*k1/16+9*k4/16);
68      k6 = dt*feval(fname,t+dt,x0-3*k1/7+2*k2/7+12*k3/7-12*k4/7+8*k5/7);
        elseif type == 2
70          k1 = dt*feval(fname,t,x0);
            k2 = dt*feval(fname,t+dt,x0+k1);
72      elseif type ==1
            k1 = dt*feval(fname,t,x0);
74      else
            k1 = dt*feval(fname,t,x0);
76          k2 = dt*feval(fname,t+dt/2,x0+k1/2);
            k3 = dt*feval(fname,t+dt,x0-k1+2*k2);
78      end
        if type == 4
80      ynew = x0 + [k1 + 2*k2 + 2*k3 + k4]/6;
        elseif type == 5
82          ynew = x0 + [ 7*k1+32*k3+12*k4+32*k5+7*k6]/90;
        elseif type == 2
84          ynew = x0 + (k1/2 + k2/2);
        elseif type == 1
86          ynew =  x0 + k1;
        else
88          ynew = x0 + (k1 + 4*k2 + k3)/6;
        end
90      tnew = t+dt;

92      %check for slip if wanted
        if slip_true
94          [tnew,ynew] = feval(eventfcn,tnew,ynew);
        end
96      %ynew(1:7,1) = round(ynew(1:7,1)*3000)/3000;
        %save data
98      yout(i,:) = ynew';

100     %step forward
        t = tnew;
102     x0 = ynew;
    end
104 time = timeseries';
    z = yout;
106 end
```

## I.8 Modularization of the Code

Sometimes data is used together and specific properties are repeatedly required for simulations. An example of this is the motor-shaft/gearbox-parent roll of material. Gain calculations and state equations need to know the inertia of the parent roll-motor system. One function is needed that can keep track of the parent roll properties, the motor properties, and gearbox properties (if any). The following functions group primitive element properties together into logical conglomerations.

### I.8.1 The Parent Roll Class Definition

The parent roll is the beginning of a process in web handling. It has mass, stiffness, width, radius, core material and dimensions, and web thickness. The following code, `Parentroll`, creates a class object in MATLAB to contain those variables and calculate the roll's mass and inertia in a consistent manner.

The parameters in the `storedProps` method define parent rolls used in this dissertation. The cores and material properties are reported to the best available knowledge and experimentation.

Listing I.15: Parent Roll Class Definition Code

```
classdef Parentroll < handle
% Object to contain parent roll parameters
% Ben Reish (c) 2019
%
%
%  Syntax:  R1 = Parentroll(param);
%     where
%       param is either a 8 x 2 cell array (up to 10 x 2 if optional
%       parameters are used) of character strings and numbers or a single
%       character string that equals one of the predefined motor types in
%       this class.
%
%  The param cell array must contain the following character strings with
%  the name plate values.  The values are in the Imperial system by
%  default. Use the set('units',1) command to change to SI units.  The
%  following example param cell array has all the values:
%
%   param = {'width',6; 'thickness', .00536; 'density', 0.5764; ...
%            'core_rad_i',1.5;'core_rad_o', 2; 'core density' 43/32.2; ...
%            'E', 68e3*144; 'Roll R' 7; 'material', 'Tyvek-01';...
```

```
%           'core width', 10};
%   width : web width in inches
%   thickness : web thickness in inches
%   density : web density in slug/ft^3
%   core_rad_i : inner core radius in inches
%   core_rad_o : outer core radiut in inches
%   core density : core material density in slug/ft^3
%   E : Young's modulus of the web in lbf/ft^2
%   Roll R : name plate horsepower of the motor
%
%   Optional parameters:
%   material : string containing the name of the material specified;  set
%              by default when using one of the predefined webs
%   core width : width of core if different from web width.  If this is
%                left out, it is assumed equal to width (in)
%
%   Methods: (user level)
%     inertia = R01.processInertia(r);
%       processInertia method uses the new value of r (in feet) to
%       calculated the web hollow cylinder inertia and adds it to the core
%       inertia and that total is returned as inertia.
%     R01.get(mfield)
%       get method returns the requested value from inside the object in
%       base units.
%     R01.set(mfield,val)
%       set method sets individual parameters, mfield (contains a string),
%       to the value of val.  object assumed val is in base units for
%       parameter mfield.
%     R01.processUnits(val) ------Currently inoperative-000-Do not use------
%       processUnits method compares val to the current unit system boolean
%       and if they are different, changes the internal unit system to the
%       requested one: 0 = Imperial, 1 = SI.
%
%


    properties (Access = public)
        material; %  string naming the material
        J_roll; %  roll inertia (slug-ft^2)
        width; %  web width (ft)
        thickness; %  web thickness (ft)
        rho; % web density (slug/ft^3)
        core_rad_i; %  inner core of roll radius (ft)
        core_rad_o; %  outer core of roll radius (ft)
        core_width = 0; %  core width (if different than web width (ft)
        core_rho; %  density of core (slug/ft^3)
        E; %  Young's Modulus (lbf/ft^2)
        R_roll; %  roll raduis (ft) from inner core rad to outer edge
        units = 0; %  unit system boolean, 0-Imperial, 1-SI
```

```
70
        end
72      properties (Access = private)
            g = 32.174; %  acceleration of gravity in ft/s^2
74          core_in; % core inertia (slug-ft^2)
        end
76      methods
            function obj = Parentroll(params)
78              % Constructor for the class
                %
80  %  Inputs:
    %    params    - 7 x 2 cell array of motor data
82  %
    %  Outputs:
84  %    -none
    try
86  % nesting structure to set new values
    if nargin ==1
88    if ~isempty(params)
      if ischar(params)
90        obj.material = params;
          s = storedProps(obj,params);
92        params = s;
      end
94
        for i = 1:length(params)
96          switch (params{i,1})
                case 'width'
98                  obj.width = params{i,2}/12;
                case 'thickness'
100                 obj.thickness = params{i,2}/12;
                case 'density'
102                 obj.rho = params{i,2};
                case 'core_rad_i'
104                 obj.core_rad_i = params{i,2}/12;
                case 'core_rad_o'
106                 obj.core_rad_o = params{i,2}/12;
                case 'roll R'
108                 obj.R_roll = params{i,2}/12;
                case 'E'
110                 obj.E = params{i,2};
                case 'core density'
112                 obj.core_rho = params{i,2};
                case 'core width'
114                 obj.core_width = params{i,2}/12;
                case 'material'
116                 obj.material = params{i,2};
                otherwise
118                 exception = MException('VerifyInput:NotUsed', ...
```

```
                        ['Input ' params{i,1} ' is not a valid name']);
120                 throw(exception);
                end
122           end
            end
124         if obj.core_width == 0
                obj.core_width = obj.width;
126         end
            % calculated core volume and inertia--should not change, so do it
128         % once and be done.
            calculate_Core(obj); %calculate core inertia
130         calculate(obj); %calculate web roll inertia

132       else
            exception = MException('VerifyInput:NotEnoughInputs', ...
134             ['motor object requires 1 input argument.']);
            throw(exception);
136       end

138       catch ME
            disp([ME.message '!'])
140         err = MException('VerifyOutput:OutOfBounds', ...
                    'Exiting due to incorrect inputs.');
142         throw(err);
          end
144
          end
146           function calculate(obj)
                % method calculates roll inertia
148             web_v = obj.width*pi*(obj.R_roll^2 - ...
                    obj.core_rad_o^2);
150             web_inertia = web_v*obj.rho*(obj.R_roll^2 + ...
                    obj.core_rad_o^2)/2;
152             obj.J_roll = web_inertia + obj.core_in;
              end
154           function calculate_Core(obj)
                % method calculates core inertia.  This function allows
156             % updating of the core properties in an existing Parentroll
                % object.
158             core_v = obj.core_width*pi*(obj.core_rad_o^2 - ...
                   obj.core_rad_i^2);
160             obj.core_in = core_v*obj.core_rho*(obj.core_rad_i^2 + ...
                   obj.core_rad_o^2)/2;
162
              end
164           function inertia = processInertia(obj,R)
                % set new roll radius, call calculate, and return inertia R
166             % is input in feet
                obj.R_roll = R;
```

```
168                 calculate(obj);
                    inertia = obj.J_roll;
170             end
                function processUnits(obj)
172                 % switches between Imperial and SI
                    %
174                 err = MException('NotFinished:IncompleteCode', ...
                    ['The method processUnits is not finished and does',...
176                 'not work.']);
                    throw(err);
178             end
                function val = get(obj,mfield)
180                 %
                    switch mfield
182                     case 'J'
                            val = obj.J_roll;
184                     case 'R'
                            val = obj.R_roll;
186                     case 'EA'
                            val = obj.E*obj.width*obj.thickness;
188                     case 'E'
                            val = obj.E;
190                     case 'material'
                            val = obj.material;
192                     case 'units'
                            if obj.units == 1
194                             disp('SI system')
                            else
196                             disp('Imperial system')
                            end
198                     otherwise
                           exception = MException('VerifyInput:NotUsed', ...
200                            ['Input ' mfield ' is not a valid name']);
                           throw(exception);
202                 end
                end
204             function set(obj,mfield,val)
                    %
206                 switch mfield
                        case 'core density'
208                         obj.core_rho = val;
                            calculate_Core(obj);
210                     case 'R'
                            obj.R_roll = val;
212                         calculate(obj);
                        case 'core_rad_i'
214                         obj.core_rad_i = val;
                            calculate_Core(obj);
216                     case 'core_rad_o'
```

```
                    obj.core_rad_o = val;
                    calculate_Core(obj);
                case {'density','rho'}
                    obj.rho = val;
                    calculate(obj);
                case 'units'
                    if obj.units ~= val
                        %change unit system
                        if ismember(val,[0 1])
                            obj.units = val;
                            processUnits;
                        else
                            exception = ...
                                MException('VerifyInput:NotUsed', ...
                                ['Input ' num2str(val) ' is not ',...
                                'valid  for ' mfield]);
                            throw(exception);
                        end
                    end
                otherwise
                   exception = MException('VerifyInput:NotUsed', ...
                       ['Input ' mfield ' is not a valid name']);
                   throw(exception);
            end
        end
        function disp(obj)
            %method displays the internal parameters

            if obj.units
            s = {'Material:',obj.material,''; ...
                'E:', obj.E,'Pa';...
                'Radius:', obj.R_roll,'m';...
                'rho:',obj.rho,'kg/m^3';...
                'Width:',obj.width,'m'; ...
                'Thickness:',obj.thickness*1000,'mm';...
                'Core Inertia:',obj.core_in,'kg-m^2';...
                'Roll Inertia:',obj.J_roll,'kg-mhh^2'};
            else
            s = { ...
                'E:', obj.E/144,'psi';...
                'Radius:', obj.R_roll*12,'in';...
                'rho:',obj.rho,'slug/ft^3';...
                'Width:',obj.width,'ft'; ...
                'Thickness:',obj.thickness*12000,'mils';...
                'Core Inertia:',obj.core_in*144,'slug-in^2';...
                'Roll Inertia:',obj.J_roll*144,'slug-in^2'};
            end
            fprintf(1,'%s\t%s %s\n','Material:',obj.material,'');
            for i=1:7
```

```
266                 fprintf(1,'%s\t%.2f %s\n',s{i,1},s{i,2},s{i,3})
                end
268             disp('    ')
            end
270         function s = storedProps(obj,mfield)
                % outputs stored parameter sets for motor
272             switch (mfield)
                    case 'Tyvek-01'
274             s = {'width',6.077; ...%  web width (in)
                    'thickness', 0.00536;... %  web thickness (in)
276                 'core_rad_i',1.5; ...%  inner core of roll radius (in)
                    'core_rad_o', 2; ...%  outer core of roll radius (in)
278                 'core width', 10; ...%  core width (if different than
                    ... %                       web width (in)
280                 'core density', 43/obj.g; ...% core density (slug/ft^3)
                    'E', 68e3*144; ...%  Young's Modulus (lbf/ft^2)
282                 'density', 1.074E-2/obj.g*144*12 ;... % web density
                    'roll R', 7 };%roll radius, (in)
284             case 'PET-01'
                s = {'width',6; ...%  web width (in)
286                 'thickness', 0.002;... %  web thickness (in)
                    'core_rad_i',1.617; ...%  inner core of roll radius (in)
288                 'core_rad_o', 2; ...%  outer core of roll radius (in)
                    'core width', 10; ...%  core width (if different than
290                 ...%                       web width (in)
                    'core density', 13.645; ...%  density of core (slug/ft^3)
292                 'E', 625e3*144; ...%  Young's Modulus (lbf/ft^2)
                    'density', 2.697045 ;... % web density
294                 'roll R', 3.5 };%roll radius, (in)
                case 'PET-02'
296             s = {'width',6; ...%  web width (in)
                    'thickness', 0.002;... %  web thickness (in)
298                 'core_rad_i',1.617; ...%  inner core of roll radius (in)
                    'core_rad_o', 2; ...%  outer core of roll radius (in)
300                 'core width', 6; ...%  core width (if different than
                    ...%                       web width (in)
302                 'core density', 1.337; ...%  density of core (slug/ft^3)
                    'E', 625e3*144; ...%  Young's Modulus (lbf/ft^2)
304                 'density', 2.697045 ;... % web density
                    'roll R', 6.5 };%roll radius, (in)
306             case 'Tyvek-02'
                s = {'width',24.5; ...%  web width (in)
308                 'thickness', 0.005;... %  web thickness (in)
                    'core_rad_i',1.4772; ...%  inner core of roll radius (in)
310                 'core_rad_o', 1.8942; ...% outer core of roll radius (in)
                    'core width', 30.5; ...%  core width (if different than
312                 ...%                       web width (in)
                    'core density', 14.269; ...%  density of core
314                 ...%                        (slug/ft^3) steel
```

```
                    'E', 68e3*144; ...%  Young's Modulus (lbf/ft^2)
316                 'density', 1.074E-2/obj.g*144*12 ;... % web density
                    'roll R', 3.788 }; %roll radius, (in)
318             case 'Tyvek-03'
                s = {'width',24.5; ...%  web width (in)
320                 'thickness', 0.005;... %  web thickness (in)
                    'core_rad_i',1.551; ...%  inner core of roll radius (in)
322                 'core_rad_o', 2; ...%  outer core of roll radius (in)
                    'core width', 24.5; ...%  core width (if different than
324                 ...%                        web width (in)
                    'core density', 1.337; ...%  density of core
326                 ...%                        (slug/ft^3) cardboard
                    'E', 68e3*144; ...%  Young's Modulus (lbf/ft^2)
328                 'density', 1.074E-2/obj.g*144*12 ;... % web density
                    'roll R', 9 };%roll radius, (in)
330             case 'Tyvek-04' %narrow tyvek parent roll
                s = {'width',6.086; ...%  web width (in)
332                 'thickness', 0.005;... %  web thickness (in)
                    'core_rad_i',1.551; ...%  inner core of roll radius (in)
334                 'core_rad_o', 2; ...%  outer core of roll radius (in)
                    'core width', 6.086; ...%  core width (if different than
336                 ...%                        web width (in)
                    'core density', 1.337; ...%  density of core
338                 ...%                        (slug/ft^3) cardboard
                    'E', 68e3*144; ...%  Young's Modulus (lbf/ft^2)
340                 'density', 0.866;... % web density (slug/ft^3)
                    'roll R', 9 ;... %roll radius, (in)
342                 'material','Tyvek-04';... %material name
                    };
344             case 'Tyvek-05' %narrow tyvek rewind roll
                s = {'width',6.086; ...%  web width (in)
346                 'thickness', 0.005;... %  web thickness (in)
                    'core_rad_i',1.617; ...%  inner core of roll radius (in)
348                 'core_rad_o', 2; ...%  outer core of roll radius (in)
                    'core width', 10; ...%  core width (if different than
350                 ...%                        web width (in)
                    'core density', 13.645; ...%  density of core (slug/ft^3)
352                 'E', 68e3*144; ...%  Young's Modulus (lbf/ft^2)
                    'density', 0.866;... % web density (slug/ft^3)
354                 'roll R', 9 };%roll radius, (in)
            end
356     end
    end
358 end
```

### I.8.2 The Motor Class Definition

The motor is the point of force application in a web line and it is the point of control as well. It has inertia, mass, rated torque, and a top speed. This version of `motor` function here will instantiate a `Parentroll` object inside the `motor` object of use with inertia calculations if an optional argument is passed to the `motor` function's constructor.

Some parts of this function are taken from [61] in order to be able to model the gain calculation methods used on the HSWL.

Listing I.16: The Motor Class Definition Code

```
classdef motor < handle
% Object to contain motor parameters
% Ben Reish (c) 2019,2020
%
% Based on calculations in:
% Brian T. Boulter. "The Effect of Speed Loop Bandwidths and Line-speed on
% System Natural Frequencies in Multi-Span Strip Processing Systems." The
% Proc. of the 1997 IEEE Industry Applications Conference, Vol. 3, IEEE,
% pp. 2157-2164, 1997.
%
%  Syntax;  DM01 = motor(param);
%     where
%       param is either a 8 x 2 cell array of character strings and numbers
%       or a single character string that equals one of the predefined
%       motor types in this class.
%
%  The param cell array must contain the following character strings with
%  the name plate values.  The values are in the Imperial system by
%  default. Use the set('units',1) command to change to SI units.  The
%  following example param cell array has all the values:
%
%  params = {'J_m',7.1; 'base_speed', 1150; 'I_max', 48; 'torque' 137;'GR',
%            1; 'LS', 800; 'J_sec' 1.43; 'power', 30};
%   J_m : motor inertia in lbm-ft^2
%   base_speed : name plate speed in rpm
%   I_max : name plate amperage in amps
%   torque : name plate torque or horsepower * 33000/2/pi/rpm
%   GR : gear ratio between motor and shaft in (motor revolutions per shaft
%        revolutions)
%   LS : line speed in feet per minute
%   J_sec : (from Boulter paper) amount of time required for motor and
%           shaft to come up to rated speed at max torque
%   power : name plate horsepower of the motor
%   J_load_c : inertia of any connected shafts and gearboxes that do not
%              change in inertia -- use set('J_load',val) to set the load
```

```
36 %             inertia value to a non-zero val.  Remember the gear
   %             ratio will be applied to this value as J is calculated. If
38 %             the HSWL is being modeled, the set command can be modified
   %             to set('J_load','unwind') or set('J_load','rewind') to
40 %             include the calculated inertia of the spindles in use on
   %             that line. Since set is being used, the object has to be
42 %             initialized prior to the use of the set command.
   %   J : total inertia, J_motor + J_load_c in slug-ft^2
44 %
   %   Version 2:  written to contain the parent roll object within the motor
46 %       Thus the initalizer must also have a paramter for the Parentroll
   %       object to create with.  Params is now a 9x2 cell array and one has
48 %       to be 'material' and its second column is one of the strings from
   %       the parent roll object.
50 %       Jout = calculateInertia(obj,R,type)
   %           R - radius in ft
52 %           type - 0 calculates inertia from the motor perspective,
   %                  1 calculates inertia from the roll perspective
54 %


56         properties (Access = public)
               J_motor; %  motor inertia (slug-ft^2)
58             J_load_c = 0; %  shafts and gearboxes that are constant inertia
               J; %  total inertia from the motor perspective
60             omega_rated; %  rated motor speed (rad/s)
               torque_rated; %  rated motor torque (ft-lbf)
62             current_rated; %  motor maximum current at rated speed (amps)
               K_m; %  torque_rated/current_rated (ft-lbf)/amp
64             J_sec; %  seconds required for motor to come up to rated speed
                      % using full rated torque, motor inertia and any
66                    % constant additional inertias are included
               S; %  motor speed at application speed (rad/s)
68             GR; %  gear ratio (motor revolutions/ shaft revolution)
               LS; %  application speed in ft/min
70             omega_co; %  open loop cross-over frequency
               hp; %  horse power of the motor (hp)
72             units = 0; %  unit system boolean, 0-Imperial, 1-SI
               Roll; % Roll object if it exists
74             R; % Roll radius if Roll exists
           end
76         properties (Access = private)
               g = 32.174; %  acceleration of gravity in ft/s^2
78             RollTrue = 0; % boolean to indicate if an object is created
                             %in the Roll property
80         end
           methods
82             function obj = motor(params)
                   % Constructor for the class
84                 %
```

```
%  Inputs:
%    params    - 8 x 2 cell array of motor data
%                      OR
%    params    - 9 x 2 cell array of motor data for Version 2
%
%  Outputs:
%    -none
try
% nesting structure to set new values
if nargin ==1
  if ~isempty(params)
  if ischar(params)
      s = storedProps(obj,params);
      params = s;
  end

    for i = 1:length(params)
        switch (params{i,1})
            case 'J_m'
                obj.J_motor = params{i,2}/obj.g;
            case 'base_speed'
                bs_rpm = params{i,2};
                obj.omega_rated = bs_rpm/60*2*pi;
            case 'I_max'
                obj.current_rated = params{i,2};
            case 'torque'
                obj.torque_rated = params{i,2};
            case 'GR'
                obj.GR = params{i,2};
            case 'LS'
                obj.LS = params{i,2};
            case 'J_sec'
                obj.J_sec = params{i,2};
            case 'power'
                obj.hp = params{i,2};
            case 'material'
                obj.Roll = Parentroll(params{i,2});
                obj.RollTrue = 1;
                obj.R = obj.Roll.get('R');
            otherwise
               exception = MException('VerifyInput:NotUsed', ...
                   ['Input ' params{i,1} ' is not a valid name']);
               throw(exception);
        end
    end
  end
  if isempty(obj.torque_rated)
      obj.torque_rated = obj.hp*5252/bs_rpm;
  end
```

```
134           obj.J = obj.J_motor + obj.J_load_c;
              calculate(obj,1,1); %default Kps = 1, and 1 ft radius
136           if obj.RollTrue
                  obj.Roll.calculate();
138           end

140       else
              exception = MException('VerifyInput:NotEnoughInputs', ...
142               ['motor object requires 1 input argument.']);
              throw(exception);
144       end

146       catch ME
            disp([ME.message '!'])
148         err = MException('VerifyOutput:OutOfBounds', ...
                    'Exiting due to incorrect inputs.');
150         throw(err);
          end
152
          end
154           function [coi,K_m] = calculate(obj,K_ps,R)
                  % method calculates motor constant, motor speed at
156               % application speed, and the cross-over frequency dependent
                  % on the input proportional gain, K_ps, and the radius, R
158               obj.K_m = obj.torque_rated/obj.current_rated;
                  obj.S = obj.LS*obj.GR/2/pi/R; %(rpm)
160               obj.omega_co = obj.K_m*K_ps/obj.J;
                  coi = obj.omega_co;
162               if nargout == 2
                      K_m = obj.K_m;
164               end
              end
166           function calculateJ_sec(obj)
                  % Calculate J_sec for motor (and roll)
168               if obj.RollTrue %from motor perspective
                      % include motor, J_load, and roll inertias
170                   Jout = obj.J + (1/obj.GR)^2*...
                          obj.Roll.processInertia(obj.R);
172               else
                      %include motor and J_load inertia
174                   Jout = obj.J;
                  end
176               obj.J_sec = Jout*obj.omega_rated/obj.torque_rated;
              end
178           function processUnits(obj)
                  % switches between Imperial and SI based on units property
180               %
                  if obj.units
182                   obj.g = 9.81; %m/s^2
```

```
        else
            obj.g = 32.174; %ft/s^2
        end
    end
    function Jout = calculateInertia(obj,R,type)
        % calcs inertia of total roll and motor and spindles and
        % core from either the motor perspective or the roll
        % perspective, depending on type.
        %   R = radius in ft
        %   type = [0,1] 0=motor, 1=roll perspective for inertia
        if nargin<3
            type = 1; %assumes from the roll perspective
        end
        obj.R = R;
        if type
            %roll view
            Jin = obj.J;
            if obj.RollTrue
                Jout = (obj.GR)^2*Jin + obj.Roll.processInertia(R);
            else
                Jout = (obj.GR)^2*Jin;
                %I_eq*w2^2 = I_m*w1^2 + I_r*w2^2
                %I_eq = I_m*(w1/w2)^2 + I_r
                %w1/w2 = GR
                %I_eq = I_m*(GR)^2 + I_r
            end
        else
            %motor view
            Jin = obj.J;
            if obj.RollTrue
                Jout = Jin + (1/obj.GR)^2*obj.Roll.processInertia(R);
            else
                Jout = Jin;
            end
        end
    end
    function disp(obj)
        %method displays the internal parameters
        s = {'Motor Inertia:',obj.J_motor*obj.g,'lbm-ft^2'; ...
            'Base Speed:', obj.omega_rated*60/2/pi,'RPM';...
            'Rated Current:', obj.current_rated,'A';...
            'Rated Torque:',obj.torque_rated,'lbf-ft';...
            'Gear Ratio:',obj.GR,'(motor revs)/(shaft rev)'; ...
            'Line Speed:',obj.LS,'FPM';...
            'Per Normal Inertia:',obj.J_sec,'sec';...
            'Horsepower:',obj.hp,'Hp'};
        for i=1:8
            fprintf(1,'%s\t%.1f %s\n',s{i,1},s{i,2},s{i,3})
        end
```

```
232             if obj.RollTrue
                    fprintf(1,'Material: %s\n',obj.Roll.material)
234             end
                disp('    ')
236         end
            function val = get(obj,mfield)
238             %
                switch mfield
240                 case 'speed'
                        val = obj.LS;
242                 case 'J'
                        val = obj.J;
244                 case 'J_m'
                        val = obj.J_motor;
246                 case 'J_load'
                        val = obj.J_load_c;
248                 case 'J_sec'
                        calculateJ_sec(obj)
250                     val = obj.J_sec;
                    case 'K_m'
252                     val = obj.K_m;
                    case 'omega_co'
254                     val = obj.omega_co;
                    case 'base speed'
256                     val = obj.omega_rated*60/2/pi;
                    case 'rated torque'
258                     val = obj.torque_rated;
                    case {'PNI','per normal inertia','J_sec'}
260                     val = obj.J_sec;
                    case 'GR'
262                     val = obj.GR;
                    case 'R'
264                     val = obj.R;
                    case 'units'
266                     if obj.units == 1
                            disp('SI system')
268                     else
                            disp('Imperial system')
270                     end
                    otherwise
272                    exception = MException('VerifyInput:NotUsed', ...
                           ['Input ' mfield ' is not a valid name']);
274                    throw(exception);
                end
276         end
            function set(obj,mfield,val)
278             %
                switch mfield
280                 case 'speed'
```

```
                    obj.LS = val;
282                 case 'K_m'
                    obj.K_m = val;
284                 case 'J_sec'
                    obj.J_sec = val;
286                 case 'R'
                    obj.R = val;
288                 if obj.RollTrue
                        obj.Roll.set('R',val);
290                 end
                case 'J_load'
292                 if ischar(val)
                        val = storedProps(obj,val);
294                     obj.J_load_c = val{1,2};
                    else
296                     obj.J_load_c = val;
                    end
298                 obj.J = obj.J_motor + obj.J_load_c/obj.GR^2;
                case 'units'
300                 if obj.units ~= val
                        %change unit system
302                     if ismember(val,[0 1])
                            obj.units = val;
304                         processUnits;
                        else
306                         exception =  ...
                            MException('VerifyInput:NotUsed',...
308                         ['Input ' num2str(val) ' is not '...
                            'valid  for ' mfield]);
310                         throw(exception);
                        end
312                 end
                otherwise
314                 exception = MException('VerifyInput:NotUsed', ...
                        ['Input ' mfield ' is not a valid name']);
316                 throw(exception);
            end
318     end
        function s = storedProps(obj,type)
320         % outputs stored parameter sets for motor
            switch (type)
322             case 'L2875'
            s = {'J_m',7.1; 'base_speed', 1150; 'I_max', 48;...
324             'torque' 137;'GR', 1; 'LS', 800; 'J_sec' 0.2461227; ...
                'power', 30};
326             case 'L2875b'
            s = {'J_m',7.1; 'base_speed', 1420; 'I_max', 48;...
328             'torque' 137;'GR', 1; 'LS', 800; 'J_sec' 0.371999; ...
                'power', 36.9};
```

```
330                     case 'Unwind L2875b'
                    s = {'J_m',7.1; 'base_speed', 1420; 'I_max', 48;...
332                     'torque' 137;'GR', 1; 'LS', 800; 'J_sec' 0.371999; ...
                        'power', 36.9;'material','Tyvek-04'};
334                     case 'Rewind L2875'
                    s = {'J_m',7.1; 'base_speed', 1150; 'I_max', 48;...
336                     'torque' 137;'GR', 1; 'LS', 800; 'J_sec' 0.2461227; ...
                        'power', 30;'material','Tyvek-05'};
338                     case 'Unwind L2875b tyv 24'
                    s = {'J_m',7.1; 'base_speed', 1420; 'I_max', 48;...
340                     'torque' 137;'GR', 1; 'LS', 800; 'J_sec' 0.371999; ...
                        'power', 36.9;'material','Tyvek-03'};
342                     case 'Rewind L2875 tyv 24'
                    s = {'J_m',7.1; 'base_speed', 1150; 'I_max', 48;...
344                     'torque' 137;'GR', 1; 'LS', 800; 'J_sec' 0.2461227; ...
                        'power', 30;'material','Tyvek-02'};
346                     case 'Unwind L2875b PET'
                    s = {'J_m',7.1; 'base_speed', 1420; 'I_max', 48;...
348                     'torque' 137;'GR', 1; 'LS', 800; 'J_sec' 0.371999; ...
                        'power', 36.9;'material','PET-02'};
350                     case 'Rewind L2875 PET'
                    s = {'J_m',7.1; 'base_speed', 1150; 'I_max', 48;...
352                     'torque' 137;'GR', 1; 'LS', 800; 'J_sec' 0.2461227; ...
                        'power', 30;'material','PET-01'};
354                 case 'L2153'
                    s = {'J_m',1.31; 'base_speed', 1770; 'I_max', 20.09;...
356                     'torque' 15*33000/2/pi/1770;'GR', 1; 'LS', 800; ...
                        'J_sec' 1.6305; 'power', 15};
358                 case 'unwind'
                    s = {'J_load',9.349e-3+4.632384e-2};
360                 case 'rewind'
                    s = {'J_load',7.15e-3+4.632384e-2};
362                 end
            end
364     end
end
```

### I.8.3 The Gains Class Definition

The `Gains` object is the point of this research. The `Gains` object calculates motor control gains for a given motor based on criteria given to it as parameters. It needs the inertia of the motor/parent roll, and motor properties, so the motor object it is calculating gains for must be given as an argument. The code implements gain calculation methods from [8, 58, 60, 61, 87, 88], as well as the methods resident on the HSWL. Currently, there is a three step process to use the `Gains` object.

A `Gains` object must be initialized using the `Gains(params)` constructor. Then the object must be configured to calculate the correct gains using the `processGains(type,J,params2)` method. Then, finally, the gains can be calculated using the `calculate(J,motorObj)` method. This code may be refactored to move repetitive code to a method.

Listing I.17: Gains Class Definition Code

```
classdef Gains < handle
% Object to calculate gains
% Ben Reish (c) 2019 V2.7
%
%  Based on papers by B. Boulter:  [1] "The Effect of Speed Loop Bandwidths
%  and Line-Speed on System Natural Frequencies in Multi-Span Strip
%  Processing Systems", 1997 and [2] "Applying drive performance
%  specifications to systems applications. I. Speed performance", 2001
%  The rest is based on my work on calculating gains as simply as possible.
%  Other methods based on papers by Ben Reish and Karl Reid [3] "A
%  SYSTEMATIC METHOD FOR DETERMINING THE CONTROLLER GAINS IN A MULTI-SPAN
%  WEB LINE" IWEB 2019, available on www.shareok.org.
%
%  This object is takes 2 steps to initiallize.  First is initializing the
%  Gains object (see Gains(...) below);  then the Gains object is set
%  specialized to whatever type of controller it is by the processGains(..)
%  function.
%
%  Syntax: obj = Gains(params)
%       params =  {'name', '.7977zeta 26w_n'; ...; %  name string
%                  'zeta', .797703;... %  damping ratio
%                  'omega_n',26.2447; ...%  natural freq. (rad/s)
%                  'omega_co', 15; ...%  cross-over frequency (rad/s)
%                  'num', 650; ...%  numerator of transfer funct.
%                  'den', [1 41.87 650]; ...%  denominator of transfer funct.
%                  };
%  params is required.
%     obj.processGains(type,I,params2)
%        type = 0,1,2,3,4,5,6,7 depending on the type of control desired
%               type 0-Reish speed control, DC motor
%               type 1-Reish Load cell control, DC motor
%               type 2-Boulter Load cell control
%               type 3-Boulter speed control
%               type 4-Boulter Dancer control
%               type 5-Rockwell speed control
%               type 6-Rockwell Load cell control
%               type 7-Rockwell Dancer control
%               type 8-Reish speed control, AC motor
%               type 9-Reish Load cell control, AC motor
%               type 10  -Reish Dancer control, AC motor
%        I = current inertia of roll
```

```
%        params2 = {contents depend on what type was selected.}
%        obj2 =  a Motor object is passed into the funciton by certain
%               types.
%
% The Routh Approximation is executed by a function in another file in
% this parent folder, RouthApprox.m.
%
% Revision 2.7 - Changed the display to be type specific
%
% Revision 2.8 - Added RA method dancer control AC motor gain calculation
%



        properties (Access = public)
            name; %  string naming the set of gains
            zeta; %  damping coefficient
            T_r = 0; % rise time (sec)
            omega_n; %  natural frequency (rad/s)
            omega_co; %  cross-over frequency (rad/s)
            num_desired; % numerator polynomial desired
            den_desired; %  denominator polynomial coefficients desired
            safety_lim = 25; %  limit value for proportional gain
            num; %  numerator coefficients
            den; %  denominator coefficients
            K_p; %  proportional gain
            K_i; %  integral gain
            R;   %  radius (ft) Time varying
            type;%  type tells which method was used to calc gains
            BW;  %  bandwidth of speed loop
            tenctl_num = []; % tension controller trans. function numerator
            tenctl_den = []; % tension controller trans. fctn. denominator
            units = 0; %  unit system boolean, 0-Imperial, 1-SI

        end
        properties (Access = private)
            g = 32.174; %  acceleration of gravity in ft/s^2
            L; %  span length (ft)
            I; %  motor and roll inertia (slug-ft^2)
            LS; %  line speed (ft/s)
            EA; %  span spring constant (EA)  (lbf/ft)
            Gr; % gear ratio (motor revolutions per shaft revolutions)
            K_m; % motor torque constant (ft-lbf/amp)
            T_peak_ten; % peak time for tension (sec)
            J_sec; % Time required to accelerate to base speed at
                   %    rated torque (sec)
            noise; % speed loop noise ratio
            Curr_BW; %  Current/Torque loop bandwidth (if known) (rad/s)
```

```
91              T_speedloop_sample; % sample time for speed loop (sec)
                tenLoop_sample_ratio; % samples of tension loop per speed loop
93              K_max; % upper limit of the proportional gain
                ratio_lead_to_crossover = 0.25; % ratio of PI lead frequency to
95                % cross-over frequency
                Dan_travel; % length in inches of dancer travel, end to end
97              Dan_omega_co_target; % dancer position control loop cross over
                  % frequency target value.
99              rec_speed; % recovery speed (rpm)
                %torque; % motor rated torque
101             iter_max = 15; % internal iteration count max
                iter_flag = 0; % boolean for if the iteration at this inertia
103                            % has been completed
                k_r = 10; %default k_r. My gain method recalculates it.
105             EC_radius; % empty core radius in ft;
                Kp_ten; % Rockwell LC control base proportional gain
107             final_value_adj; %adjustment multiplier for final value
                mpn = 1; % dancer mass
109             Cdn = 0.04; % dancer damping
                Ksn = 0;  % dancer spring constant
111         end
            methods
113             function obj = Gains(params)
                    % Constructor for the class
115                 %
%  Inputs:
117 %    params    - 7 x 2 cell array of motor data
%
119 %  Outputs:
%    -none
121 try
% nesting structure to set new values
123 if nargin ==1
  if ~isempty(params)
125   if ischar(params)
        s = storedProps(obj,params);
127     params = s;
    end

129
      for i = 1:length(params)
131       switch (params{i,1})
              case 'name'
133               obj.name = params{i,2};
              case 'zeta'
135               obj.zeta = params{i,2};
              case 'omega_n'
137               obj.omega_n = params{i,2};
              case {'T_r','rise time'}
139               obj.T_r = params{i,2};
```

```
                    case 'num'
141                     obj.num = params{i,2};
                    case 'den'
143                     obj.den = params{i,2};
                    case 'omega_co'
145                     obj.omega_co = params{i,2};
                    otherwise
147                    exception = MException('VerifyInput:NotUsed', ...
                           ['Input ' params{i,1} ' is not a valid name']);
149                    throw(exception);
                end
151           end
          end
153       if isempty(obj.omega_n)
              try
155           obj.omega_n = (pi-atan(sqrt((1-obj.zeta^2)/obj.zeta)))...
                        /obj.T_r/sqrt(1-obj.zeta^2);
157           catch ME
              disp([ME.message '!'])
159           err = MException('VerifyInput:OutOfBounds', ...
                 'Gains object must have parameters zeta and omega_n or T_r.');
161           throw(err);
              end
163       end
          % calculate
165       obj.den_desired = [1 2*obj.zeta*obj.omega_n obj.omega_n^2];
          obj.num_desired = obj.omega_n^2;
167


169     else
            exception = MException('VerifyInput:NotEnoughInputs', ...
171            ['motor object requires 1 input argument.']);
            throw(exception);
173     end

175     catch ME
          disp([ME.message '!'])
177       err = MException('VerifyOutput:OutOfBounds', ...
                  'Exiting due to incorrect inputs.');
179       throw(err);
        end
181
        end
183           function gains = calculate_My_Ten(obj,obj2,I)
                  % Method calculates simple Kp and Ki for tension control
185               % according to reference [3] above.
                  % obj is the Gains object.  obj2 is the Gains object for
187               % the associated speed control to this tension control.
```

```
189             speednum = obj2.get('num');
                speedden = obj2.get('den');
191             num_t = conv(speednum*obj.R,obj.num);
                if length(num_t)<2
193                 num_t = [0 num_t];
                end
195             den_t = conv(speedden,obj.den);

197             obj.I = I;

199             [num_r,den_r] = WebController.RouthApprox(num_t,den_t,2);
                if length(num_r)<2
201                 num_r = [0 num_r];
                end
203             if length(num_r)<3
                    num_r = [0 num_r];
205             end

207             A = [obj.omega_n*obj.zeta -num_r(2) (-2*obj.zeta*...
                    obj.omega_n+den_r(2));...
209                 (2*obj.omega_n^3*obj.zeta^2-num_r(2)/num_r(3)*...
                    obj.zeta*obj.omega_n^3) -num_r(3) (-obj.omega_n+...
211                 den_r(3))];
                a_r = rref(A);
213             obj.k_r = a_r(1,3);
                obj.K_p = a_r(2,3);
215             obj.K_i = obj.k_r*obj.omega_n^3*obj.zeta/num_r(3);
                obj.tenctl_num = conv([obj.K_p obj.K_i],num_r);
217             temp(1) = length(den_r)+1;
                temp(2) = length(num_r);
219             tempb = zeros(1,temp(1));
                for i = temp(1):-1:(temp(1)-temp(2)+1)
221                 tempb(i) = num_r(i-(temp*[1 -1]'));
                end
223             obj.tenctl_den = [den_r 0]+tempb;
                obj.K_p = abs(obj.K_p);
225             obj.K_i = abs(obj.K_i);
                gains=[obj.K_p obj.K_i];
227         end
            function gains = calculate_Boulter(obj,I)
229             % method calculates simple Kp and Ki for tension control
                % with a load cell according to reference [1] above.
231             omega_co_ten = pi/obj.T_peak_ten;
                %omega_co_ten = 0.5*obj.omega_co;
233             if isempty(obj.K_p)
                    obj.K_p = 1;
235             end
                obj.K_i = 0.2*omega_co_ten;
237             %Calc omega_n and zeta
```

```
        obj.omega_n = sqrt(obj.K_i/obj.I*obj.K_m);
        obj.zeta = obj.K_p/2/obj.omega_n/obj.I*obj.K_m;

        gains = [obj.K_p 0.2*omega_co_ten, obj.omega_co*[0.7 7]];
    end
    function gains = calculate_Dancer_Boulter(obj)
        % this method is pulled from the HSWL code
        Dan_storage_time = obj.Dan_travel*60/12/obj.LS;
        obj.K_p = obj.Dan_omega_co_target*Dan_storage_time;
        obj.K_i = obj.K_p*obj.Dan_omega_co_target*...
            obj.ratio_lead_to_crossover;
        %Calc omega_n and zeta
        obj.omega_n = sqrt(obj.K_i/obj.I*obj.K_m);
        obj.zeta = obj.K_p/2/obj.omega_n/obj.I*obj.K_m;

        gains = [obj.K_p,obj.K_i, obj.omega_co, obj.K_i/obj.K_p*...
            obj.ratio_lead_to_crossover];
    end
    function gains = calculate_Speed(obj,I)
        % method calculates simple Kp and Ki for speed control
        % according to reference [3] above.
        % J*W(s)*s=K_m*u(s)+T(s)*R
        % W(s)/U(s) = K_m/J/s
        % (kp*s+ki)/s*K_m/J/s
        %-----------------
        % 1+ (kp*s+ki)/s*K_m/J/s
        % (kp*s+ki)*K_m/J
        %-----------------------
        % s^2 + (kp*s+ki)*K_m/J

        obj.K_i = obj.I*obj.omega_n^2/obj.K_m;
        obj.K_p = 2*obj.zeta*obj.omega_n*obj.I/obj.K_m;
        if obj.K_p > obj.safety_lim
            %limits proportional gain to safety limit value
            obj.K_p = obj.safety_lim;
        end
        obj.omega_co = obj.K_m*obj.K_p/obj.I;
        obj.num = obj.K_m/obj.I*[obj.K_p obj.K_i];
        obj.den = [1 (obj.K_p*obj.K_m/obj.I) ...
            (obj.K_i*obj.K_m/obj.I)];
        gains= [ obj.K_p obj.K_i];
    end
    function gains = calculate_Speed_Boulter(obj,obj2,I)
        % method calculates simple Kp and Ki for speed control from
        % paper [2]
        % obj is this Gains object.  obj2 is the motor object

        % Collect information from motor object
        Sb = obj2.get('base speed');
```

```
287                 torque = obj2.get('rated torque');
                    obj.J_sec = obj2.get('J_sec');
289                 obj.K_m = obj2.get('K_m');


291
                    obj.K_max = maxPropGainCalc(obj,obj.BW);
293                 if obj.units
                        obj.J_sec = I*Sb/9.55/torque;
295                 else
                        obj.J_sec = I*Sb/308/torque;
297                 end

299                 %test_BW = obj.K_p/obj.J_sec;
                    BW_max  = obj.K_max/obj.J_sec;
301                 BW_lim1 = pi/3/obj.T_speedloop_sample;
                    if ~isempty(obj.Curr_BW)
303                     %if no current/torque loop bandwidth is given
                         BW_lim2 = obj.Curr_BW/5;
305                 else
                        BW_lim2 = BW_lim1;
307                 end
                    if BW_lim2 < BW_lim1
309                     if BW_max > BW_lim2
                            BW_max = BW_lim2;
311                     end
                    else
313                     if BW_max > BW_lim1
                            BW_max = BW_lim1;
315                     end
                    end
317                 obj.BW = BW_max;
                    obj.K_p = obj.BW*obj.J_sec;
319                 if obj.K_p > obj.safety_lim
                        %limits proportional gain to safety limit value
321                     obj.K_p = obj.safety_lim;
                    end
323                 spdReg_w_co = obj.K_p/obj.J_sec;
                    if 0
325                     spdReg_w_ld = 0.4; %
                    else
327                     spdReg_w_ld = spdReg_w_co*...
                            obj.ratio_lead_to_crossover; %from HSWL
329                 end
                    obj.K_i = spdReg_w_ld*obj.K_p;
331                 %Calc omega_n and zeta
                    obj.omega_n = sqrt(obj.K_i/obj.I*obj.K_m);
333                 obj.zeta = obj.K_p/2/obj.omega_n/obj.I*obj.K_m;

335                 gains= [ obj.K_p obj.K_i];
```

```
end
function gains = calculate_Speed_Rockwell(obj,obj2,I)
    % Rockwell speed gains based on rated motor speed and
    % requested band width
    obj2.calculateJ_sec();
    obj.J_sec = obj2.J_sec;
    obj.R = obj2.R;
    obj.K_p = obj.omega_co*obj.J_sec;
    if obj.K_p > obj.K_max
        obj.K_p = obj.K_max;
    end
    obj.K_i = obj.K_p^2/obj.J_sec...
        *obj.ratio_lead_to_crossover;
    gains = [obj.K_p obj.K_i];

end
function gains = calculate_LC_Rockwell(obj,obj2)
    % Rockwell LC gains obj2 is a speed control gains object
    %

    KP_speed = obj2.K_p;
    obj.K_p = obj.Kp_ten*(obj2.R/obj.EC_radius)^2/KP_speed;
    obj.K_i = obj.omega_co*obj.K_p*obj.ratio_lead_to_crossover;

    gains = [obj.K_p obj.K_i];
end
function gains = calculate_Dancer_Rockwell(obj,obj2)
    % Rockwell Dancer gains obj2 is a speed control gains object
    %
    % this method is pulled from the HSWL code
    Dan_storage_time = obj.Dan_travel*60/12/obj.LS;
    obj.K_p = obj.Dan_omega_co_target*Dan_storage_time;
    obj.K_i = obj.K_p*obj.Dan_omega_co_target*...
        obj.ratio_lead_to_crossover;
    %Calc omega_n and zeta
    obj.omega_n = sqrt(obj.K_i/obj.I);
    obj.zeta = obj.K_p/2/obj.omega_n/obj.I;

    gains = [obj.K_p,obj.K_i, obj2.omega_co, ...
        obj2.omega_co/obj.ratio_lead_to_crossover];
end
function gains = calculate_Speed_AC_Reish(obj,obj2)
    %Reish speed AC gains (only operates on the speed error)
    obj.R = obj2.R;
    obj.I = obj2.calculateInertia(obj.R,0);
    obj.J_sec = obj2.J_sec;
    obj.K_i = obj.omega_n^2*obj.I;
    obj.K_p = 2*obj.zeta*obj.omega_n*obj.I;
    obj.omega_co = obj.K_i/obj.K_p;
```

```
385             obj.BW = obj.K_p/obj2.J_sec;
                gains = [obj.K_p, obj.K_i];
387             obj.num = [obj.K_p obj.K_i]/obj.I;
                obj.den = [1 obj.K_p/obj.I obj.K_i/obj.I];%
389         end
            function gains = calculate_LC_AC_Reish(obj,obj2)
391             %Reish LC AC gains (only operates on the tension error)
                %
393             % obj2 is a Gains object for the inner loop speed control
                obj.R = obj2.R;
395             speednum = obj2.get('num');
                speedden = obj2.get('den');
397             num_t = conv(speednum,obj.num);
                if length(num_t)<2
399                 num_t = [0 num_t];
                end
401             den_t = conv(speedden,obj.den);


403
                [num_r,den_r] = WebController.RouthApprox(num_t,den_t,2);
405             if length(num_r)<2
                    num_r = [0 num_r];
407             end
                if length(num_r)<3
409                 num_r = [0 num_r];
                end
411
                A = [obj.omega_n*obj.zeta -num_r(2) (-2*obj.zeta*...
413                 obj.omega_n+den_r(2));...
                    (2*obj.omega_n^3*obj.zeta^2-num_r(2)/num_r(3)*...
415                 obj.zeta*obj.omega_n^3) -num_r(3) (-obj.omega_n+...
                    den_r(3))];
417             a_r = rref(A);
                obj.k_r = a_r(1,3);
419             obj.K_p = a_r(2,3);
                obj.K_i = obj.k_r*obj.omega_n^3*obj.zeta/num_r(3);
421             obj.K_p = abs(obj.K_p)/obj.tenLoop_sample_ratio;
                obj.K_i = abs(obj.K_i)/obj.tenLoop_sample_ratio;
423             if ~isempty(obj.EC_radius)
                    obj.K_p = obj.K_p*(obj.R/obj.EC_radius)^2;
425                 obj.K_i = obj.K_i*(obj.R/obj.EC_radius)^2;
                end
427             if ~isempty(obj.final_value_adj)
                    obj.K_i = obj.K_i*obj.final_value_adj;
429                 %obj.K_i = 0;
                end
431             obj.omega_co = obj.K_i/obj.K_p;
                obj.tenctl_num = conv([obj.K_p obj.K_i],num_r);
433             temp(1) = length(den_r)+1;
```

```
        temp(2) = length(num_r);
435     tempb = zeros(1,temp(1));
        for i = temp(1):-1:(temp(1)-temp(2)+1)
437         tempb(i) = num_r(i-(temp*[1  -1]'));
        end
439     obj.tenctl_den = [den_r 0]+tempb;
        gains=[obj.K_p obj.K_i];
441 end
    function gains = calculate_Dan_AC_Reish(obj,obj2)
443     % gain calc. for dancer feedback, obj is Gains object and
        % obj2 is the Gains object for the speed control of same
445     % motor
        obj.R = obj2.R;
447     speednum = obj2.get('num');
        speedden = obj2.get('den');
449     num_t = conv(speednum,obj.num*obj.R/obj.mpn);
        if length(num_t)<2
451         num_t = [0 num_t];
        end
453     den_t = conv(speedden,obj.den);%span
        den_t = conv(den_t,[1 obj.Cdn/obj.mpn obj.Ksn/obj.mpn]);
455
        % Approximate the 5th order polynomial
457     [num_r,den_r] = WebController.RouthApprox(num_t,den_t,2);
        if length(num_r)<2
459         num_r = [0 num_r];
        end
461     if length(num_r)<3
            num_r = [0 num_r];
463     end

465     A = [obj.omega_n*obj.zeta -num_r(2) (-2*obj.zeta*...
            obj.omega_n+den_r(2));...
467         (2*obj.omega_n^3*obj.zeta^2-num_r(2)/num_r(3)*...
            obj.zeta*obj.omega_n^3) -num_r(3) (-obj.omega_n+...
469         den_r(3))];
        a_r = rref(A);
471     obj.k_r = a_r(1,3);
        obj.K_p = a_r(2,3);
473     obj.K_i = obj.k_r*obj.omega_n^3*obj.zeta/num_r(3);
        obj.K_p = abs(obj.K_p)/obj.tenLoop_sample_ratio;
475     obj.K_i = abs(obj.K_i)/obj.tenLoop_sample_ratio;

477     gains = [obj.K_p obj.K_i];
    end
479 function gains = calculate(obj,I,obj2)
        % method calculates which ever type of gain calculation
481     % that was set by processGains without resetting type.
        % type 0-Reish speed control
```

```
483                 % type 1-Reish Load cell control
                    % type 2-Boulter Load cell control
485                 % type 3-Boulter speed control
                    % type 4-Boulter Dancer control
487                 % type 5-Rockwell speed control
                    % type 6-Rockwell Load cell control
489                 % type 7-Rockwell Dancer control
                    % type 8-Reish Speed Ac control
491                 % type 9-Reish Load Cell Ac control
                    % type 10-Reish Dancer AC Control
493                 if nargin <3
                        obj2 = [];
495                 end
                    if abs(I - obj.I)>0
497                     obj.I = I;
                        obj.iter_flag = 0;
499                 end

501                 switch obj.type
                        case 0
503                         %Speed control method
                            gains = calculate_Speed(obj,I);
505                     case 1
                            %My method
507                         if isempty(obj2)
                                exception = MException('VerifyInput:Missing', ...
509                             ['Type 1 Tension control requires '...
                                'reference to the speed control object as '...
511                             'third input.']);
                                throw(exception);
513                         else
                                gains = calculate_My_Ten(obj,obj2,I);
515                         end
                        case 2
517                         %Boulter tension regulation
                            gains = calculate_Boulter(obj,I);
519                     case 3
                            if isempty(obj2)
521                             exception = MException('VerifyInput:Missing', ...
                                ['Boulter type speed control requires '...
523                             'reference to the motor object as third '...
                                'input.']);
525                             throw(exception);
                            else
527                           if ~isempty(obj.BW)
                                obj.iter_flag = 1;
529                           end
                              if obj.iter_flag
531                             gains = calculate_Speed_Boulter(obj,obj2,I);
```

```
                else
533                 ii = 0;
                    test = 1; test1 = 1; test2 = 1;
535                 gstore = [0 0];
                  while test && ii< obj.iter_max
537                   gains = calculate_Speed_Boulter(obj,obj2,I);
                      if abs(gains(1) - gstore(1))<1e-4
539                       test1 = 0;
                      else
541                       test1 = 1;
                      end
543                   if abs(gains(2) - gstore(2)) < 1e-4
                          test2 = 0;
545                   else
                          test2 = 1;
547                   end
                      if ~test1 && ~test2
549                       test = 0;
                          obj.iter_flag = 1;
551                   end
                      ii = ii + 1;
553                   gstore = gains;
                  end
555               %disp(ii)
                end
557           end
          case 4
559           %Boulter Dancer gain from HSWL code
              gains = calculate_Dancer_Boulter(obj);
561       case 5
              %Rockwell gains (follows Boulter but not max value)
563           if isempty(obj2)
                  exception = MException('VerifyInput:Missing', ...
565               ['Type 5 Speed control requires '...
                  'reference to the motor object as third '...
567               'input.']);
                  throw(exception);
569           else
                  gains = calculate_Speed_Rockwell(obj,obj2,I);
571           end
          case 6
573           %Rockwell LC gains (follows Boulter but
              % not max value)
575           if isempty(obj2)
                  exception = MException('VerifyInput:Missing', ...
577               ['Type 6 Rockwell LC control requires '...
                  'reference to the speed gains object as '...
579               'third input.']);
                  throw(exception);
```

```
581                 else
                        gains = calculate_LC_Rockwell(obj,obj2);
583                 end
                case 7
585                 %Rockwell Dancer gains (follows Boulter but
                    % not max value)
587                 if isempty(obj2)
                        exception = MException('VerifyInput:Missing', ...
589                     ['Type 7 Rockwell Dancer control requires '...
                        'reference to the speed control object as '...
591                     'third input.']);
                        throw(exception);
593                 else
                        gains = calculate_Dancer_Rockwell(obj,obj2);
595                 end
                case 8
597                 %Reish speed AC gains
                    if isempty(obj2)
599                     exception = MException('VerifyInput:Missing', ...
                        ['Type 8 RA method Speed control requires '...
601                     'reference to the motor object as third '...
                        'input.']);
603                     throw(exception);
                    else
605                     gains = calculate_Speed_AC_Reish(obj,obj2);
                    end
607             case 9
                    %Reish LC AC gains
609                  if isempty(obj2)
                        exception = MException('VerifyInput:Missing', ...
611                     ['Type 9 RA method LC control requires '...
                        'reference to the speed control object as '...
613                     'third input.']);
                        throw(exception);
615                  else
                        gains = calculate_LC_AC_Reish(obj,obj2);
617                  end
                 case 10
619                 %Reish Dancer AC gains
                     if isempty(obj2)
621                     exception = MException('VerifyInput:Missing', ...
                        ['Type 10 RA method LC control requires '...
623                     'reference to the speed control object as '...
                        'third input.']);
625                     throw(exception);
                    else
627                     gains = calculate_Dan_AC_Reish(obj,obj2);
                    end
629             otherwise
```

```
                    exception = MException('VerifyInput:NotUsed', ...
631                     ['Type ' obj.type ' is not a valid type']);
                    throw(exception);
633             end
                obj.den_desired = [1 2*obj.zeta*obj.omega_n obj.omega_n^2];
635             obj.num_desired = obj.omega_n^2;
                %obj.omega_co = obj.K_i/obj.K_p;
637             %obj.BW = obj.K_p/obj.J_sec;
            end
639         function processGains(obj,type,I,params)
                % Sets type of gain calculation desired and calculates
641             % type is {0 1 2 3 4 5 6 7 8 9} 0= Speed DC, 1 = my method
                % DC,2 = Boulter, 3 = Boulter Speed control method, 4 =
643             % Boulter Dancer position control method, 5 = Rockwell
                % speed, 6 = Rockwell LC, 7 = Rockwell Dancer, 8 - my method
645             % speed AC, 9 - my method load cell AC, 10 - my method
                % dancer AC,
647             if isempty(obj.type)
                    obj.type = type;
649             end
                obj.I = I;
651             if obj.type == 0
                    %RA method speed DC
653                 if nargin == 4
                        for i = 1:length(params)
655                       switch (params{i,1})
                              case 'span length'
657                               obj.L = params{i,2};
                              case 'radius'
659                               obj.R = params{i,2};
                              case 'line speed'
661                               obj.LS = params{i,2};
                              case 'span spring'
663                               obj.EA = params{i,2};
                              case 'torque constant'
665                               obj.K_m = params{i,2};
                              otherwise
667                               exception = ...
                                      MException('VerifyInput:NotUsed', ...
669                         ['Parmeter ' params{i,1}...
                             ' is not a valid parameter' ...
671                          ' for General Speed control method.']);
                                  throw(exception);
673                       end
                        end
675                 else

677                     exception = ...
                            MException('VerifyInput:NotEnoughInputs', ...
```

```
679                         ['A list of parameters is required']);
                        throw(exception);
681                 end
                elseif obj.type == 1
683                 %RA method tension DC
                    if nargin == 4
685                     for i = 1:length(params)
                          switch (params{i,1})
687                           case 'span length'
                                obj.L = params{i,2};
689                           case 'radius'
                                obj.R = params{i,2};
691                           case {'line speed', 'LS'} %in ft/s
                                obj.LS = params{i,2};
693                           case 'span spring'
                                obj.EA = params{i,2};
695                           case 'torque constant'
                                obj.K_m = params{i,2};
697                           case 'k_r'
                                obj.k_r = params{i,2};
699                           otherwise
                                exception = ...
701                                MException('VerifyInput:NotUsed', ...
                                ['Parmeter ' params{i,1}...
703                                 ' is not a valid parameter' ...
                                 ' for RA Method Tension control method.']);
705                                 throw(exception);
                          end
707                     end
                        % create num and den for span
709                     Vn = obj.LS; %convert to ft/s
                        obj.num = -obj.EA/obj.L;
711                     obj.den = [1 Vn/obj.L];
                    else
713
                        exception = ...
715                        MException('VerifyInput:NotEnoughInputs', ...
                            ['A list of parameters is required']);
717                     throw(exception);
                    end
719             elseif obj.type == 2
                    %Boulter method
721                 if nargin == 4
                        for i = 1:length(params)
723                       switch (params{i,1})
                            case {'Kp','K_p','kp'}
725                             obj.K_p = params{i,2};
                            case 'Peak Time'
727                             obj.T_peak_ten = params{i,2};
```

```
                    case 'span length'
729                     obj.L = params{i,2};
                    case 'radius'
731                     obj.R = params{i,2};
                    case {'line speed', 'LS'} %in ft/s
733                     obj.LS = params{i,2};
                    case 'span spring'
735                     obj.EA = params{i,2};
                    case 'torque constant'
737                     obj.K_m = params{i,2};
                    otherwise
739                     exception = ...
                           MException('VerifyInput:NotUsed', ...
741                    ['Parmeter ' params{i,1}...
                     ' is not a valid parameter' ...
743                  ' for Boulter Tension control method.']);
                        throw(exception);
745               end
                end
747
            else
749
                exception = ...
751                MException('VerifyInput:NotEnoughInputs', ...
                    ['A list of parameters is required']);
753             throw(exception);
            end
755         elseif obj.type == 3
             %Boulter Speed method
757          obj.K_p = 1; %default value
            if nargin == 4
759             for i = 1:length(params)
                  switch (params{i,1})
761                   case 'J_sec'
                        obj.J_sec = params{i,2};
763                   case 'PN_noise'
                        obj.noise = params{i,2};
765                   case 'Kp'
                        obj.K_p = params{i,2};
767                   case 'Current BW'
                        obj.Curr_BW = params{i,2};
769                   case 'Sample Time'
                        obj.T_speedloop_sample = params{i,2};
771                   case 'BW'
                        obj.BW = params{i,2};
773                   case 'recovery speed'
                        obj.rec_speed = params{i,2};
775                   case 'torque'
                        obj.torque = params{i,2};
```

```
777                         case 'torque constant'
                                obj.K_m = params{i,2};
779                         case 'lead to crossover ratio'
                                obj.ratio_lead_to_crossover = params{i,2};
781                         otherwise
                                exception = ...
783                                 MException('VerifyInput:NotUsed', ...
                            ['Parmeter ' params{i,1}...
785                          ' is not a valid parameter' ...
                             ' for Boulter Speed control method.']);
787                             throw(exception);
                        end
789                 end
                    obj.K_max = maxPropGainCalc(obj,obj.BW);
791                 BW_max  = obj.K_max/obj.J_sec;
                    BW_lim1 = pi/3/obj.T_speedloop_sample;
793                 if ~isempty(obj.Curr_BW)
                        BW_lim2 = obj.Curr_BW/5;
795                 else
                        BW_lim2 = BW_lim1;
797                 end
                    if BW_lim2 < BW_lim1
799                     if BW_max > BW_lim2
                            BW_max = BW_lim2;
801                     end
                    else
803                     if BW_max > BW_lim1
                            BW_max = BW_lim1;
805                     end
                    end
807                 obj.BW = BW_max;
                    obj.K_p = obj.BW*obj.J_sec;
809             end
            elseif obj.type == 4
811             %Boulter dancer gain calc from HSWL
                if nargin == 4
813                 for i = 1:length(params)
                        switch (params{i,1})
815                         case 'Dancer travel'
                                obj.Dan_travel = params{i,2};
817                         case 'Dancer crossover freq'
                                obj.Dan_omega_co_target = params{i,2};
819                         case 'ratio_lead_to_crossover'
                                obj.ratio_lead_to_crossover = params{i,2};
821                         case {'Line Speed','LS','line speed'}
                                obj.LS = params{i,2};
823                         case 'torque constant'
                                obj.K_m = params{i,2};
825                         otherwise
```

```
                    exception = ...
827                     MException('VerifyInput:NotUsed', ...
                [['Parmeter ' params{i,1} ...
829              ' is not a valid parameter' ...
                 ' for Boulter Dancer control method.']);
831                 throw(exception);
              end
833         end
            else
835          exception = MException('VerifyInput:NotUsed', ...
               ['Parmeter list needed ' ...
837             ' for Boulter Dancer control method.']);
                    throw(exception);
839     end
        elseif obj.type == 5
841         %Rockwell gains speed
             if nargin == 4
            for i = 1:length(params)
841          switch (params{i,1})
                 case {'BW','band width'}
                     obj.BW = params{i,2};
847              case 'ratio_lead_to_crossover'
                     obj.ratio_lead_to_crossover = params{i,2};
849              case {'Line Speed','LS','line speed'}
                     obj.LS = params{i,2};
851              case 'omega_co'
                     obj.omega_co = params{i,2};
853              otherwise
                     exception = ...
855                      MException('VerifyInput:NotUsed', ...
                [['Parmeter ' params{i,1}...
857              ' is not a valid parameter' ...
                 ' for Rockwell speed control method.']);
859                 throw(exception);
              end
861         end
             else
863          exception = MException('VerifyInput:NotUsed', ...
               ['Parmeter list needed ' ...
865             ' for Rockwell speed control method.']);
                    throw(exception);
867         end
         elseif obj.type == 6
869         %Rockwell LC gains
             if nargin == 4
            for i = 1:length(params)
871          switch (params{i,1})
                 case {'BW','band width'}
873                  obj.BW = params{i,2};
```

```
875                         case 'ratio_lead_to_crossover'
                                obj.ratio_lead_to_crossover = params{i,2};
877                         case 'Kp_ten'
                                obj.Kp_ten = params{i,2};
879                         case 'omega_co'
                                obj.omega_co = params{i,2};
881                         case {'Empty Core Radius','EC_radius'}
                                obj.EC_radius = params{i,2};
883                         otherwise
                                exception = ...
885                                 MException('VerifyInput:NotUsed', ...
                            ['Parmeter ' params{i,1}...
887                             ' is not a valid parameter' ...
                             ' for Rockwell LC control method.']);
889                             throw(exception);
                        end
891                     end
                        if isempty(obj.ratio_lead_to_crossover)
893                         obj.ratio_lead_to_crossover = 1;
                        end
895                     else
                        exception = MException('VerifyInput:NotUsed', ...
897                         ['Parmeter list needed ' ...
                             ' for Rockwell LC control method.']);
899                             throw(exception);
                    end
901             elseif obj.type == 7
                    %Rockwell Dancer gains
                    if nargin == 4
                    for i = 1:length(params)
905                     switch (params{i,1})
                            case 'Dancer travel'
907                             obj.Dan_travel = params{i,2};
                            case 'Dancer crossover freq'
909                             obj.Dan_omega_co_target = params{i,2};
                            case 'ratio_lead_to_crossover'
911                             obj.ratio_lead_to_crossover = params{i,2};
                            case {'Line Speed Max','LS','line speed max'}
913                             obj.LS = params{i,2};
                            otherwise
915                             exception =...
                                    MException('VerifyInput:NotUsed', ...
917                         ['Parmeter ' params{i,1}...
                             ' is not a valid parameter' ...
919                          ' for Rockwell Dancer control method.']);
                                throw(exception);
921                     end
                    end
923                  else
```

```
                exception = MException('VerifyInput:NotUsed', ...
925                 ['Parmeter list needed ' ...
                     ' for Rockwell Dancer control method.']);
927                 throw(exception);
                end
929         elseif obj.type == 8
            %Reish Speed AC gains
931         if nargin == 4
               for i = 1:length(params)
933              switch (params{i,1})
                     case 'name'
935                      obj.name = params{i,2};
                     case {'safety Limit','Kp limit'}
937                      obj.safety_lim = params{i,2};
                     case {'line speed', 'LS'} %in ft/s
939                      obj.LS = params{i,2};
                     case 'zeta'
941                       obj.zeta = params{i,2};
                     case 'omega_n'
943                       obj.omega_n = params{i,2};
                     otherwise
945                      exception = ...
                             MException('VerifyInput:NotUsed', ...
947                 ['Parmeter ' params{i,1} ...
                     'is not a valid parameter' ...
949                  ' for RA Method Speed AC control method.']);
                         throw(exception);
951              end
               end
953         end
           % create num and den for motor
955

            obj.num = [2*obj.zeta*obj.omega_n obj.omega_n^2];
957         obj.den = [1 2*obj.zeta*obj.omega_n obj.omega_n^2];
         elseif obj.type == 9
959         %Reish LC AC gains
            if nargin == 4
961            for i = 1:length(params)
                 switch (params{i,1})
963                  case {'safety Limit','Kp limit'}
                         obj.safety_lim = params{i,2};
965                  case {'span length','L'}
                         obj.L = params{i,2};
967                  case 'radius'
                         obj.R = params{i,2};
969                  case 'EC_radius'
                         obj.EC_radius = params{i,2};
971                  case {'line speed', 'LS'} %in ft/s
                         obj.LS = params{i,2};
```

```
973                         case {'span spring','EA'}
                                obj.EA = params{i,2};
975                         case 'final_value_adj'
                                obj.final_value_adj = params{i,2};
977                          case {'Gr','gear ratio'}
                                obj.Gr = params{i,2};
979                         case 'tenLoop_sample_ratio'
                                obj.tenLoop_sample_ratio = params{i,2};
981                         case 'k_r'
                                obj.k_r = params{i,2};
983                         case 'zeta'
                                 obj.zeta = params{i,2};
985                         case 'omega_n'
                                 obj.omega_n = params{i,2};
987                         case 'name'
                                obj.name = params{i,2};
989                         otherwise
                                exception =...
991                                    MException('VerifyInput:NotUsed', ...
                            ['Parmeter ' params{i,1} ...
993                          ' is not a valid parameter' ...
                             ' for RA Method LC AC control method.']);
995                             throw(exception);
                          end
997                     end
                        % create num and den for span
999                     Vn = obj.LS/60; %convert to ft/s
                        obj.num = [0 obj.EA/obj.L];
1001                    obj.den = [1 Vn/obj.L];
                    else
1003                     exception = MException('VerifyInput:NotUsed', ...
                            ['Parmeter list needed ' ...
1005                         ' for RA LC control method.']);
                                throw(exception);
1007                end
                elseif obj.type == 10
1009                %Reish Dan AC gains
                    if nargin == 4
1011                    for i = 1:length(params)
                          switch (params{i,1})
1013                          case {'safety Limit','Kp limit'}
                                  obj.safety_lim = params{i,2};
1015                          case {'span length','L'}
                                  obj.L = params{i,2};
1017                          case 'radius'
                                  obj.R = params{i,2};
1019                          case 'EC_radius'
                                  obj.EC_radius = params{i,2};
1021                          case {'line speed', 'LS'} %in ft/s
```

```
                        obj.LS = params{i,2};
                    case {'span spring','EA'}
                        obj.EA = params{i,2};
                    case 'final_value_adj'
                        obj.final_value_adj = params{i,2};
                    case {'Gr','gear ratio'}
                        obj.Gr = params{i,2};
                    case 'tenLoop_sample_ratio'
                        obj.tenLoop_sample_ratio = params{i,2};
                    case {'Dancer Mass','Mpn','dan_mass'}
                        obj.mpn = params{i,2};
                    case 'Cdn'
                        obj.Cdn = params{i,2};
                    case 'Ksn'
                        obj.Ksn = params{i,2};
                    case 'k_r'
                        obj.k_r = params{i,2};
                    case 'zeta'
                         obj.zeta = params{i,2};
                    case 'omega_n'
                         obj.omega_n = params{i,2};
                    case 'name'
                        obj.name = params{i,2};
                    otherwise
                        exception =...
                            MException('VerifyInput:NotUsed', ...
                    ['Parmeter ' params{i,1} ...
                     ' is not a valid parameter' ...
                     ' for RA Method Dan AC control method.']);
                        throw(exception);
                  end
                end
                % create num and den for span
                Vn = obj.LS/60; %convert to ft/s
                obj.num = [0 obj.EA/obj.L];
                obj.den = [1 Vn/obj.L];
            else
                exception = MException('VerifyInput:NotUsed', ...
                   ['Parmeter list needed ' ...
                    ' for RA Dancer control method.']);
                        throw(exception);
            end
        end
        %obj.omega_co = obj.K_m*Ks/obj.I*60/2/pi;
        if obj.T_r == 0
            obj.T_r = (pi-atan(sqrt((1-obj.zeta^2)/obj.zeta)))...
                /obj.omega_n/sqrt(1-obj.zeta^2);
        end
        %gains = calculate(obj,I);
```

```
1071
                end
1073            %%
                function disp(obj)
1075                %method displays the internal parameters based on type
                    %
1077                if isempty(obj.type)
                        obj.type = -99;
1079                end
                    switch obj.type
1081                    case 0 % type 0-Reish speed control, DC motor
                            s = {'Omega_n:',obj.omega_n,'rad/s'; ...
1083                        'zeta:', obj.zeta,'';...
                        'K_p:', obj.K_p,'';...
1085                        'K_i:',obj.K_i,'1/s';...
                        'K_m:',obj.K_m,'torque/amp'};
1087                    case 1 % type 1-Reish Load cell control, DC motor
                            s = {'Omega_n:',obj.omega_n,'rad/s'; ...
1089                        'zeta:', obj.zeta,'';...
                        'K_p:', obj.K_p,'';...
1091                        'K_i:',obj.K_i,'1/s';...
                        'K_m:',obj.K_m,'torque/amp'};
1093                    case 2 % type 2-Boulter Load cell control
                            s = {'Bandwith:' obj.BW,'rad/s';...
1095                            'K_p:', obj.K_p,'';...
                            'K_i:',obj.K_i,'1/s';...
1097                            'Lead to crossover ratio:'
                                obj.ratio_lead_to_crossover,'';...
1099                            };
                        case 3 % type 3-Boulter speed control
1101                        s = {'Bandwith:' obj.BW,'rad/s';...
                            'J_sec:',obj.J_sec,'s';...
1103                            'K_p:', obj.K_p,'';...
                            'K_i:',obj.K_i,'1/s';...
1105                            'Lead to crossover ratio:'...
                                obj.ratio_lead_to_crossover,'';...
1107                            };
                        case 4 % type 4-Boulter Dancer control
1109                    case 5 % type 5-Rockwell speed control
                            s = {'Bandwith:' obj.BW,'rad/s';...
1111                            'Kp_ten:',obj.Kp_ten,'';...
                            'J_sec:',obj.J_sec,'s';...
1113                            'omega_co:',obj.omega_co,'rad/s';...
                            'K_p:', obj.K_p,'';...
1115                            'K_i:',obj.K_i,'1/s';...
                            'Lead to crossover ratio:'...
1117                            obj.ratio_lead_to_crossover,'';...
                            };
1119                    case 6 % type 6-Rockwell Load cell control
```

```
                s = {'Bandwith:' obj.BW,'rad/s';...
1121                'Kp_ten:',obj.Kp_ten,'';...
                    'J_sec:',obj.J_sec,'s';...
1123                'omega_co',obj.omega_co,'rad/s';...
                    'K_p:', obj.K_p,'';...
1125                'K_i:',obj.K_i,'1/s';...
                    'Lead to crossover ratio:'...
1127                    obj.ratio_lead_to_crossover,'';...
                    };
1129            case 7 % type 7-Rockwell Dancer control
                s = {'Bandwith:' obj.BW,'rad/s';...
1131                'Dancer travel',obj.Dan_travel*12, 'in';...
                    'Dancer crossover freq',...
1133                    obj.Dan_omega_co_target,'rad/s';...
                    'ratio lead to crossover',...
1135                    obj.ratio_lead_to_crossover,'';...
                    'Line Speed Max',obj.LS*60,'FPM';...
1137                'PN storage:',obj.Dan_travel/obj.LS,'s';...
                    'K_p:', obj.K_p,'';...
1139                'K_i:',obj.K_i,'1/s';...
                    };
1141            case 8 % type 8-Reish speed control, AC motor
                s = {'Omega_n:',obj.omega_n,'rad/s'; ...
1143            'zeta:', obj.zeta,'';...
                'T_r:',obj.T_r,'sec';...
1145            'K_p:', obj.K_p,'';...
                'K_i:',obj.K_i,'1/s';...
1147            'BW:',obj.BW,'rad/s';...
                };
1149            case 9 % type 9-Reish Load cell control, AC motor
                s = {'Omega_n:',obj.omega_n,'rad/s'; ...
1151            'zeta:', obj.zeta,'';...
                'T_r:',obj.T_r,'sec';...
1153            'K_p:', obj.K_p,'';...
                'K_i:',obj.K_i,'1/s';...
1155            'BW:',obj.BW,'rad/s';...
                'Gr:',obj.Gr,'motor rev/shaft rev';...
1157            'num:',obj.num,'';...
                'den:',obj.den,'';...
1159            };
                case 10 % type   -Reish Dancer control, AC motor
1161        s = {'Omega_n:',obj.omega_n,'rad/s'; ...
                'zeta:', obj.zeta,'';...
1163            'T_r:',obj.T_r,'sec';...
                'K_p:', obj.K_p,'';...
1165            'K_i:',obj.K_i,'1/s';...
                'Dancer Mass:',obj.mpn*g,'lbm';...
1167            };
            otherwise
```

```
1169                    s = {'Name','',obj.name;...
                         'Omega_n:',obj.omega_n,'rad/s'; ...
1171                     'zeta:', obj.zeta,'';...
                         'omega_co', obj.omega_co,'rad/s'; ...
1173                         %  cross-over frequency (rad/s)
                         };
1175                end
                    if ~isempty(obj.name)
1177                    fprintf(1,'%s\n',obj.name)
                    end
1179                m=size(s);
                    for i=1:m(1,1)
1181                    fprintf(1,'%s\t%.3f %s\n',s{i,1},s{i,2},s{i,3})
                    end
1183                disp('    ')
                    if obj.type==-99
1185                    obj.type = [];
                    end
1187            end

1189            function val = get(obj,mfield)
                    %
1191                switch mfield
                        case 'I'
1193                        val = obj.I;
                        case 'R'
1195                        val = obj.R;
                        case 'EA'
1197                        val = obj.EA;
                        case {'L','span length'}
1199                        val = obj.L;
                        case 'omega_co'
1201                        val = obj.omega_co;
                        case 'zeta'
1203                        val = obj.zeta;
                        case 'omega_n'
1205                        val = obj.omega_n;
                        case 'LS'
1207                        val = obj.LS;
                        case 'K_m'
1209                        val = obj.K_m;
                        case 'K_p'
1211                        val = obj.K_p;
                        case 'K_i'
1213                        val = obj.K_i;
                        case 'Mpn'
1215                        val = obj.mpn;
                        case 'Cdn'
1217                        val = obj.Cdn;
```

```
                case 'Ksn'
1219                val = obj.Ksn;
                case 'num'
1221                val = obj.num;
                case 'den'
1223                val = obj.den;
                case 'Peak Time'
1225                val = obj.T_peak_ten;
                case 'lead to crossover ratio'
1227                val = obj.ratio_lead_to_crossover;
                case 'tension control num'
1229                val = obj.tenctl_num;
                case 'tension control den'
1231                val = obj.tenctl_den;
                case 'units'
1233                if obj.units == 1
                        disp('SI system')
1235                else
                        disp('Imperial system')
1237                end
                otherwise
1239                exception = MException('VerifyInput:NotUsed', ...
                        ['Input ' mfield ' is not a valid name']);
1241                throw(exception);
            end
1243        end
            function set(obj,mfield,val)
1245            %
                switch mfield
1247                case 'I'
                        obj.I = val;
1249                case 'R'
                        obj.R = val;
1251                case {'LS', 'line speed'}
                        obj.LS = val;
1253                case 'span length'
                        obj.L = val;
1255                case {'K_p','Kp','kp'}
                        obj.K_p = val;
1257                case 'Kp_ten'
                        obj.Kp_ten = val;
1259                case {'Kp_max','Safety Limit'}
                        obj.Kp_max = val;
1261                case 'zeta'
                        obj.zeta = val;
1263                case 'Gr'
                        obj.Gr = val;
1265                case 'Mpn'
                        obj.mpn = val;
```

```
1267                    case 'Cdn'
                            obj.Cdn = val;
1269                    case 'Ksn'
                            obj.Ksn = val;
1271                    case {'omega_n','natural freq'}
                            obj.omega_n = val;
1273                    case {'omega_co','crossover freq'}
                            obj.omega_co = val;
1275                    case 'Peak Time'
                            obj.T_peak_ten = val;
1277                    case 'lead to crossover ratio'
                            obj.ratio_lead_to_crossover = val;
1279                    case 'units'
                            if obj.units ~= val
1281                            %change unit system
                                if ismember(val,[0 1])
1283                                obj.units = val;
                                    processUnits;
1285                            else
                                  exception=MException('VerifyInput:NotUsed',...
1287                              ['Input ' num2str(val) ' is not valid  for '...
                                  mfield]);
1289                                throw(exception);
                                end
1291                        end
                        otherwise
1293                        exception = MException('VerifyInput:NotUsed', ...
                                ['Input ' mfield ' is not a valid name']);
1295                        throw(exception);
                    end
1297            end
                function k_max = maxPropGainCalc(obj,BW)
1299                % function to interpolate maximum proportional gain allowed
                    % per paper [2], fig. 3.
1301                if obj.noise < 0.0001
                        k_max = (obj.omega_n/BW)^0.197 + 95;
1303                elseif obj.noise < 0.001
                        yu = (obj.omega_n/BW)^0.197 + 95;
1305                    yl = (obj.omega_n/BW)^0.381 + 9;
                        k_max = yl*(yu/yl)^((obj.noise-0.001)/(0.0001-0.001));
1307                elseif obj.noise < 0.01
                        yu = (obj.omega_n/BW)^0.381 + 9;
1309                    yl = (obj.omega_n/BW)^0.425 + .9;
                        k_max = yl*(yu/yl)^((obj.noise-0.01)/(0.001-0.01));
1311                elseif obj.noise < 0.1
                        yu = (obj.omega_n/BW)^0.425 + .9;
1313                    yl = (obj.omega_n/BW)^0.508 + .09;
                        k_max = yl*(yu/yl)^((obj.noise-0.1)/(0.01-0.1));
1315                else
```

```
            k_max = (obj.omega_n/BW)^0.508 + .09;
        end
    end
    function RatioTest(obj,cs)
        %Functions to mimic the rise and fall of those in Fig 5 of
        %paper [2]
        pernormal_speed = obj.recovery_speed/obj2.get('base speed');
        yax = pernormal_speed*obj.BW*obj.J_sec*100/obj.load_shock;
        switch cs
            case 1
                t = 2.5;
            case 2
                t = 2.1;
            case 3
                t = .95;
            otherwise
                t = .87;
        end
        y=1;
        flag =1;
        while abs(y-yax)>1e-4
           switch cs
            case 1
                %0.1 curve
                y=sqrt(1.45)*((1-exp(-1*t))-(1-exp(-(1/10)*t)));
            case 2
                %0.2 curve
                y=sqrt(2)*((1-exp(-1*t))-(1-exp(-(1/5)*t)));
            case 3
                %0.3 curve
                y=sqrt(3)*((1-exp(-1*t))-(1-exp(-(3/10)*t)));
            otherwise
                %0.4 curve
                y=sqrt(4.3)*((1-exp(-1*t))-(1-exp(-(4/10)*t)));
           end
          if y > yax && flag
              tu = t;
              t=t+1;
              yu=y;
          elseif y ≤ yax && flag
              yl = y;
              tl = t;
              t = t - 0.5;
              flag = 0;
          else
              %bounded iteration
              if y-yax < 0
                  yl = y;
                  tl = t;
```

```
                    else
                        yu = y;
                        tu = t;
                    end
                    t = 0.5*(tu+tl);
                end
            end

        end
        function s = storedProps(obj,mfield)
            % outputs stored parameter sets for motor
            switch (mfield)
                case 'test-01'
            s = {'name', '.7zeta 10w_n'; ...; %  name string
                 'zeta', .7;... %  damping ratio
                 'omega_n',10; ...%  natural freq. (rad/s)
                 'omega_co', 2; ...%  cross-over frequency (rad/s)
                 'num', 10; ...%  numerator of transfer funct.
                 'den', [1 12 22]; ...%  den. transfer funct.
                 };
            case 'TEST-02'
            s = {'name', '.9zeta 20.5w_n'; ...; %  name string
                 'zeta', .9;... %  damping ratio
                 'omega_n',20.57; ...%  natural freq. (rad/s)
                 'omega_co', 8; ...%  cross-over frequency (rad/s)
                 'num', 423; ...%  numerator of transfer funct.
                 'den', [1 12 432]; ...%  den. transfer funct.
                 };
            case 'TEST-03'
            s = {'name', '.7977zeta 26w_n'; ...; %  name string
                 'zeta', .797703;... %  damping ratio
                 'omega_n',26.2447; ...%  natural freq. (rad/s)
                 'omega_co', 15; ...%  cross-over frequency (rad/s)
                 'num', 650; ...%  numerator of transfer funct.
                 'den', [1 41.87 650]; ...%  den. transfer funct.
                 };
                case 'TEST-04'
            s = {'name', '.9zeta 20w_n'; ...; %  name string
                 'zeta', .9;... %  damping ratio
                 'omega_n',20.57; ...%  natural freq. (rad/s)
                 'omega_co', 15; ...%  cross-over frequency (rad/s)
                 'num', 400; ...%  numerator of transfer funct.
                 'den', [1 1.8*20.57 400]; ...%  den. transfer funct.
                 };
                 case 'TEST-05'
            s = {'name', '.25zeta 140w_n'; ...; %  name string
                 'zeta', .25;... %  damping ratio
                 'omega_n',140; ...%  natural freq. (rad/s)
                 'omega_co', 15; ...%  cross-over frequency (rad/s)
```

```
                    'num', 19600; ...%  numerator of transfer funct.
                    'den', [1 2*.25*140 19600]; ...%  den. transfer funct.
                    };
                    case 'TEST-06'
                s = {'name', '0.9zeta 3.5w_n'; ...; %
                    'zeta', .9;... %  damping ratio
                    'omega_n',3.5; ...%  natural freq. (rad/s)
                    'omega_co', .5; ...%  cross-over frequency (rad/s)
                    'num', 3.5^2; ...%  numerator of transfer funct.
                    'den', [1 2*.9*3.5 3.5^2]; ...%  den. transfer funct.
                    };
                otherwise
                    exception = MException('VerifyInput:NotUsed', ...
                        ['Input ' mfield ' is not valid  for '...
                        'a stored set of properties']);
                    throw(exception);
            end
        end
    end
end
```

### I.8.4 The Routh Approximation Method

The code contained here is migrated to MATLAB code and some refactoring has been accomplished to aid the migration. The code comes from [69, 70] and the examples from the papers have been used to verify the refactoring has been accomplished. The sources describe a metric for measuring the energy transferred by the Approximant polynomial. The metric is tabulated for each material and order in Table I.4. Bode plots of the transfer function used for the RA method gain calculation and their approximants are shown in Figures I.4 and I.5 for PET and 24 inch wide Tyvek, respectively.

Listing I.18: The Routh Approximation Method Code

```
function [reduced_num,reduced_den] = RouthApprox(num,den,ord,table)
%funciton for reducing a characteristic polinomial via the Hutton and
%Rabins method in "Simplification of High-Order Mechanical Systems Using
% the Routh Approximation"
%
%   Syntax: [reduced_num,reduced_den] = RouthApprox(num,den,ord,[table])
%   where
%       num             : Original transfer function numerator coefficients
%       den             : Original transfer function denominator coeffs.
%       ord             : order of desired transfer function approximation
%       table           : 0 or 1, optional gives a table of power based on
```

Table I.4: Hutton's Impulse Energy Metric for Comparing an Approximant to the Original Polynomial from [70]

| Order | Amplitude |
|---|---|
| Narrow Tyvek | |
| 1 | 0.85343 |
| 2 | 0.96784 |
| 3 | 1.00000 |
| Wide Tyvek | |
| 1 | 0.85343 |
| 2 | 0.96784 |
| 3 | 1.00000 |
| PET | |
| 1 | 0.85343 |
| 2 | 0.96784 |
| 3 | 1.00000 |

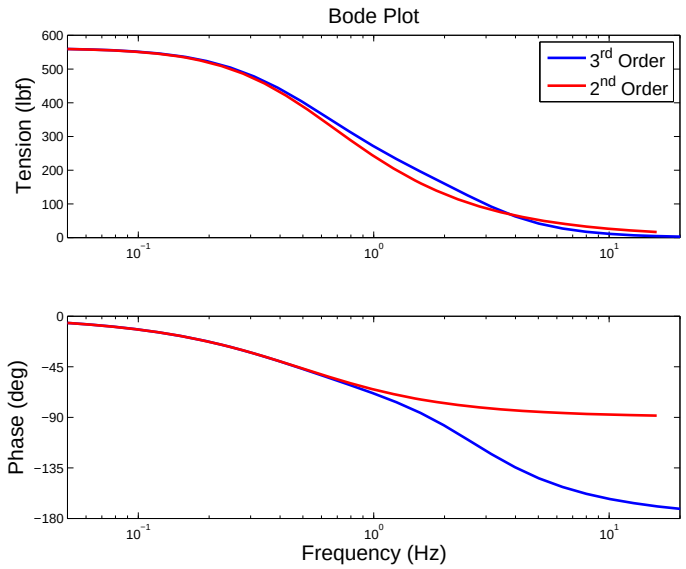



Figure I.4: Bode Plot of Model Created for Control of Tension in PET Material and Its Approximant

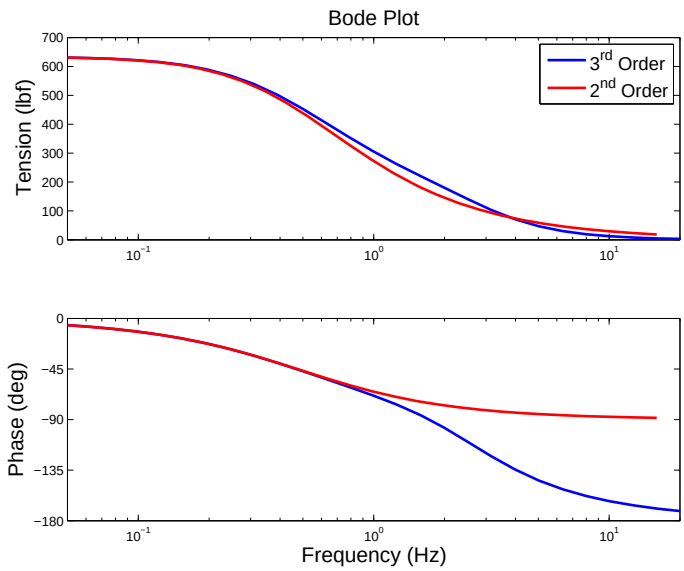



Figure I.5: Bode Plot of Model Created for Control of Tension in Wide Tyvek Material and Its Approximant

```
%                           order
% reduced_num      : New numerator coefficients
% reduced_den      : New denominator coefficients of reduced
%                           polynomial
%
if nargin < 1
        den = [2 36 204 360 240];
        num =[ 28 496 1800 2400];
end
% ord = 2;
if nargin < 3
    ord = 2;
end
if nargin < 4
    table = 0;
end
k = ord+1;

n = size(den);
m = size(num);
if n(1)>n(2)
    %make den a row vector
    den = den';
    n = n(1);
else
    n = n(2);
end
a_n = zeros(n,n);
alpha = zeros(n,1);
if m(1)>m(2)
```

```
    %make den a row vector
43      num = num';
        m = m(1);
45  else
        m = m(2);
47  end
    b_n = zeros(n,n);
49  beta = zeros(n,1);

51  if mod(n,2)==0
        for i = 0:n/2-1
53          a_n(1,i+1) = den(n-i*2);
            a_n(2,i+1) = den(n-i*2-1);
55      end
    else
57      %odd n
        for i = 0:(n-1)/2-1
59          a_n(1,i+1) = den(n-i*2);
            a_n(2,i+1) = den(n-i*2-1);
61      end
        a_n(1,i+2) = den(n-i*2-2);
63  end
        i=2;
65      while i<=n
            alpha(i-1) = a_n(i-1,1)/a_n(i,1);
67          j = 1;
            while i+j<=n
69              j = j+1;
                a_n(i+1,j-1) = a_n(i-1,j) - alpha(i-1)*a_n(i,j);
71          end
            i=i+1;
73      end
    %-------------------------------------------------------------------------
75  %       N U M E R A T O R   V A L U E S
    %-------------------------------------------------------------------------
77  if mod(m,2)==0
        for i = 0:m/2-1
79          b_n(2,i+1) = num(m-i*2);
            b_n(3,i+1) = num(m-i*2-1);
81      end
    else
83      %odd m
        if m == 1
85          b_n(2,1) = num(1);
        else
87          for i = 0:(m-1)/2-1
                b_n(2,i+1) = num(m-i*2);
89              b_n(3,i+1) = num(m-i*2-1);
            end
```

```
91          b_n(2,i+2) = num(m-i*2-2);
        end
93  end
    for i = 2:n
95      beta(i-1) = b_n(i,1)/a_n(i,1);
        if mod(n,2)==0
97          if m == 1

99          else
                for j = 2:n-i+2
101                 b_n(i+2,j-1) = b_n(i,j)-beta(i-1)*a_n(i,j);
                end
103         end
        else
105         for j = 2:(n-1)-i+2
                b_n(i+2,j-1) = b_n(i,j)-beta(i-1)*a_n(i,j);
107         end
        end
109 end
    %-----------------------------------------------------------------------
111 %       P U T   T O G E T H E R   T H E   k t h   O R D E R   T F
    %-----------------------------------------------------------------------
113 if max(a_n(:,1)<0)>0
        %only execute for stable polynomials
115     error(['Routh Approximation routine: '...
            'Denominator is not a stable characteristic polynomial.'])
117     reduced_num = 0;
        reduced_den = 0;
119 else
    if k ==2
121     reduced_den = [1 alpha(k-1)];
        reduced_num = [beta(k-1)];
123 else
        i=2;
125     flag = 1;
        a_new = zeros(k,k);
127     b_new = zeros(k,k);
        a_new(k+[-1 0],k+[-1 0]) = [alpha(1) 1; 0 1];
129     b_new(k-1,k+[-1 0]) = [0 beta(1)];
        while flag
131         store = conv(alpha(i)*[1 0],a_new(k-i+1,:));%+a_new(k-i+2,:);
            a_new(k-i,:) = store(2:end)+a_new(k-i+2,:);
133         if i<k-1
                i = i+1;
135         else
                flag = 0;
137         end
        end
139     flag = 1; %reset for numerator
```

```
    i=2;
    % need to check for what order, p, the original numerator was; if it
    % was scalar, can't add zeros. But, if p >1, calc numerator up to p if
    % p ≤ k-1
    while flag
        store = conv([alpha(i) 0],b_new(k-i+1,:));
        b_new(k-i,:) =  store(2:end)+b_new(k-i+2,:)+[zeros(1,k-1) beta(i)];
        if i<k-1 && i<m
            i = i+1;
        else
            flag = 0;
        end
    end
%-------------------------------------------------------------------------
%       P U T   T F   I N   C O R R E C T   O R D E R
%-------------------------------------------------------------------------
if k>m
    if m == 1
        reduced_num = b_new(1,k:-1:2);
    else
    reduced_num = b_new(k-m,k:-1:k-m+1);
    end
else
    reduced_num = b_new(1,k:-1:2);
end

    reduced_den = a_new(1,k:-1:1);
     i = 1;
    while reduced_num(i) == 0
        i = i+1;
    end
    reduced_num = reduced_num(i:end);
%
end
if table
    % create a power value of RMS amplitude transmitted through transfer
    % function to see how much power is captured by different orders of
    % approximation.  See the sited paper.
    sigma = zeros(1,n);
    sigma(1) = beta(1)^2/2/alpha(1);
    i = 2;
    while i<n
        sigma(i) = sigma(i-1) + beta(i)^2/2/alpha(i);
        i = i+1;
    end
    sigma = sqrt(sigma);
    fprintf(1,'\n')
    fprintf(1,'Order\tAmp.\n')
    for i = 1:n-1
```

```
189             fprintf(1,'%2d\t%1.5f\n',i,sigma(i)/sigma(n-1))
        end
191 end
    end
```

### I.8.5 The Ducotey-Good Traction Class Definition

The `DucoteyGood` object calculates the air film height between the roll and the web. That height is translated to a traction coefficient. As the air film height increases the traction coefficient is driven to zero. The math contained in this function and most of the sample data come from [38–40, 42]. The migration to MATLAB code was necessary to allow use with other models.

Listing I.19: DucoteyGood Class Definition Code

```
classdef DucoteyGood < handle
2 %  The DucoteyGood class is a class definintion to create a self-contained
  %  set of parameters and calculations to determine the coefficient of
4 %  traction between a roller and a web based on the assumption of an air
  %  film separating the two.  Ducotey and Good produced several papers:
6 %    1 ---- "The Importance of Traction in Web Handling", Journal of
  %           Tribology. Vol. 117, October 1995, pg. 679--684.
8 %    2 ---- "The Effect of Web Permeability and Side Leakage on the Air
  %           Film Height Between a Roller and Web." Journal of Tribology.
10 %          Vol. 120, July 1998, pg. 559--565.
  %    3 ---- "Predicting Traction in Web Handling." Transactions of the
12 %          ASME.  Vol. 121, July 1999, pg. 618--624.
  %    4 ---- "A Numerical Algorithm for Determining the Traction Between a
14 %          Web and a Circumferentially Grooved Roller." Transactions of
  %          the ASME. Vol. 122, July 2000, pgs. 578--584.
16 %
  %  This code is derived from #3.
18 %
  %   Syntax:  DG = DucoteyGood(webprops,rollerprops);
20 %      where
  %   webprops    -  9x2 cell array of 'property name' 'property value' pairs
22 %                  that contains (in base units, English units default):
  %                  {'tension',10;...  %lbf
24 %                   'density', 1.0740000E-0002/32.174*144*12;... %slug/ft^3
  %                   'speed', 400/60;... %ft/s
26 %                   'alpha',0;... %permeability ft^3/s/(area-pressure drop)
  %                   'E',66e3*144;... %Young's modulus in lbf/ft^2
28 %                   'width',6.07/12;... %of web in ft
  %                   'rmsroughness',3.378e-6/0.0254/12;... %surface
30 %                   ... %        roughness in ft (rms)
  %                   'roughness',2.667e-6/0.0254/12 ... % surface roughness
32 %                   ... %        in ft (direct measure)
```

```
%                   'thickness',.00536/12 ...
%                  }
%                   The parsing mechanism will error out if you do not use
%                   the names given above for the web properties.
%   rollerprops     -  5x2 cell array of 'property name' 'property value'
%                    pairs that contains:
%                    {'radius',1.5/12;... % ft of roller
%                     'speed',400/60;... % ft/s of roller
%                     'beta',90*pi/180;... % radians wrap angle
%                     'rmsroughness',11.557e-6/0.0254/12;... % rms surface
%                     ...  %      roughness in ft
%                     'roughness',9.119e-6/0.0254/12; ... % surface
%                     ...  %      roughness in ft
%                    }
%                    The parsing mechanism will error out if you do not use
%                    the names given above for the roller properties.
% This initiallized the class instance.  To obtain a prediction of the
% coefficient of traction, use:
%     mu_T = DG.predict(web_speed,roll_speed,Tension,mu_s)
%      where
%     web_speed       -   use base units of (default) ft/s or m/s
%     roll_speed      -   use base untis of (default) ft/s or m/s
%     Tension         -   (default) lbf or N (this term will be converted
%                         to a PLI measure by dividing the Tension by the
%                         width of the web.
%     mu_s            -   Coefficient of static friction between web and
%                         roller.  It has no units.
%
%
% The class also has DG.get('field_name') and DG.set('Field_name',Val)
% methods for individually manipulating the parameters.  These two methods
% are the only way to access certain private variables of the class.
%
%
  % class properties
  properties (Access = public)
    % these values are asked by the user in the constructor
    R = 1.5/12; %ft radius
    T =  10; %lbf tension
    m = .000001; %slug mass of web on roller
    V_w =  400/60; %ft/s speed of web
    V_r =  399.98/60; %ft/s speed of roller
    %etta = 18.3e-6; %N-s/m^2 dynamic viscosity of air
    etta = 0.000000375937816; %lbf*s/ft^2 dynamic viscosity of air
    rho = 1.0740000E-0002/32.174*144*12 ; %density of web in slug/ft^3
    alpha =  0; %(ft^3/s)/(A_0-P) permeability of web
    E = 66e3*144; %lbf/ft^2 young's modulus
    beta = 90*pi/180; %rad wrap angle
    w = 6.07/12; %ft web width
```

```
82      Rq_w = 3.378e-6/0.0254/12; %ft web roughness, rms
        Rq_r = 11.557e-6/0.0254/12; %ft ranges in paper from 0.229-11.557e-6m
84                                  %roller roughness
        thickness = 0.00536/12; % ft web thickness
86      Ra_w = 2.667e-6/0.0254/12; %ft web ave. roughness
        Ra_r = 9.119e-6/0.0254/12; %ft roller ave. roughness
88      SideLeakage = 1; %switch for assuming side leakage occurs 1=true,0=false
        Tol = 1e-6; % error tolerance in calculated epsilon_b
90    end
      % hidden variables
92    properties (Access = protected)
          k = 0.643; %constant from the theoretical analysis
94        epsilon = 0;      %
          epsilon_a = 0;
96        epsilon_b = 0;
          h_0 = 0;
98        h_sL = 0;
          delta_h_sL = 0;
100       U = 0;%V_w + V_r; %ft/s sum speed of roller and web
          Rq_c = 0;%sqrt(obj.Rq_w^2 + obj.Rq_r^2); %rms roughness of interface
102       Ra_ave = 0;%(obj.Ra_w + obj.Ra_r)/2; %ft average roughness of interface
          g = 32.174; %ft/s^2 acceleration of gravity
104       chi = 0; %R/w;
          psi = 0; %etta*alpha/R;
106       Omega = 0; %Ra_ave/R;
          xi = 0; %Rq_c/R;
108       units = 1; %unit switch 1=English, 0=SI
      end
110   % class methods
      methods
112
        function obj = DucoteyGood(web,roller)
114       %  Constructor for DucoteyGood.
          %
116       %  Inputs:
          %    web       - 9 x 2 cell array of web data
118       %    roller    - 5 x 2 cell array of roller data
          %
120       %  Outputs:
          %    -none
122       try
          % nesting structure to set new values
124       if nargin >=1
            if ~isempty(web)
126         if ischar(web)
                s = storedProps(obj,web);
128             web = s;
            end
130           for i = 1:length(web)
```

```
            switch (web{i,1})
                case 'tension'
                    obj.T = web{i,2};
                case 'density'
                    obj.rho = web{i,2};
                case 'speed'
                    obj.V_w = web{i,2};
                case 'alpha'
                    obj.alpha = web{i,2};
                case 'E'
                    obj.E = web{i,2};
                case 'width'
                    obj.w = web{i,2};
                case 'rmsroughness'
                    obj.Rq_w = web{i,2};
                case 'roughness'
                    obj.Ra_w = web{i,2};
                case 'thickness'
                    obj.thickness = web{i,2};
                otherwise
                   exception = MException('VerifyInput:NotUsed', ...
                       ['Input ' web{i,1} ' is not a valid name']);
                   throw(exception);
            end
        end
    end
    if nargin >=2
        if ~isempty(roller)
            if ischar(roller)
                s = storedProps(obj,roller);
                roller = s;
            end
          for i = 1:length(roller)
            switch (roller{i,1})
                case 'radius'
                    obj.R = roller{i,2};
                case 'beta'
                    obj.beta = roller{i,2};
                case 'speed'
                    obj.V_r = roller{i,2};
                case 'rmsroughness'
                    obj.Rq_r = roller{i,2};
                case 'roughness'
                    obj.Ra_r = roller{i,2};
                otherwise
                   exception = MException('VerifyInput:NotUsed', ...
                       ['Input ' roller{i,1} ' is not a valid name']);
                   throw(exception);
            end
```

```
            end
            %calculate mass of web on roller...
            %vol*density...
            obj.m = obj.rho*obj.thickness; %convert density to mass/area
            obj.T = obj.T/obj.w;   %convert Tension to tension/width
            obj.U = obj.V_w + obj.V_r; % sum speed
            obj.Rq_c = sqrt(obj.Rq_w^2 + obj.Rq_r^2);%rms rough. of interface
            obj.Ra_ave = (obj.Ra_w + obj.Ra_r)/2; %roughness of interface
            obj.chi = obj.R/obj.w;
            obj.psi = obj.etta*obj.alpha/obj.R;
            obj.Omega = obj.Ra_ave/obj.R;
            obj.xi = obj.Rq_c/obj.R;
            processLimits(obj);
          end
        end
      end
      catch ME
        disp([ME.message '!'])
        err = MException('VerifyOutput:OutOfBounds', ...
                'Exiting due to incorrect inputs.');
        throw(err);
      end
    end
    function process(obj)
        %Calculate internal variables
        obj.epsilon = 6*obj.etta*obj.U/(obj.T);
        obj.h_0 = obj.k*obj.R*obj.epsilon^(2/3);
        obj.h_sL = obj.R/(sqrt(12*obj.beta/obj.epsilon*(obj.R/obj.w)^2 + ...
            1/(obj.k*obj.epsilon^(2/3))^2));
        obj.delta_h_sL = obj.h_0 - obj.h_sL;
        %epsilon_b = fsolve(@epb,epsilon);
    end
    function [mu_T,muTvar] = predict(obj,web_speed,roll_speed,Tension,mu_s)
        % predicts coefficient of traction given web and roller speed and
        % tension (tension as a unit of force, not PLI), and coefficient of
        % static friction
        %
        obj.U = web_speed+roll_speed;
        obj.V_w = web_speed;
        obj.V_r = roll_speed;
        obj.T = Tension/obj.w;
        process(obj)
        if obj.epsilon <= obj.epsilon_a
            mu_T = mu_s;
        elseif obj.epsilon <= obj.epsilon_b
            mu_T = mu_s/(obj.epsilon_b - obj.epsilon_a)*(obj.epsilon_b -...
                obj.epsilon);
        else
            mu_T = 0;
```

```
        end
        if nargout == 2
            %may work on this, but not now
            muTvar = 0;
        end
    end
    function processUnits(obj)
        % changes internal variables to requested units system
        % specifically: it converts air viscosity from lbf-s/ft^2 to
        % N-s/m^2 and back again.  It also sets the acceleration of gravity
        % to the proper values, though I don't think that is actually used
        % in the code...
        if obj.units == 1
            obj.g = 32.174; %ft/s^2 acceleration of gravity
            if obj.etta == 18.3e-6
                %if etta is the default value
                obj.etta = 0.000000375937816; %lbf*s/ft^2 dynamic
                                              %viscosity of air
            else
                obj.etta = obj.etta/47.880259; %ft-s/ft^2/(N-s/m^2)
            end
        else
            obj.g = 9.81; %m/s^2 acceleration of gravity
            if obj.etta == 0.000000375937816
                %if etta is the default value
                obj.etta = 18.3e-6; %N-s/m^2 dynamic viscosity of air
            else
                obj.etta = obj.etta*47.880259; %N-s/m^2/(ft-s/ft^2)
            end
        end
        obj.psi = obj.etta*obj.alpha/obj.R;
    end
    function processLimits(obj)
        %updates epsilon_a and epsilon_b if beta or any of the other
        %parameters change.  THIS HAS TO BE CALLED.
        %
        obj.epsilon_a = (obj.Omega/obj.k)^(3/2); % beginning point
                                                 % of air entrainment
        eps_tol = 1;
        eguess = [1e-9 1e-3];%[.005 1e4]*obj.epsilon;
        e_check(1) = -12*obj.psi/eguess(1)*obj.beta+(sqrt(12*obj.beta/...
            eguess(1)*obj.chi^2+1/(obj.k*eguess(1)^(2/3))^2))^(-1)...
            -3*obj.xi;
        e_check(2) = -12*obj.psi/eguess(2)*obj.beta+(sqrt(12*obj.beta/...
            eguess(2)*obj.chi^2+1/(obj.k*eguess(2)^(2/3))^2))^(-1)...
            -3*obj.xi;
        if e_check(1)/e_check(2) > 0
            disp(['Initial intervale of epsilon_b search does not '...
                'contain a zero crossing' '!'])
```

```
278             err = MException('VerifyOutput:OutOfBounds', ...
                    'No Zero crossing in interval.');
280             throw(err);
            end
282         if obj.SideLeakage
                I_max = 1e4;
284             I = 0;
                eg_prev=eguess(2);
286             while eps_tol > obj.Tol && I<I_max
                    e_c1 = e_check*[0 -1; 1 0];
288                 eg = eguess*e_c1'/(e_c1*[1 1]');
                    e_ch = -12*obj.psi/(eg)*obj.beta + ...
290                     (sqrt(12*obj.beta/(eg)*obj.chi^2 + ...
                        1/(obj.k*(eg)^(2/3))^2))^(-1) - 3*obj.xi;
292                 if e_ch/e_check(1)<0 && abs(eg/eguess(1))>100
                        eguess(1) = eguess(1)*10;
294                     e_check(1) = -12*obj.psi/(eguess(1))*obj.beta + ...
                              (sqrt(12*obj.beta/(eguess(1))*obj.chi^2 + ...
296                           1/(obj.k*(eguess(1))^(2/3))^2))^(-1) - 3*obj.xi;
                    elseif e_ch/e_check(1)<0
298                     e_check(2) = e_ch;
                        %eguess(2) = eguess*[1 1]'*.5;
300                     eguess(2) = eg;
                        eps_tol = abs(eg_prev - eguess(2));
302                     eg_prev = eguess(2);
                    else
304                     e_check(1) = e_ch;
                        %eguess(1) = eguess*[1 1]'*.5;
306                     eguess(1) = eg;
                        eps_tol = abs(eg_prev - eguess(1));
308                     eg_prev = eguess(1);
                    end
310                 I = I+1;
                end
312             if I == I_max
                    exception = MException(...
314                     'ProcessLimits:IterExceeded', ...
                        'Sideleakage solver hit limit.');
316                 throw(exception);
                    disp(['DucoteyGood -- Epsilon_b iteration hit iteration'...
318                     'limit'])
                    disp(['beta ' num2str(obj.beta) 'V_w ' num2str(obj.V_w)...
320                     'e_ch ' num2str(e_ch)])
                end
322             obj.epsilon_b = eg_prev;
            else
324             obj.epsilon_b = ((3*obj.xi)/obj.k)^(3/2);
            end
326     end
```

```
function [a] = findCrossing(obj,mfield)
328       % finds the value of mfield that makes epsilon = epsilon_a
          % and epsilon_b for current object parameters
330       %  mfield = {speed,tension}
          %
332       switch mfield
              case {'speed','Speed','V'}
334
                  a = [obj.epsilon_a*obj.T/(6*obj.etta)...
336                       obj.epsilon_b*obj.T/(6*obj.etta)]/2;
              case {'tension','Tension'}
338
                  a = [obj.epsilon_a/(6*obj.etta*obj.U) ...
340                       obj.epsilon_b/(6*obj.etta*obj.U)];
              otherwise
342               disp(['Value of ' mfield ' is not {speed,tension}'])
                  err = MException('VerifyInput:OutOfBounds', ...
344                       'Not an option.');
                  throw(err);
346       end
      end
348   function mval = get(obj,mfield)
      %
350   %  Get a requested member variable.
      %
352   switch(mfield)
        case {'R','radius'}
354       mval = obj.R;
        case {'V_r','rollerspeed'}
356       mval = obj.V_r;
        case {'webspeed','V_w'}
358       mval = obj.V_w;
        case {'U'}
360       mval = obj.U;
        case {'h_0'}
362       mval = obj.h_0;
        case {'h_sL'}
364       mval = obj.h_sL;
        case {'epsilon'}
366       mval = obj.epsilon;
        case {'epsilon_a','epsilon_b'}
368       mval = [obj.epsilon_a obj.epsilon_b];
        case{'rho'}
370       mval = obj.rho;
        case{'alpha','permeability'}
372       mval = obj.alpha;
        case{'wrapangle','beta'}
374       mval = obj.beta;
        case{'E','modulus'}
```

```
376             mval = obj.E;
            case{'w','width'}
378             mval = obj.w;
            case{'thickness'}
380             mval = obj.thickness;
            case{'SideLeakage'}
382             mval = obj.SideLeakage;
            case{'roughness','Rq_w','Rq_r','Ra_w','Ra_r','Rq_c','Ra_ave'}
384             mval = {'Rq_w',obj.Rq_w;'Rq_r',obj.Rq_r;'Rq_c',obj.Rq_c;...
                    'Ra_w',obj.Ra_w;'Ra_r',obj.Ra_r;'Ra_ave',obj.Ra_ave};
386         case {'Tol','tolerance'}
              mval = obj.Tol;
388         case {'Units','units'}
              mval = obj.units;
390               if obj.units == 1
                      disp('English Units Used')
392               else
                      disp('SI Units Used')
394               end
        end
396     end
        function set(obj,mfield,val)
398     %
        %  Set a requested member variable.
400     %
        switch(mfield)
402       case {'R','radius'}
            obj.R = val;
404       case {'V_r','rollerspeed'}
            obj.V_r = val;
406       case {'webspeed','V_w'}
            obj.V_w = val;
408       case {'U'}
            obj.U = val;
410       case{'rho'}
            obj.rho = val;
412       case{'thickness'}
            obj.thickness = val;
414       case{'alpha','permeability'}
            obj.alpha = val;
416         obj.psi = obj.etta*obj.alpha/obj.R;
            processLimits(obj);
418       case{'wrapangle','beta'}
            obj.beta = val;
420         processLimits(obj);
          case{'viscosity','etta'}
422         obj.etta = val;
            obj.psi = obj.etta*obj.alpha/obj.R;
424         processLimits(obj);
```

```
        case{'E','modulus'}
426       obj.E = val;
        case{'w','width'}
428       obj.w = val;
        case{'Ra_r','roller average roughness'}
430       obj.Ra_r = val;
          obj.Ra_ave = (obj.Ra_w + obj.Ra_r)/2; %roughness of interface
432       obj.Omega = obj.Ra_ave/obj.R;
          processLimits(obj);
434     case{'Ra_w','web average roughness'}
          obj.Ra_w = val;
436       obj.Ra_ave = (obj.Ra_w + obj.Ra_r)/2; %roughness of interface
          obj.Omega = obj.Ra_ave/obj.R;
438       processLimits(obj);
        case{'Rq_w','web rms roughness'}
440       obj.Rq_w = val;
          obj.Rq_c = sqrt(obj.Rq_w^2 + obj.Rq_r^2); %rms rough. of interface
442       obj.xi = obj.Rq_c/obj.R;
          processLimits(obj);
444     case{'Rq_r','roller rms roughness'}
          obj.Rq_r = val;
446       obj.Rq_c = sqrt(obj.Rq_w^2 + obj.Rq_r^2);%rms rough. of interface
          obj.xi = obj.Rq_c/obj.R;
448       processLimits(obj);
        case {'Tol','tolerance'}
450       obj.Tol = val;
          case {'Units','metric','SI','English','British','units'}
452           if strcmp(mfield, 'Units') || strcmp(mfield,'units')
                  if ismember(val,[0,1])
454               %units = 1 means British units, 0 means SI
                     obj.units = val;
456               else
                      disp(['Value of ' num2str(val) ' is not {0,1}'])
458                   err = MException('VerifyInput:OutOfBounds', ...
                          'Not a logical value.');
460                   throw(err);
                  end
462           elseif strcmp(mfield,'metric') || strcmp(mfield,'SI')
                  obj.units = 0;
464           else
                  obj.units = 1;
466           end
              processUnits(obj);
468           processLimits(obj);
    end
470 end
    function [prp] = storedProps(obj,mfield)
472     % stored web and roller property sets
        switch (mfield)
```

```
474                 case 'DucoteyRoller1'
                        %from paper #3 Roller 1
476                     prp = {'radius',.127/2;'speed',25.4;'beta',92*pi/180;...
                            'rmsroughness',.686e-6;'roughness',.432e-6};
478                 case 'DucoteyRoller4'
                        %from paper #3 Roller 4
480                     prp = {'radius',.127/2;'speed',25.4;'beta',92*pi/180;...
                            'rmsroughness',6.731e-6;'roughness',5.334e-6};
482                 case 'DucoteyRoller5'
                        %from paper #3 Roller 5
484                     prp = {'radius',.127/2;'speed',25.4;'beta',92*pi/180;...
                            'rmsroughness',6.045e-6;'roughness',4.699e-6};
486                 case 'DucoteyRoller6'
                        %from paper #3 Roller 6
488                     prp = {'radius',.127/2;'speed',25.4;'beta',92*pi/180;...
                            'rmsroughness',11.557e-6;'roughness',9.119e-6};
490                 case 'DucoteyWebNP-1'
                        %from paper #3 Web NP-1
492                     prp = {'tension',2.87*15.24;...
                             'density', 678.2;...
494                          'speed', 25.4;...
                             'alpha',0.00000273; ...
496                          'E',66e3*144;...
                             'width',.1524;...
498                          'thickness',.0000711;...
                             'rmsroughness',4.496e-6;...
500                          'roughness',3.531e-6 ...
                            };
502                 case 'DucoteyWebNP-2'
                        %from paper #3 Web NP-2
504                     prp = {'tension',2.87*15.24;...
                             'density', 661.5;...
506                          'speed', 25.4;...
                             'alpha',0.00000424; ...
508                          'E',66e3*144;...
                             'width',.1524;...
510                          'thickness',.0000762;...
                             'rmsroughness',4.496e-6;...
512                          'roughness',3.531e-6 ...
                            };
514                 case 'DucoteyWebNP-3'
                        %from paper #3 Web NP-3
516                     prp = {'tension',2.87*15.24;...
                             'density', 697.5;...
518                          'speed', 25.4;...
                             'alpha',0.00000125; ...
520                          'E',66e3*144;...
                             'width',.1524;...
522                          'thickness',.0000737;...
```

```
                'rmsroughness',3.376e-6;...
524             'roughness',2.667e-6 ...
                };
526         case 'DucoteyWebPF-2'
                %from paper #3 Web PF-2
528             prp = {'tension',2.87*15.24;...
                'density', 27.68;...
530             'speed', 25.4;...
                'alpha',0.0; ...
532             'E',66e3*144;...
                'width',.1524;...
534             'thickness',.000508;...
                'rmsroughness',.737e-6;...
536             'roughness',.432e-6 ...
                };
538         case 'DucoteyWebPF-3'
                %from paper #3 Web PF-3
540             prp = {'tension',2.87*15.24;...
                'density', 30.45;...
542             'speed', 25.4;...
                'alpha',0.0; ...
544             'E',66e3*144;...
                'width',.3048;...
546             'thickness',.000559;...
                'rmsroughness',.406e-6;...
548             'roughness',.229e-6 ...
                };
550         case 'ReishWebTyv-1'
                % Tyvek material used on Euclid line
552             prp = {'tension',10;...  %lbf
                'density', 1.0740000E-0002/32.174*144*12;... %slug/ft^3
554             'speed', 400/60;... %ft/s
                'alpha',0;... %permeability ft^3/s/(area-pressure drop)
556             'E',66e3*144;... %Young's modulus in lbf/ft^2
                'width',6.07/12;... %of web in ft
558             'rmsroughness',3.378e-6/0.0254/12;... %surface
                ... %        roughness in ft (rms)
560             'roughness',2.667e-6/0.0254/12; ... % surface roughness
                ... %        in ft (direct measure)
562             'thickness',.00536/12 ...
                };
564         case 'ReishWebTyv-2'
                % Tyvek material used on Euclid line simulated in WTS
566             prp = {'tension',10;...  %lbf
                'density', 1.0740000E-0002/32.174*144*12;... %slug/ft^3
568             'speed', 400/60;... %ft/s
                'alpha',0;... %permeability ft^3/s/(area-pressure drop)
570             'E',46.3e3*144;... %Young's modulus in lbf/ft^2
                'width',6.077/12;... %of web in ft
```

```
572                 'rmsroughness',(15e-6 + 0.003e-6)/12;... %surface
                    ... %        roughness in ft (rms)
574                 'roughness',15e-6/12; ... % surface roughness
                    ... %        in ft (direct measure)
576                 'thickness',.00536/12 ...
                    };
578             case 'ReishRoller1'
                    %Steel idler from Euclid line
580                 prp = {'radius',1.5/12;... % ft of roller
                        'speed',400/60;... % ft/s of roller
582                     'beta',90*pi/180;... % radians wrap angle
                        'rmsroughness',69.3e-6/12;... % rms surface
584                     ...  %      roughness in ft
                        'roughness',63e-6/12 ... % surface
586                     ...  %      roughness in ft
                        };
588             case 'ReishRoller2'
                    %Aluminum idler from High-Speed Web line
590                 prp = {'radius',2/12;... % ft of roller
                        'speed',400/60;... % ft/s of roller
592                     'beta',90*pi/180;... % radians wrap angle
                        'rmsroughness',35.2e-6/12;... % rms surface
594                     ...  %      roughness in ft
                        'roughness',32e-6/12 ... % surface
596                     ...  %      roughness in ft
                        };
598             case 'ReishWebTyv-3'
                    % Tyvek material used on sent to DuPont for material
                    % characterization. dynamic COF = 0.1236, static COF =
                    % 0.2396
602                 prp = {'tension',10;...  %lbf
                        'density', 1.0740000E-0002/32.174*144*12;... %slug/ft^3
604                     'speed', 400/60;... %ft/s
                        'alpha',8.63e-4;... %perm. ft^3/s/(ft^2-lbf/ft^2)
606                     'E',78389*144;... %Young's modulus in lbf/ft^2
                        'width',6.0/12;... %of web in ft
608                     'rmsroughness',(137.5e-6)/12;... %surface
                        ... %        roughness in ft (rms)
610                     'roughness',125e-6/12; ... % surface roughness
                        ... %        in ft (direct measure)
612                     'thickness',.00782/12 ...
                        };
614             case 'ReishWebPET-1'
                    % PET material used in Wrinkling study on HSWL
616                 % COF = 0.2 - 0.4
                    prp = {'tension',10;...  %lbf
618                     'density', 2.522;... %slug/ft^3
                        'speed', 400/60;... %ft/s
620                     'alpha',0;... %permeability ft^3/s/(ft^2-lbf/ft^2)
```

```
                    'E',625e3*144;... %Young's modulus in lbf/ft^2
622                 'width',6.0/12;... %of web in ft
                    'rmsroughness',(69e-6 + 0.00e-6)/12;... %surface
624                 ... %       roughness in ft (rms)
                    'roughness',63e-6/12; ... % surface roughness
626                 ... %       in ft (direct measure)
                    'thickness',.002/12 ...
628                 };
            otherwise
630             disp(['The name of ' mfield ' is not one of the options.'])
                err = MException('VerifyInput:OutOfBounds', ...
632                 'Not a stored web or roller type.');
                throw(err);
634         end
        end
636     function visualizeEpsB(obj)
            % plots function of epsilon_b over the field of the search
638         sp = logspace(-8,-3,20);
            e_ch = zeros(1,length(sp));
640         for i =1:length(sp)
                eg = sp(i);
642             e_ch(i) = -12*obj.psi/(eg)*obj.beta + ...
                    (sqrt(12*obj.beta/(eg)*obj.chi^2 + ...
644                 1/(obj.k*(eg)^(2/3))^2))^(-1) - 3*obj.xi;
            end
646         figure()
            semilogx(sp,e_ch,'d--')
648         xlabel('\epsilon_b')
            ylabel('function value')
650     end
        function visualizeSeal(obj)
652         % plots function of xi from Blevins pg. 507 to test if a
            % labyrinth seal is a better explanation of the situation.  xi>10
654         % indicates a labyrinth seal
            %  the 432.7e-6ft for L1 comes from Table 2 of Ducotey Good,
656         %  "predicting traction..." for 76 peaks per cm for roller 7
            sp = logspace(-8,3,40);
658         exi = zeros(1,length(sp));
            for i =1:length(sp)
660             t = obj.R*obj.beta/sp(i);
                hdel = obj.h_0-obj.k*obj.R*obj.epsilon_b^(2/3);
662             U=hdel/2*obj.R*obj.beta*obj.w/(hdel*obj.R*obj.beta)/t;
                %U = sp(i);
664             exi(i) = U*obj.h_0^2/obj.etta/431.7e-6;
            end
666         figure()
            loglog(sp,exi,'d--')
668         xlabel('Speed (ft/s)')
            ylabel('\xi')
```

```
670         end
    end
672 end
```

### I.8.6 The Lead-Lag Filter Class Definition

The BRLeadLag object implements a continuous time lead-lag filter from [67, 68]. The function has an internal state that has to be integrated as well to calculate the filtered output. Using the functions adds a state to the system being integrated by ode45 or other solver.

Listing I.20: BRLeadLag Class Definition Code

```
classdef BRLeadLag < handle
2 %BRLeadLag implements a lead-lag compensator with a persistent set of times
%
4 %  This function requires an added state to the state vector.  That would
%  be x in this case.  The output xdot is the derivative to be integrated.
6 %  the output y is the compensated output of the input u.  So, if this is
%  applied to a tension feedback, the load cell signal is u.  The state x
8 %  is one of the overall system states, passed to the function, and the y
%  is the lead-lagged load cell tension value.
10 %
%  params is a cell array:
12 %    params = {'T1',.2; 'T2',4;'up sat', 100;'low sat',-100};
%    where T1 is the lead time constant and T2 is the lag time constant.
14 %    The up sat and low sat are saturation limits for the internal state, x
%    which apply below the calculation.
16 %
%  The sat_true property is set to 0 unless the saturation limit (set in
18 %  the params) is hit, either positive or negative.  The property is
%  accessed by the isSat() function.
20 %
%  The gain can be used to change the magnitude of the filter.
22 properties
    T1;
24     T2;
    up_sat;
26     low_sat;
    sat_true;
28     gain = 1;
end
30 methods
    function obj = BRLeadLag(params)
32         for i = 1:length(params)
            switch (params{i,1})
34                 case 'T1'
                    obj.T1 = params{i,2};
```

```
36                  case 'T2'
                        obj.T2 = params{i,2};
38                  case 'up sat'
                        obj.up_sat = params{i,2};
40                  case 'low sat'
                        obj.low_sat = params{i,2};
42                  otherwise
                        exception = MException('VerifyInput:NotUsed', ...
44                      ['Parmeter ' params{i,1} ' is not a valid parameter']);
                        throw(exception);
46              end
            end
48      end
        function [y,xdot,ydot] = compensate(obj,x,u,udot)
50          if nargin <4
                udot = 0;
52          end
            if x > obj.up_sat
54              x = obj.up_sat;
            end
56          if x < obj.low_sat
                x = obj.low_sat;
58          end
            xdot = 1/obj.T2*(u-x);
60          y = obj.T1/obj.T2*obj.gain*u+(1-obj.T1/obj.T2*obj.gain)*x;
            if nargout == 3
62              ydot = (obj.T1/obj.T2*udot+(1-obj.T1/obj.T2)*xdot)/100;
            end
64      end
        function set(obj,params)
66          for i = 1:length(params)
                switch (params{i,1})
68                  case 'T1'
                        obj.T1 = params{i,2};
70                  case 'T2'
                        obj.T2 = params{i,2};
72                  case 'up sat'
                        obj.up_sat = params{i,2};
74                  case 'low sat'
                        obj.low_sat = params{i,2};
76                  case 'gain'
                        obj.gain = params{i,2};
78                  otherwise
                        exception = MException('VerifyInput:NotUsed', ...
80                      ['Parmeter ' params{i,1} ' is not a valid parameter']);
                        throw(exception);
82              end
            end
84      end
```

```
        function val = get(obj,mfield)
86          switch (mfield)
                case 'T1'
88                  val = obj.T1 ;
                case 'T2'
90                  val = obj.T2;
                case 'up sat'
92                  val = obj.up_sat;
                case 'low sat'
94                  val = obj.low_sat;
                case 'gain'
96                  val = obj.gain;
                otherwise
98                  exception = MException('VerifyInput:NotUsed', ...
                        ['Parmeter ' mfield ' is not a valid parameter']);
100                 throw(exception);
            end
102     end
        function val = isSat(obj)
104         %return the value of sat_true to user
            val = obj.sat_true;
106     end
    end
108 end
```

# APPENDIX J

# SIMULATION CODES

This appendix contains the simulation codes for MATLAB that were used to model the EWL and the HSWL. Simulations included studies of the normal operation for goodness of fit of the gains and studies of slip between web and roller.

## J.1 Euclid Web Line Simulation Code

```
function [ SS, SSder, time, y, Events] = EUWNDmotDer14(tfinal)
%Matlab simulation of Example 4 morphed into Euclid Unwind section
%   WTS Example4 has 5 rollers and 4 spans.  An unwind, an idler, a dancer,
%   an idler, and a wind.
%   In 2, converted dancer to pendulum dancer.
%   In 3, converted span tension calculation to take into account varying
%   span length.
%   In 4, converted line to 5 roller 4 span model of EUclid
%   Euclid Unwind section has 10 rollers and 9 spans.
%   In 5, removed unwind eccentricity and added bump shape to unwind.
%   In 6, brought back all 10 rollers and 9 spans. Included the strain in
%   the span length calculations.
%   In 7, introduced an eccentric idler (#2)
%   In 8, used speed loop gains from Euclid line.  Used 17 rollers and 16
%   spans for moving the hard limit away from where I am watching the web
%   line.  Then I added an integration event to catch when the dancer hit
%   its hard limit of +-20 deg.
%   In 9, Went back to linear dynamics...
%   In 10, switched control to load cell simulation.  Used 12 and 1.2 for
%   gains initially.
%   In 11, Brought back nonlinear span equations for pendulum and clean
%   roll conditions to compare to linear equations for IAB.
%   In 12, Copied version 9 but used gains from version 11 to compare
%   linear model to the nonlinear model.
%   In 13, Copied version 12 and will use this one to do sensitivity
%   studies on the Young's modulus and bearing friction in the rollers and
```

```
%   dancer arm.
%   In 14,  copied version 12 to study eccentricity, bump converted to
%   eccentricity, and roll shape with eccentricity.  The 'case_select'
%   variable decides which case is run.  'case_select'=1 does 0.5in
%   eccentricity. 'case_select' = 2 runs the eccentricity calculated from
%   the bump.  'case_select' = 3 runs the roll shape of the bump.
%

global interror y0 errors_ss h hstep1 z0
%file path to the element property export from WTS.
%file='C:\Users\quietman\Documents\MATLAB\WTS\ECLID_LINE_UNWIND_PEND.Dat';
file='C:\Users\quietman\Documents\MATLAB\WTS\EUCLID_FULL_LINE.Dat';

dancers = 1; %number of dancers in system
%                           filename, number of rolls, number of spans
[r,s,c]=ElementPropReadIn(file,25,24);
Rolls = r{1};
Rolls = Rolls(:,1:17);
Spans = s{1};
Spans = Spans(:,1:16);
g=32.174; %ft/s^2 gravity

if nargin == 0
    tfinal = 60; %sec ending time for simulation
end
%-------------------------------------------------------------------------%
%      C A S E   S E L E C T I O N   I N P U T                            %
%-------------------------------------------------------------------------%
rshape = 1; %unwind roll shape: 1 = bump, 2= smaller bump, 3 =
%               flat, 4 = multi - freq cosine, 5 = perfect circle
case_select = 3;  %1=eccentricity, 2=bump converted into
%               eccentricity, 3 = roll shape

deltaV = [0 0/60]; %ft/s line speed change for each powered roller
Deltaduration = [3 3]; %sec how long it takes to change line speed by deltaV

if case_select == 3
    rshape = 1; %bump
else
    rshape = 5; %smooth roll
end
interror = [0 0];

%bump finding initial values
r_2i = 0.5/12; %ft radius of angle position recorder

hstep = 0.05*tfinal;

clear r s
```

```
rolls = size(Rolls,2); %number of columns in Rolls
spans = size(Spans,2); %number of columns in Spans

Titles=c{1};
dancer=[]; %stores column index for any dancer rollers
driven=[]; %stores column index for any driven rollers
wind = []; %stores column index for any winding rollers
unwind = []; %stores column index for any unwinding rollers
for i=1:rolls
    if strcmp(Titles(i,:),'Dan Roll')
        dancer=[dancer i];
    end
    if strcmp(Titles(i,:),'Drv Roll')
        driven=[driven i];
    end
    if strcmp(Titles(i,:),'Unw Roll')
        unwind=[unwind i];
    end
    if strcmp(Titles(i,:),'Wnd Roll')
        wind=[wind i];
    end
end
%count the number of inputs
inpcount = sum([find(unwind) find(wind) find(driven)]>0);

%       [rolls, spans, 2states for each dancer, 2states for each unwind
%       theta, 2states for span lengths in and out of each dancer, unwind
%       radius, unwind span 1 length, inputs]
cnts = [rolls, spans,dancers*2,2*sum(unwind)+1,dancers*2,1,1,inpcount];

L = Spans(4,:)/12; %feet
width_w = Spans(3,1)/12;%feet
rho_w = Spans(5,:)*12^3/g;
%        lbm/in^3*(12in)^3/ft^3 / (lbm*ft/(s^2*lbf)) = lbf*s^2/ft^4
A = Spans(2,1)/1000/12 * width_w; %ft^2
E = Spans(1,1)*1000*12^2;%lbf/ft^2

%Rollers
Rn = Rolls(3,:)/12/2;% radius in ft for all rollers
Jn = Rolls(13,:); %lbf*ft*s^2
Cmn = Rolls(19,:); % Torque/speed constant of a motor
Kmn = Rolls(18,:); % Torque constant of a motor, lbf*ft/amp
Bfn = Rolls(20,:)*0.0033/0.0002; % lbf*ft*sec, Bearing friction

theta_in=0*Bfn;
theta_on=theta_in;
theta_in(1,2:end-1)=atan2((Rolls(2,1:end-2)-Rolls(2,2:end-1)),...
    (Rolls(1,2:end-1)-Rolls(1,1:end-2)- Rn(2:end-1)-Rn(1:end-2)));
theta_on(1,2:end-1)=atan2((Rolls(2,3:end)-Rolls(2,2:end-1)),...
```

```
    (Rolls(1,3:end)-Rolls(1,2:end-1)- Rn(3:end)-Rn(2:end-1)));
126
Mn = Rolls(10,:)/g; %lbm / (lbm*ft/(s^2*lbf)) = slugs
128 Cn = Rolls(16,:)/1.5*.002; %translational drag. lbf*s/ft
Kn = Rolls(17,:); %spring constant, not listed. lbf/ft
130 fda = Rolls(26,:)/12; %lbf*ft initial torque
Larm = Rolls(12,:)/12; %ft arm length of pendulum
132 Jpn = .209; % second polar moment of pendulum

134 Initvel=Rolls(24,:)/60; %fpm /60=fps
fda(4) = fda(4) + .8*4*5.032/12;  %-3.28 for 400 fpm, -2.4 for 200fpm
136 Rn(unwind) = 7/12;
Rolls(23,10) = 0.1; %inch eccentricity of idler (conversion to ft taken
138 %                    care of later)
if case_select == 3
140     %roll shape inertia and eccentricity
        Rolls(23,unwind) = 0.00089; %inches eccentricity
142     Mn(unwind) = 29.267/g;
        Jn(unwind) = 1.543123;
144 elseif case_select == 2
    %eccentricity from bump CGcalc.m :r=0.00089in angle=57.77697deg
146     Rolls(23,unwind) = 0.00089;
        Mn(unwind) = 29.267/g;
148     Jn(unwind) = 1.543123;
else
150     % eccentricity = 0.5in
        Rolls(23,unwind) = 0.5;
152     Mn(unwind) = 27.783/g;
        Jn(unwind) = 1.54024;
154 end
E = 68.970*1000*12^2;
156 draw=0.99765;%.9945 for 46259,

158 eccent = 13;  %Roller eccentric
Rolls(23,eccent) = 0.02;%inches eccentricity
160 z0= [ones(rolls,1)*Initvel(1); ones(spans,1)*Spans(6,1)]; %temporary..
%             overwritten later in code
162 %% SOLVE STEADY STATE------------------------------------------------------
initten = [10, 12.4, 5.4, Spans(6,end)]; %wound-off tension and final tension
164 f_z = @(z) state_function(0,[Initvel(1);z(2:driven(1)-1); ...
      Initvel(driven(1:2))';...
166     z(driven(2)+1:driven(3)-1); Initvel(driven(2))*draw;
        z(rolls+[1:8]); 2*initten(2) - ...
168     z(rolls+8);z(rolls+[10:spans]);0;0;z(rolls+spans+2*dancers+[1]);...
        Initvel(unwind)/Rn(1);0; L(2:3)'*0;...  % eccentric roller
170     ... %                                  position starts out at 0deg
        z(rolls+spans+2*dancers+[6 7])],...%:rolls+spans+2*dancers)], ...
172    [z(rolls+spans+2*dancers+[8:11])]); %[.45; 8.0e-8]);%
z = fsolve(f_z,[zeros(39,1);0; Rn(1); L(1); zeros(4,1)], ...
```

```
174      optimoptions('fsolve','Algorithm','Levenberg-Marquardt')); %L(2)

176  velocities_ss = [Initvel(1); z(2:rolls)];
     velocities_ss(driven) = [Initvel(driven(1:2)) Initvel(driven(2))*draw];
178  tensions_ss = [z(rolls+[1:8]); 2*initten(2)-z(rolls+8);z(rolls+[10:spans])];
     dancer_ss = z(sum(cnts(1:2))+[1 2]);%[0 0]';
180  motors_ss = z(rolls+spans+2*dancers+8:rolls+spans+2*dancers+inpcount+7);
     %motor_inputs_ss(1) = motor_inputs_ss(1);
182  state_derivative_ss = f_z(z);
     %% plot the steady-state.
184
     %unwind gains
186  Kp(1)= 12;%4;12.391;%5.41;% Taken from WTS
     Ki(1)=.0103;%1.223;%0.00449;% Taken from WTS
188  Kd(1)=0.0102/100;%0.0111;% Taken from WTS
     %unwind speed loop gains
190  Kp(2) = .00203; %16.771738/810; %taken from Euclid
     Ki(2) = .00036; %51.978535/810; %taken from Euclid
192  Kd(2) = 0;
     % driven roller gains
194  Kp(3) = .102; %.0239; % Taken from WTS
     Ki(3) = .0178; %.0140; % Taken from WTS
196  Kd(3) = 0/100; % Taken from WTS

198  % Solve stuff
     if 1
200  z0 = [velocities_ss; tensions_ss; dancer_ss; 0;...
     Initvel(unwind)/Rn(unwind);velocities_ss(2)/Rn(2);...
202  z(sum(cnts(1:3))+[3:4],1); Rn(1);L(1); motors_ss]; %29x1
     figure(1);
204  subplot(2,2,1),plot(velocities_ss*60,'>')
     title('Velocities')
206  xlabel('Roller #')
     ylabel('fpm')
208  subplot(2,2,2),plot(tensions_ss,'o')
     title('Tensions')
210  xlabel('Span #')
     ylabel('lbf')
212  %ylim([9.2 10.2])
     subplot(2,2,3),bar(dancer_ss*180/pi)
214  title('Dancer Position and Velocity')
     xlabel('')
216  ylabel('magnitude')
     subplot(2,2,4),bar(motors_ss)
218  title('Motor Inputs (amps)')
     xlabel('Motor #')
220  ylabel('Magnitude')
     else
222  %z0(1,1) = 6.6667;
```

```
%znot = load('400LinCleanINIT2');
znot = load('400bumpLinInit2');
%znot = load('400NonlinCleanINIT');
%znot = load('400bumpNonInit');
z0 = znot.z0; %[znot.z0(1:22,1); 0; znot.z0(23:29,1)];
%% plot the steady-state.
figure(1);
subplot(2,2,1),plot(z0(1:cnts(1))*60,'>')
title('Velocities')
xlabel('Roller #')
ylabel('fpm')
subplot(2,2,2),plot(z0(cnts(1)+1:sum(cnts(1:2))),'o')
title('Tensions')
xlabel('Span #')
ylabel('lbf')
%ylim([9.2 10.2])
subplot(2,2,3),bar(z0(sum(cnts(1:2))+1:sum(cnts(1:3)))*180/pi)
title('Dancer Position and Velocity')
xlabel('')
ylabel('magnitude')
subplot(2,2,4),bar(z0(sum(cnts(1:7))+1:end))
title('Motor Inputs (amps)')
xlabel('Motor #')
ylabel('Magnitude')
end
z0(41) = 7/12;
z0(37) = 11.4258;

initial_derivative = state_function_controller(0,z0);
y0=initial_derivative;
h =waitbar(0, 'Please Wait');
hstep1 = 0.05*tfinal;
waitbar(0,h);

tstart=0:0.005:tfinal;
refine = 4;

options = odeset('Events',@events,'Refine',refine);
tout = 0;
zout = z0.';
teout = [];
yeout = [];
ieout = [];
istep = 0; % limit the number of bounces off the full extension or full
%          close situation to 50 (arbitrary).

flag = 0;
while tout(end) < tfinal && istep < 550
    if ~flag
```

```
272     [time,z,te,ye,ie] = ode45(@state_function_controller,...
            tstart,z0,options);
274     else
            [time,z,te,ye,ie] = ode45(@state_function_controller,...
276         tstart,z0,options);
        end
278
        nt = length(time);
280   tout = [tout; time(2:nt)];    % dummy variable to store data
      zout = [zout; z(2:nt,:)];     % cummy variable to store data
282   teout = [teout; te];          % Events at tstart are never reported.
      yeout = [yeout; ye];  %value of state vector at time of event
284   ieout = [ieout; ie];  % which event funciton happened

286   if time(nt) < tfinal %check to see if integration needs to continue
          %find which event caused the stop of integration
288       eventcase = ie;
          if isempty(ie)
290           eventcase = 4;
          end
292       switch eventcase;
              case 1
294               %dancer went to +20deg
                  z0(1:sum(cnts(1:2))) = z(nt,1:sum(cnts(1:2)));
296               % Set the new initial conditions, with .9 attenuation.
                  z0(sum(cnts(1:2))+1) = 20*pi()/180;
298               z0(sum(cnts(1:3))) = -.9*z(nt,sum(cnts(1:3)));
                  % Set the rest of the initial conditions
300               z0(sum(cnts(1:3))+1:sum(cnts(1:7))) =...
                      z(nt,sum(cnts(1:3))+1:sum(cnts(1:7)));
302           case 2
                  %dancer went to -20deg
304               z0(1:sum(cnts(1:2))) = z(nt,1:sum(cnts(1:2)));
                  % Set the new initial conditions, with .9 attenuation.
306               z0(sum(cnts(1:2))+1) = -20*pi()/180;
                  z0(sum(cnts(1:3))) = -.9*z(nt,sum(cnts(1:3)));
308               % Set the rest of the initial conditions
                  z0(sum(cnts(1:3))+1:sum(cnts(1:7))) =...
310                   z(nt,sum(cnts(1:3))+1:sum(cnts(1:7)));
              otherwise
312               flag = 1;
          end
314
         % A good guess of a valid first timestep is the length of the last valid
316      % timestep, so use it for faster computation.  'refine' is 4 by default.
          options = odeset(options,'MaxStep',0.005);
318       tstart1 = [time(nt) .005*(1-rem(time(nt),.005)/0.005)+time(nt)];
          tstart = [tstart1(1) tstart1(2):0.005:tfinal];
320
```

```
        istep = istep + 1;
322   end
  end
324
  time = tout; % Return expected value names
326 z = zout;  % Return expected value names
  %------------------ State Function --------
328     function xdot=state_function(T,x,u)
          xdot=x'*0;
330       v = x(1:rolls);
          t = x(rolls+1:rolls+spans);
332       if dancers>0
              x_dance = x(rolls+spans+1:rolls+spans+dancers);
334           x_dancedot = x(rolls+spans+dancers+1:rolls+spans+2*dancers);
          else
336           x_dance = 0;
              x_dancedot = 0;
338       end
          t0 = initten(1);
340       tf = initten(end);
          theta1 = mod(x(sum(cnts(1:3))+1,1),2*pi());
342       [r_1, r_1dot] = Rollshape(theta1,x(sum(cnts(1:3))+2),Rn(unwind),...
              rshape);
344
          tens = [t0; t];
346       for i = 1:spans
              if i == dancer
348               [temp(3), temp(4)] = PendDancerLeng(x_dance,-x_dancedot,...
                      L(i),Larm(i),1.38/12,1.5/12,1.5/12); %.24
350               sp0 = [v(i+[0 1])', tens(i+[0 1],1)', L(i)...
                      , ... %fixed span length
352                   temp(4), theta_in(i),z0(i+[0 1],1)'];
              elseif i+1==dancer
354               [temp(1), temp(2)] = PendDancerLeng(x_dance,-x_dancedot,...
                      L(i),Larm(i+1),3.084/12,1.5/12,1.5/12); %.2486
356               sp0 = [v(i+[0 1])', tens(i+[0 1],1)', L(i)...
                       ,... %fixed span length
358                   temp(2), theta_in(i+1),z0(i+[0 1],1)'];
              elseif i == unwind
360               if case_select == 3   %fixed span length
                  d1 = (Rolls(1,1)-Rolls(1,2))^2+(Rolls(2,1)-Rolls(2,2))^2;
362               l1 = sqrt(d1 - (r_1+Rn(2))^2)-tens(unwind+1)/E/A*L(unwind);
                  dl1 = -(r_1+Rn(2))*r_1dot/sqrt(d1^2-(r_1+Rn(2))^2);
364               sp0 = [v(i+[0 1])',tens(i+[0 1],1)',l1,dl1,theta_on(i)...
                      ,z0(i+[0 1],1)'];
366               else
                      dl1 =0;
368                   sp0 = [v(i+[0 1])' tens(i+[0 1],1)' L(i) 0 theta_in(i)...
                      ,z0(i+[0 1],1)'];
```

```
370                 end
                else
372                 sp0 = [v(i+[0 1])' tens(i+[0 1],1)' L(i) 0 theta_in(i)...
                        ,z0(i+[0 1],1)'];
374             end

376             xdot(rolls+i) = span2Lin(T, sp0, E, A);
            end
378
        % dancer
380     xdot(rolls+spans+1:rolls+spans+[1:dancers]) = x_dancedot;
        %  equation A27
382     % added negative sign 5-28-17
        xdot(rolls+spans+dancers+1:rolls+spans+dancers+dancers) = 1./Jpn...
384         .*(-Cn(dancer).*x_dancedot  -...
            (Kn(dancer).*x_dance + fda(dancer)/12)...
386         - t(dancer-1).*sin(theta_in(dancer))*(Larm(dancer)+Rn(dancer))...
            -t(dancer).*sin(theta_on(dancer))*(Larm(dancer)-Rn(dancer))...
388         -0.317*g*8.895/12*sin(x(rolls+spans+1)));

390     % Rollers
        %velocity dot
392     xdot(1:rolls) = 1./Jn.*(-Bfn.*v' + Rn.^2.*([t(1:end); tf]' -...
            [t0; t(1:end)]'));
394
        %eccentric idler
396     xdot(eccent) = xdot(eccent)+1/Jn(eccent)*(...
            Rn(eccent)*Mn(eccent)*g*Rolls(23,eccent)/12*sin(...
398         x(sum(cnts(1:3))+3,1))/1);

400     % unwind specifically
        if case_select == 3
402         theta1ddot = 1/Jn(unwind)*(-Bfn(unwind)*v(unwind)/1  ...
                + r_1*t(1) -Kmn(unwind)*u(1)+0.298*g*Rolls(23,...
404             unwind)/12*sin(x(sum(cnts(1:3))+1,1)+57.77697/180*pi));
            xdot(unwind) = x(sum(cnts(1:3))+2,1)*r_1dot + r_1*theta1ddot;
406         xdot(sum(cnts(1:3))+[1 2]) = [x(sum(cnts(1:3))+2) theta1ddot];
        elseif case_select == 2
408         %eccentricity from bump
            xdot(unwind) =  xdot(unwind)+1/Jn(unwind)*(...
410             Rn(unwind)*0.298*g*Rolls(23,unwind)/12*sin(...
                x(sum(cnts(1:3))+1,1)+57.77697/180*pi));  %
412         xdot(sum(cnts(1:3))+1) = x(unwind)/Rn(unwind);
        else
414         %just eccentricity
            xdot(unwind) =  xdot(unwind)+1/Jn(unwind)*(...
416             Rn(unwind)*Mn(unwind)*g*Rolls(23,unwind)/12*sin(...
                x(sum(cnts(1:3))+1,1)+57.77697/180*pi));
418         xdot(sum(cnts(1:3))+1) = x(unwind)/Rn(unwind);
```

```
        end
420
        xdot(sum(cnts(1:3))+3) = x(eccent)/Rn(eccent);
422
        % driven
424     xdot(driven) =xdot(driven)+1./Jn(driven).*(-Cmn(driven).*...
           v(driven)' + Rn(driven).*Kmn(driven).*u(2:end)');
426     %span length rate of change
        xdot(sum(cnts(1:4))+[1 2]) = [temp(2) temp(4)]';
428     % unwind radius derivative and span 1 length derivative
        xdot(sum(cnts(1:5))+[1 2]) = [r_1dot dl1]';
430     xdot = xdot';
    end
432
%--------------------State Function controller ------------------
434 function ydot = state_function_controller(T,y)
    % Calls the state_function derivative and adds the control derivative
436 % to be integrated
        ydot = y*0;
438
        if T == 0
440         r3 = IndScurveInput(T,Initvel(2),(deltaV(2)+Initvel(2))...
               *draw,30,100,1);
442         r2 = IndScurveInput(T,Initvel(2),deltaV(2)+Initvel(2),...
               30,100,1); % for driven
444     end
        r1 = ScurveInput(T,0,deltaV(1),Deltaduration(1)); %for dancer
446     r2 = IndScurveInput(T,Initvel(driven(3)),deltaV(2)+Initvel(...
           driven(3)),30/60,100/60); % for driven
448     r3 = IndScurveInput(T,Initvel(2),(deltaV(2)+Initvel(2))*draw,...
           30/60,100/60);
450     %calculate the state derivative with these inputs
        ydot(1:sum(cnts(1:7)),1) = state_function(T,y(1:sum(cnts(1:7))),...
452         y(sum(cnts(1:7))+[1:4]));
        errors = [(r1(1)-y(sum(cnts(1:2))+1)) (r2(1)-y(driven(1))) ...
454          (r2(1)-y(driven(2))) (r3(1)-y(driven(3)));...
             ...
456                (r1(2)-ydot(sum(cnts(1:2))+1)) (r2(2)-ydot(driven(1)))...
             (r2(2)-ydot(driven(2))) (r3(2)-ydot(driven(3)))];
458
        %Calculate motor input derivatives
460     trimV = (Kp(1)*(errors(2,1)) ...
            + Ki(1)*(errors(1,1)) ...
462         + Kd(1)*(y(sum(cnts(1:3))))); %y(rolls+spans+2*dancers+3)
        udot(1) = (Kp(2)*((r2(2)-ydot(unwind)))...
464         + (Ki(2)*((r2(1)-y(unwind))+trimV)));
        %
466     udot(2) = (Kp(3)*errors(2,2)...
            + Ki(3)*(errors(1,2)) + ...
```

```
468             Kd(3)*0);
          udot(3) = (Kp(3)*errors(2,3)...
470             + Ki(3)*(errors(1,3)) + ...
                Kd(3)*0);
472       udot(4) = (Kp(3)*errors(2,4)...
                + Ki(3)*(errors(1,4)) + ...
474             Kd(3)*0);
          %add error signal to integrated vector
476       ydot(sum(cnts(1:7))+[1:inpcount]) = udot';
          % add the errors to the derivative to  to be integrated
478 % update waitbar
          if T> hstep1
480             waitbar( hstep1/tfinal/2, h)
                hstep1 = hstep1 +.1*tfinal;
482       end
          % save derivative for next iteration.
484       y0=ydot;
  end
486 %------------------Sub function for torque fall-off-------------
    function phi = torquecomponent(theta,c,d)
488     %calculates angle to use cosine of for finding the moment component
        %that is available to the pendulum
490     %
        % theta : pendulum angle
492     % c :  pendulum length
        % d :  perpendicular length of the force to its point of origin
494     %         (from the pivot of the air cylinder to the pendulum arm)
        %
496     phi  = theta + atan(d*(1-cos(theta))/(c - d*sin(theta)));
    end
498 %-----------------Integration Events function-----------------------
    function [value,isterminal,direction] = events(t,y)
500 % Locate the time when height passes through zero in a decreasing direction
  % and stop integration.
502     value = [y(sum(cnts(1:2))+1) - 20*pi()/180 ...
            y(sum(cnts(1:2))+1) + 20*pi()/180];   % detect dancer angle = 20deg
504     isterminal = [1  1];   % stop the integration
        direction = [1  -1];   % [positive negative] direction
506     end
  %-------------------------------------------------------------------
508 %% Plotting
  %-------------------------------------------------------------------
510


512 velocities_out = z(:,1:rolls)*60; %fpm
  tensions_out = z(:,rolls+1:rolls+spans);
514 dancer_out = z(:,sum(cnts(1:2))+[1  2])*180/pi; %degrees
  motor_inputs_out = [z(:,sum(cnts(1:7))+1:sum(cnts(1:7))+inpcount)];
516
```

```
velindex = [1 4 9 10 15];
tenindex = [1 4 9 10 15];
waitbar( .6, h)

figure(2);
subplot(2,2,1)
plot(time,velocities_out(:,velindex))
title('velocity');
legend(['v' num2str(velindex(1))],['v' num2str(velindex(2))],...
    ['v' num2str(velindex(3))],['v' num2str(velindex(4))],'location','best');
grid;

subplot(2,2,2)
plot(time,tensions_out(:,tenindex))
title('tension');
legend(['t' num2str(tenindex(1))],['t' num2str(tenindex(2))],...
    ['t' num2str(tenindex(3))],['t' num2str(tenindex(4))],'location','best');
grid;

subplot(2,2,3)
plot(time,dancer_out(:,1));%,time,dancer_out(:,1))
title('dancer');
legend( '\theta (deg)','location','best')%,'xdot');
grid;

subplot(2,2,4)
plot(time,motor_inputs_out)
title('motor inputs');
legend('u1','u2','location','best');
grid;

waitbar( .7, h)
figure(3)
ff=z(:,rolls+[8:9])*[.5 .5]';
ffplot = ff;
m=mean(ff);
ff=ff-m;

T = mean(time(2:end)-time(1:end-1));              % Sampling period
Fs = 1/T;            % Sampling frequency
Lfft = length(ff);             % Length of signal
if mod(Lfft,2)>0
    Lfft = Lfft-1;
end
Y = fft(ff);
P2 = abs(Y/Lfft);
P1 = P2(1:Lfft/2+1);
P1(2:end-1) = 2*P1(2:end-1);
f = Fs*(0:(Lfft/2))/Lfft;
```

```
566
    freq = [0.1999 0.422 0.91 1.067 1.82, 2.1 2.12 3.44 3.536 4.24 ...
568 4.528 5.075 5.083 5.13 5.89 6.456 6.74 8.48 11.22];

570 stor = [];
    for i = 1:numel(freq)
572     stor = [stor find(f-freq(i)>0,1,'first')];
    end
574
    waitbar( .8, h)
576 loglog(f,P1,f(stor),P1(stor),'r*')
    title('Frequency response of Loadcell Tension')
578 xlabel('f (Hz)')
    ylabel('Tension Variation')
580 xlim([0 100])

582 figure(4)
    plot(time,ffplot)
584 title('Loadcell Tension')
    xlabel('Time (sec)')
586 ylabel('Tension (lbf)')

588 file =['C:\Users\quietman\Desktop\Euclid line\10-25-17' ...
        '\TenFb_DanP_Dia_Dancer_10_lb_unwind_tension.xlsx'];
590 a = xlsread(file,1,'d15:j11662');
    for i=1:numel(time)-8;
592     lc(i,1) = ones(1,8)/8*(z([0:7]+i,17+[8:9])*[.5 .5]');
        lc(i,2) = ones(1,8)/8*(a([0:7]+i,2));%pulroll
594     lc(i,3) = ones(1,8)/8*(a([0:7]+i,6));%unwind load cell
        lc(i,4) = ones(1,8)/8*(z([0:7]+i,17+[14:15])*[.5 .5]');
596 end
    figure(6)
598 %plot(time(9:end)',lc(:,1),time(9:end),lc(:,2))%a(:,1),a(:,2),'r')
    plot(time(9:end)',lc(:,1),time(9:end),lc(:,2),time(9:end),lc(:,3),...
600     time(), z(:,17+[14:15])*[.5 .5]')%time(9:end),lc(:,4))
    xlabel('Time (sec)','FontSize', 16)
602 ylabel('Tension (lbf)','FontSize', 16)
    title('Load Cell Tension Indication','FontSize', 16)
604 legend('Sim Unwind','Pull Roll','Unwind','Sim Pull Roll')

606 waitbar( 1, h)
    SS = z0;
608 SSder = initial_derivative;
    y=z;
610 close(h)
    if nargout >4
612     Events = {teout, yeout, ieout};
    end
614 end
```

## J.2 Euclid Web Line with Slip Simulation Code

Slip was investigated on the EWL to verify its presence and compare models of slip to one another. One of the key elements about slip between the web and roller is the interface coefficient of friction. the coefficient was calculated from tests detailed in section H.5 of the Appendix. This was the 18th major version of the Slip Model.

```
function [ SS, SSder, time, z, Events ] = SlipModel18( tfinal )
%Euclid line Slip model with 25 rollers.
% Added eccentricity to driven #3 roll.  Making the roller
% velocity derivative and span tension derivative all callable functions.
% Then, will change which one is called based on the condition of the event
% for  slip.  This model initially (and all previous versions: 01, 02, 03)
% used the diameter instead of the radius.  Version 05 amends that.
%   Detailed explanation goes here
%   The main use of this script is to study the cause of slip per the test
%   outlined by WTS and John Newton.  Here, the bearing friction is set to
%   either 0.0065 or 0.065.  Slip occurs at the larger bearing friction
%   value, but not at the smaller value. (c) Ben Reish 2018
%
%   Version 06 - Dr. Reid wants to study what happens if the equations
%   aren't switched until the next time step.  So see the trigger, set a
%   flag, but don't change equations yet.  That's not working.  I have
%   copied thed events function, making a slip event and a stick event.
%  Now along with changing the state equations, the event function is
%   switched out.
%
%   Ver. 07 - Put the eccentricity on the unwind instead of the driven
%   roller.  Tension set to 5 lbf.
%
%   Ver. 08 - Changing the line to Example 4 from WTS.  5lbf tension and
%   unwind eccentricity.  Values taken from EXAMPLE 4 MODIFIED TO BE LIKE
%   EUCLID LINE.wts.  Controller gains have to be scaled to the right
%   units.  The control loop was updated to the modified PID variant.  The
%   Steady-state was found with a slight change.  Added just a little
%   dancer damping.  Nearly the same thing can be done by adding dancer
%   spring constant.
%
%   Ver. 09 - Taking V08 and linearizing it to coincide with WTS
%   simulation. Changing the dancer equations and the spans into and out of
%   the dancer equations.  Also, changing what the slip event does.  Now it
%   only updates the current values of the tension and leaves the equations
%   alone.  Added slipind variable to select what roller to watch for slip.
%   Adding a nip capability to driven roller.  Added slipind variable to
%   select what roller to watch for slip.
%
%   Version 9b - Linearized to WTS equations for spans into and out of
```

```
%   dancer roller. Adding a nip capability to driven roller through use of
%   nip5  boolean.
%
%   Version 10 - Took 9b and am attempting to post process for slip.
%   PostProcessedSlip function works but does not impact the other
%   variables that would be impacted.  Going to try to execute at the start
%   of the state_derive loop for that one time.
%
%   Version 11 - Took 10 and am using the tests used in Whitworth's Ph. D.
%   thesis instead of what got transcribed into WTS help.  Whitworth's work
%   is different than WTS in small ways, but there is not enough
%   information in the WTS help to explain the change in logic.
%
%   Version 12 - Took 11 and made my own RK4 integrator to control when
%   slip is checked for.
%
%   Version 13 - Took 11 and am removing the slip checks in the event
%   functions.  These will only by used to check for limits like dancer
%   travel or tensions going to 0. Used odeBR4 to integrate and check for
%   slip each iteration.
%
%   Version 14 - Took 13 and am making a persistent variable to capture
%   values of ydot to use in a backward finite difference derivative to
%   calculate the 2nd derivative for my PID controllers. (Unfinished)
%
%   Version 15 - Using 13, putting in the Euclid line dimensions from 1 to
%   roller 17.  Applied switch to allow for actual viscous bearing drag or
%   Coulomb friction drag. Fixed Pullroll inertia and mass (it was calced
%   with a 12 in diameter instead of a 6in one).  Added Moad's gains for
%   the drives.  They go unstable unless I divide them by 12 which similar
%   to the effect of converting to radians from degrees (pi/180 approx
%   1/12).  Using parasitic torque on roller 9 from experiments. Allows use
%   of a nipped SWRAP lead by nip10=1.
%
%   Version 16 - Using 15 and updating the frictions and bearing drags for
%   the new arrangement of rollers.
%
%   Version 17 - Expanding Version 16 to all 25 rollers and 5 controllers.
%   Dancer control is modeled here with all of Moad's gains.  Initially I
%   divided by 60 on the speed gains because of the model units in feet,
%   seconds, and pounds instead of fpm.  I divided the tension loop gains
%   on the unwind by 12 (this was a hold over from when I modeled the line
%   with a translational dancer for simplification and did not correct it
%   when I switched back to a pendulum dancer.
%
%   Version 18 - Copied 17.  Putting slip on roller 21 becaue Ducotey-Good
%   method from WTS says that one should slip. Used a new data set for the
%   Euclid line that has the last configuration of the roller and spans.
%   Adding the Ducotey - Good traction model to the mix.  Editted the
```

```
%   PostProcessing function to only use the DG object on the roller it was
%   set up for.  Uses constant coeffs. (Mun) elsewhere.
%
%
if nargin <1
    tfinal = 30;
end
file='C:\Users\quietman\Documents\MATLAB\WTS\EUCLID_LINE_041019D.Dat';
dancers = 1; %number of dancers in system
%                           filename, number of rolls, number of spans
[r,s,c]=ElementPropReadIn(file,25,24);
Rolls = r{1};
Spans = s{1};
funnames = {@NonSlipVelDer,@Span2,@Span2,@Span2,@Span2};
RecordEvents(0,0,0,1);
DG = DucoteyGood('ReishWebTyv-1','ReishRoller1'); %built into DucoteyGood
%       object
DG.set('Ra_w',3.8e-6/.0254/12); % 12\muft roughness
DG.set('Rq_w',3.8e-6/.0254/12+14.9e-6/12); % 12\muft roughness
%all commented out is 0 permeability...
%Tyvek 1443R permeability
DG.set('alpha',4.166e-2/144); % ft^3/s/(ft^2-lbf/ft^2)
%Tyvek 1025D permeability
%DG.set('alpha',8.983e-2/144); % ft^3/s/(ft^2-lbf/ft^2)
%NP-3 permeability
%DG.set('alpha',2.828e-2/144); % ft^3/s/(ft^2-lbf/ft^2)
%NP-1 permeability
%DG.set('alpha',5.158e-2/144); % ft^3/s/(ft^2-lbf/ft^2)

%Set roller roughness
%  DG.set('Ra_r',8e-6/12);
%      DG.set('Rq_r',8.8e-6/12);
% DG.set('Ra_r',32e-6/12);
%     DG.set('Rq_r',35.6e-6/12);
% all commented out is 63e-6/12
% DG.set('Ra_r',125e-6/12);
%     DG.set('Rq_r',137.5e-6/12);
% DG.set('Ra_r',250e-6/12);
%     DG.set('Rq_r',275e-6/12);
% DG.set('Ra_r',500e-6/12);
%     DG.set('Rq_r',550e-6/12);

%Set Web Roughness
%    tyvek web roughness = [3.8e-6/.0254/12 4.7e-6/.0254/12]; % ft
% all commented out is Ra_w = 3.8e-6/.0254/12 ft Tyvek 1025
% DG.set('Ra_w',40e-6/12);
%     DG.set('Rq_w',43.8e-6/12);
% DG.set('Ra_w',4.7e-6/.0254/12);   %Tyvek 1073
%     DG.set('Rq_w',5.1699e-6/.0254/12);
```

```
%———————— Run Parameters ————————————————
rolls = 25;
spans = 24;
deltaV = ones(1,rolls)*400/60; %increase in speed in ft/s
Initial_Vel = ones(1,rolls)*00/60; %initial speeds of rollers in ft/s
%Initial_Vel(17) = 399.7/60;
deltaV = deltaV*4/4;
Initial_Vel = Initial_Vel*4/4;
t0 = 4; %lbf wound in tension
tf = 5; %outgoing tension in lbf
lineten = 5; % line tension setpoint
lineten2 = 20; %starting tension of rewind
dancer = [4];
acc = [];
unwind = [1];
rewind = [25];
eccent = [1];
Ind = zeros(rolls,1);
cnts = [rolls spans 2 1 5];
slipind = 3;
nip10 = 1; % 1 use nip, 0 don't use nip
nip17 = 1; % 1 use nip, 0 don't use nip
SFDR = 0;  % 1 use SFDR in slip check, 0 normal Whitworth
bearing_sliding_friction = 1;  %boolean to toggle between
%                               viscous and sliding friction models
accel_rate = 30/60; %feet per min/sec in ft/s^2  Used in IndScurve
jerk_rate = 50/60; %feet per min/sec^2 in ft/s^3
    %Ben's Gain calc Method Gains
%unwind tension loop gains
Kp(1)=0.4735/30;  %4/12;%
Ki(1)=0.2742; % 9.4/12;%
Kd(1)=0/12;%
%unwind speed loop gains
Kp(2)=2.7928; %18.95/60;%1.8;%
Ki(2)=16.1576; %51.14/60;%5.19;%
Kd(2)=0.0/60;%0.005/100;%
%S-wrap Lead speed loop gains
Kp(3) = 5.5421; %345.232/60; %
Ki(3) = 32.0624; %265.354/60; %
Kd(3) = 0/60;
%S-wrap Follower speed loop gains
Kp(4) = 5.5421; %345.232/60; %
Ki(4) = 32.0624; %265.354/60; %
Kd(4) = 0/60;
%Pull roll tension loop gains
Kp(5) = 0; %345.232/60; %
Ki(5) = 0; %265.354/60; %
Kd(5) = 0/60;
%Pull roll speed loop gains
```

```
Kp(6) = 5.9853; %345.232/60; %
Ki(6) = 34.6265; %265.354/60; %
Kd(6) = 0/60;
%rewind tension loop gains
Ki(7) = 0.1718*.05;%1.8; %
Kp(7) = .0001552*.1%5.19; %
Kd(7) = 0;%0;
%rewind speed loop gains
Ki(8) =  3.91;%1.8; %
Kp(8) = 22.697;%5.19; %
Kd(8) = 0.0/60;%0;
g = 32.174; %ft/s^2 gravity
slipflag = 0; % signal to indicate that there is slip...
y0 = zeros(57,1); %y0 = [y0; -0.005];
simrun = 0;
%---------- Roller params ----------
L = Spans(4,:)/12; %feet
width_w = Spans(3,1)/12;%feet
rho_w = Spans(5,:)*12^3/g;%lbm/in^3*(12in)^3/ft^3 / (lbm*ft/(s^2*lbf))
%                                 = lbf*s^2/ft^4
A = Spans(2,1)/1000/12 * width_w; %ft^2
E = Spans(1,1)*1000*12^2;%lbf/ft^2
%Rollers
Rn = Rolls(3,:)/12/2;% radius in ft for all rollers
Jn = Rolls(13,:); %lbf*ft*s^2
Cmn = Rolls(19,:); % Torque/speed constant of a motor
Kmn = Rolls(18,:); % Torque constant of a motor, lbf*ft/amp
Bfn = Rolls(20,:)*(1-0); % lbf*ft*sec, Bearing friction  ----------<
theta_arm = Rolls(11,:)*pi/180;%dancer arm angle from horizontal in rad
Mn = Rolls(10,:)/g; %slugs
Mun = [ones(1,rolls)]*.2403; % static friction max from measurements
Mun([9 20]) = [0.1185 0.15];  %from measurements, tables in
%               EuclidExp.pdf roller 9 and 13...
e_3 = 0.0/12; %ft eccentricity
Jpn = 0.209; %inertia of the dancer arm
Kn = [zeros(1,5) 0 0 0 0 0 0 0 zeros(1,13)];
Cdn = [zeros(1,3) 0.15 0 0 0 0 0 0 0 0 zeros(1,13)];
f_da = 34; %36.2lbf external force on dancer.
Rn_n = [zeros(1,9) 1 zeros(1,5) 1 zeros(1,8)]*3/24; %Nip radius in feet
Jn_n = [zeros(1,9) 1 zeros(1,5) 1 zeros(1,8) ]*4.635e-3; %nip inertias
Bfn_n = [zeros(1,9) 1 zeros(1,5) 1 zeros(1,8)]*0.0006; %nip bearing friction
Bfn = [0.00060729 0.00016691 0.00126336 0.00017803 0.00004665 ...
    0.00002655 0.00050257 0.00062522 0.00061941 0.00049733  ...
    0.00061733*ones(1,15)] ;
Bsldfn = [.1 0.05227297 0.41489625      0.17357908      0.23150706...
        0.13421975  0.16057590  0.20012461      0.20024913 .1 .1 ...
        0.15779922*ones(1,14)]; %mu's from SpinDownTest.m
Bsldfn = Bsldfn*diag([38*(1.5/24) 12.89*.915/12*[1 1] ...
    1.188*.915/12 .876*.915/12*[1 1] 12.89*.915/12*[1 1 1] ...
```

```
    78.454*.915/12*[1 1] 12.89*.915/12*[1 1 1 1 1] 42*1.5/24 ...
238     12.89*.915/12*ones(1,8)])*.05;
theta_arm = zeros(1,rolls);
240 theta_arm(dancer) = pi/2;
Larm = [zeros(1,3),15.925,zeros(1,21)]/12;
242 ParasiticTorque = [0 0*Rn(2) 0 0.00*Rn(4) 0 0 0 0 Rn(9)*1.419*0 ...
    zeros(1,10) Rn(20)*0 zeros(1,5)];
244 %---------- Span params ----------
%E = 46300*144; %Young's modulus in lbf/ft^2
246 A = 6.07/12*0.00536/12; % cross-sectional area in ft^2
rho = 1.0740000E-0002/g*144*12 ; %density of web in slug/ft^3
248 rot_dir = [-1 1 -1 1 -1 -1 -1 1 -1 -1 1 1 -1 1 1 -1 ...
    1 -1 1 1 -1 1 -1 1 -1]; %forward web direction CW = -1, CCW = 1
250 [sdta, theta_in,theta_on] = PrintLine(file,25,rot_dir,5,theta_arm);
theta_in=theta_in(1:rolls);
252 theta_on = theta_on(1:rolls);
theta_wrap = WrapAngle(sdta,Rn);
254 DG.set('beta',theta_wrap(20));
fprintf(1,'Roller 20 Wrap Angle: %2.3f%s\n',theta_wrap(20)*180/pi,char(176))
256 %---------- Initial Conditions ----------
z = fsolve(@steadystate,rand(57,1), optimoptions('fsolve','Algorithm',...
258     'Levenberg-Marquardt','TolFun',1e-16,'MaxIter',4000,'TolX',1e-6));
z([10:11,16],1) = Initial_Vel([10:11, 16])';
260 z(rolls+17,1) = 2*lineten2-z(rolls+18);
z(rolls+dancer-1,1) = (f_da*5.032/12-z(rolls+dancer)*cos(theta_on(dancer))...
262     *(15.925-1.5)/12)/((15.92+1.5)/12)/cos(pi-theta_in(dancer));
z(rolls+10,1) = [.5 .5]*z(rolls+[9 11],1);
264 z(rolls+14,1) = 2*lineten-z(rolls+15,1);
z(sum(cnts(1:4))) = z(eccent)/Rn(eccent);
266 velocities_ss = z(1:rolls,1);
tensions_ss = z(rolls+[1:spans]);% 3.4*ones(14,1); z(rolls+[20:22])];
268 dancer_ss = [0 0]';
motors_ss = z(sum(cnts(1:4))+[1:5]);
270
% Solve stuff
272 z0 = [velocities_ss; ...
    tensions_ss; ...
274     dancer_ss;...
    z(sum(cnts(1:4)));...
276     motors_ss]; %7x1
figure(1);
278 subplot(2,2,1),plot(velocities_ss*60,'>')
title('Velocities')
280 xlabel('Roller #')
ylabel('fpm')
282 subplot(2,2,2),plot(tensions_ss,'o')
title('Tensions')
284 xlabel('Span #')
ylabel('lbf')
```

```
%ylim([9.2  10.2])
subplot(2,2,3),bar(dancer_ss*12)
title('Dancer Position (in) and Velocity (in/s)')
xlabel('')
ylabel('magnitude')
subplot(2,2,4),bar(motors_ss)
title('Motor Inputs (amps)')
xlabel('Motor #')
ylabel('Magnitude')
SS = z0;
h =waitbar(0, 'Please Wait');
hstep1 = 0.05*tfinal;
waitbar(0,h);
tstart = 0;
SSder = State_Controller(tstart,z0);
%-------------------------------------------------
%  I N T E G R A T I O N
%-------------------------------------------------
tstart=0:0.005:tfinal;
refine = 4;
%options = odeset('Events',@events1,'Refine',refine,'RelTol',1e-5);%for
%ode45
options = {'EventFunction',@PostProcessedSlip;'TimeStep',0.002;...
    'SlipTrue',1}; %for odeBR4
tout = 0;
zout = z0.';
teout = [];
yeout = [];
ieout = [];
istep = 0; % limit the number of bounces off the full extension or full
simrun=1;
slipflag = 0;
timetest = 0.002; % ---- updated each timestep to compare and current time
%           and wait until the next one to switch equations
%           close situation to 150 (arbitrary).
while tout(end) < tfinal && istep < 5500

    [time,z] = odeBR4(@State_Controller,...
        tstart([1 end]),z0,options);%,options); ,te,ye,ie
ie=[];
    nt = length(time);
  tout = [tout; time(2:nt)];    % dummy variable to store data
  zout = [zout; z(2:nt,:)];     % cummy variable to store data
%  teout = [teout; te];          % Events at tstart are never reported.
%  yeout = [yeout; ye];  %values of event function (1 of them will be 0 )
  if size(ie,1)>1
    ieout = [ieout; ie+10*slipflag];  %
  else
      ieout = [ieout; ie'+10*slipflag];
```

```
    end
336 if time(nt) < tfinal %check to see if integration needs to continue
        %find which event caused the stop of integration
338     if isempty(ie)
            options = odeset(options,'MaxStep',0.005);
340         tstart1 = [time(nt) (.005-rem(time(nt),.005))+time(nt)];
            tstart = [tstart1(1) tstart1(2):0.005:tfinal];
342
            [time,z] = ode45(@State_Controller,...
344       tstart,z(end,:),options);
          nt = length(time);
346    tout = [tout; time(2:nt)];    % dummy variable to store data
      zout = [zout; z(2:nt,:)];     % dummy variable to store data
348   teout = [teout; te];          % Events at tstart are never reported.
      yeout = [yeout; ye];  %values of event function (1 of them will be 0 )
350   ieout = [ieout; ie];  %
        else
352          z0 = z(nt,:);
            for i=1:numel(ie)
354             eventcase = ie(i);
          if slipflag == 0
356             switch eventcase;
            case 1
358             %T3 < T2*exp(-Mun*theta_wrap(3))
                Ind(slipind) = -1;
360             options = odeset(options,'MaxStep',.005); %@events2);
                  nm1=slipind-1;
362               n = slipind;
                z0(rolls+nm1) = (z0(rolls+nm1)*L(nm1)+z0(rolls+n)*L(n))/...
364                 (L(nm1)+L(n)*exp(Ind(n)*Mun(n)*theta_wrap(n)));
                z0(rolls+n) = z0(rolls+nm1)*exp(Ind(n)*Mun(n)*theta_wrap(n));
366         case 2
                %T3 < T2*exp(Mun*theta_wrap(3))
368             Ind(slipind) = 1;
                options = odeset(options,'MaxStep',.005);%@events2);
370               nm1=slipind-1;
                  n = slipind;
372             z0(rolls+nm1) = (z0(rolls+nm1)*L(nm1)+z0(rolls+n)*L(n))/...
                    (L(nm1)+L(n)*exp(Ind(n)*Mun(n)*theta_wrap(n)));
374             z0(rolls+n) = z0(rolls+nm1)*exp(Ind(n)*Mun(n)*theta_wrap(n));
                end
376       else
            switch eventcase;
378         case 1
                % Returning from slip condition 1
380             funnames{1} = @NonSlipVelDer;
                funnames{2} = @Span2;
382             funnames{3} = @Span2;
                Ind(2) = 0;
```

```
384                 options = odeset('Events',@events1);
                    %slipflag = 0;
386             case 2
                    % Returning from slip condition 2
388                 funnames{1} = @NonSlipVelDer;
                    funnames{2} = @Span2;
390                 funnames{3} = @Span2;
                    Ind(2) = 0;
392                 options = odeset('Events',@events1);
                    %slipflag = 0;
394                 end
            end
396             switch eventcase
             case 3
398                 istep = 5500;
             case 4
400                 istep = 5500;
                    end
402          end
        end
404     z0=z0';
        tstart1 = [time(nt) (.005-rem(time(nt),.005))+time(nt)];
406     tstart = [tstart1(1) tstart1(2):0.005:tfinal];
        istep = istep + 1;
408   end
  end
410
  time = tout; % Return expected value names
412 z = zout;  % Return expected value names
  %[time,z,EEvents] = PostProcessedSlip(time,z);
414 %-----------------------------------------------------------------
  %           S T A T E   D E R I V A T I V E   F U N C T I O N
416 %-----------------------------------------------------------------
    function xdot = state_derive(T,x,u)
418
        v = x(1:rolls,1);
420     t = x(rolls+[1:spans],1);
        xdot = x*0;
422     % ---- roller velocity derivs
        cv = FrictionCalc(v,t,'Parasitic_Torque',0,'sliding_friction',0);
424     xdot(1:rolls,1) = 1./Jn'.*cv;
        xdot(20,1) = feval(funnames{1},t(19:20),v(20),20);
426     xdot(unwind,1) = xdot(unwind,1) + 1/Jn(1)*(16.029*(u(1)*Rn(1))+...
            Rn(1)^2*t0+Rn(1)*Mn(1)*e_3*g*sin(x(sum(cnts(1:4)),1)));
428     if nip10
            %use nip roller equation for roller 10
430         xdot(10,1) = 1/(Rn(10)^2*Jn_n(10)+Rn_n(10)^2*Jn(10))*( ...
                -(Rn_n(10)^2*Bfn(10)+Rn(10)^2*Bfn_n(10))*v(10) + ...
432             Rn_n(10)^2*Rn(10)^2*(t(10) - t(9))+ ...
```

```
            Rn_n(10)^2*Rn(10)*62.494*u(2));
434     else
            %no nip
436         xdot(10,1) = xdot(10,1) + 1/Jn(10)*(62.494*(u(2)*Rn(10)));
        end
438         xdot(11,1) = xdot(11,1) + 1/Jn(11)*(62.494*(u(3)*Rn(11)));
        if nip17
440         %use nip roller equation for roller 17
            xdot(16,1) = 1/(Rn(16)^2*Jn_n(16)+Rn_n(16)^2*Jn(16))*( ...
442             -(Rn_n(16)^2*Bfn(16)+Rn(16)^2*Bfn_n(16))*v(16) + ...
                Rn_n(16)^2*Rn(16)^2*(t(16) - t(15))+...
444             Rn_n(16)^2*Rn(16)*33.48*u(4));
        else
446         %no nip
            xdot(16,1) = xdot(16,1) + 1/Jn(16)*(33.48*(u(4)*Rn(16)));
448     end

450         xdot(rolls,1) = xdot(rolls,1) + 1/Jn(rolls)*(16.029*(u(5)...
                *Rn(rolls)) - Rn(rolls)^2*tf); %-Rn(rolls)^2*tf*0
452     % ---- Dancer derivs
        xdot(sum(cnts(1:2))+1,1) = x(sum(cnts(1:2))+2,1);
454     xdot(sum(cnts(1:2))+2,1) = 1/(Jpn)*(f_da*5.032/12 ...  %/(Mn(dancer))
            - t(3)*cos(pi-theta_in(dancer))*(15.925+1.5)/12 ...
456         - t(4)*cos(theta_on(dancer))*((15.925-1.5)/12) ...
            - Kn(4)*x(sum(cnts(1:2))+1) - Cdn(4)*x(sum(cnts(1:2))+2)...
458         - 13.95/12*.317*g*sin(x(sum(cnts(1:2))+1)));
        % ---- tension derivs
460     tens = [t0; t]; %include wound in tension
        for i = 1:spans
462         if i == dancer
                [temp(3), temp(4)] = PendDancerLeng(x(sum(cnts(1:2))+1),...
464                 -x(sum(cnts(1:2))+2),L(i),Larm(i),1.38/12,1.5/12,1.5/12);
                sp0 = [v(i+[0 1])', tens(i+[0 1],1)', L(i)+...
466                 temp(3), ... %Length does not change
                        temp(4), theta_on(i)]';
468         elseif i+1==dancer
                [temp(1), temp(2)] = PendDancerLeng(x(sum(cnts(1:2))+1),...
470                 -x(sum(cnts(1:2))+2),L(i),Larm(i+1),3.084/12,1.5/12,1.5/12);
                sp0 = [v(i+[0 1])', tens(i+[0 1],1)', L(i)+...
472                 temp(1),... %Length does not change
                    temp(2), theta_in(i+1)]';
474         else
                sp0 = [v(i+[0 1])' tens(i+[0 1],1)' L(i) 0 theta_on(i)]';
476         end

478         xdot(rolls+i,1) = span2(T, sp0, E, A);
        end
480     if Ind(20)~=0
            %if slip is indicated, then use the slip equations
```

```
482             % Added Ducotey Good traction to find Mu 7-16-19
                mu = DG.predict(v(16),v(20),t(20),Mun(20));%.9999*Mun(20);
484             props = [L(19)+L(20)*exp(Ind(20)*mu*theta_wrap(20)), E*A, ...
                    t(18), t(20), v(19), v(21)];
486             xdot(rolls+19,1) = feval(funnames{2},props);
                props = [xdot(rolls+19,1),exp(Ind(20)*mu*theta_wrap(20))];
488             xdot(rolls+20,1) = feval(funnames{3},props);
            end
490         %Update the eccentric roller's angular speed
            xdot(sum(cnts(1:4)),1) = x(eccent,1)/Rn(eccent);
492     end
    %-------------------------------------------------------------------------
494 %           S T A T E   C O N T R O L L E R   F U N C T I O N
    %-------------------------------------------------------------------------
496     function [ydot] = State_Controller(t,y)
            ydot = y*0;
498         % Setup inputs
            if t == 0
500                 % for unwind speed
                r1 = IndScurveInput(t,z0(1),deltaV(1),accel_rate,jerk_rate,1);
502                 % for Driven Speed
                r2 = IndScurveInput(t,Initial_Vel(10),deltaV(10),accel_rate,...
504                 jerk_rate,1);
                    % for pullroll Speed
506             r3 = IndScurveInput(t,z0(16),deltaV(1),accel_rate,jerk_rate,1);
                    % for Rewind Speed
508             r4 = IndScurveInput(t,z0(25),deltaV(1),accel_rate,jerk_rate,1);
            end
510         % get reference values
            if t > 0 && t<10
512         r1 = IndScurveInput(t,z0(1),deltaV(1),accel_rate,jerk_rate);
                %for unwind speed
514         r2 = IndScurveInput(t,Initial_Vel(10),deltaV(10),accel_rate,...
                jerk_rate);
516             % for driven speed
            r3 = IndScurveInput(t,z0(16),deltaV(1),accel_rate,jerk_rate);
518             % for pullroll speed
            r4 = IndScurveInput(t,z0(25),deltaV(1),accel_rate,jerk_rate);
520             % for Rewind Speed
            else
522             r1 = IndScurveInput(t,z0(1),deltaV(1),accel_rate,jerk_rate);
                    %for unwind speed
524             r2 = IndScurveInput(t,Initial_Vel(10),deltaV(10),accel_rate,...
                    jerk_rate);
526                 % for driven speed
                r3 = IndScurveInput(t,z0(16),deltaV(1),accel_rate,jerk_rate);
528                 % for pullroll speed
                r4 = IndScurveInput(t,z0(25),deltaV(1),accel_rate,jerk_rate);
530                 % for Rewind Speed
```

```
        end
532     % get state derivative
        ydot(1:sum(cnts(1:4)),1) = state_derive(t,y(1:sum(cnts(1:4)),1),...
534         y(sum(cnts(1:4))+[1:cnts(5)],1));
        %calc error from dancer
536     F(1) = 0 - y(rolls+spans+1,1);
        Fdot(1) = 0 - y(rolls+spans+2,1);
538     %calc error from loadcell for pull roll, check to see if used
        F(2) = 0;%[.5 .5]*y(rolls+[18 19],1);
540     Fdot(2) = 0;%[.5 .5]*ydot(rolls+[18 19],1);
        %calc error from loadcell
542     F(3) = [.5 .5]*y(rolls+[17 18],1);
        Fdot(3) = [.5 .5]*ydot(rolls+[17 18],1);
544     % find errors in speed
        errors = [ r1(1)-y(1),    r2(1)-y(10),    r2(1)-y(11),      ...
546         r3(1)-y(16),        r4(1)-y(rolls);...
                r1(2)-ydot(1), r2(2)-ydot(10), r2(2)-ydot(11), ...
548         r3(2)-ydot(16),     r4(2)-ydot(rolls)];
        % find trim speeds
550     TrimV(1) = (Kp(1)*(Fdot(1))+Ki(1)*(F(1))+...
            Kd(1)*(ydot(rolls+spans+2,1)-y0(rolls+spans+2,1)));
552     TrimV(2) = (Kp(5)*(-Fdot(2))+Ki(5)*(lineten-F(2)));
        TrimV(3) = (Kp(7)*(-Fdot(3))+Ki(7)*(lineten-F(3)));
554     % calc motor current derivatives
        ydot(sum(cnts(1:4))+1,1) = (Kp(2)*(errors(2,1)+TrimV(1)*0)+...
556         Ki(2)*(errors(1,1)+TrimV(1))+...
            Kd(2)*(TrimV(1)+r2(3)-(ydot(1)-y0(1))/.005));
558     ydot(sum(cnts(1:4))+2,1) = Kp(3)*errors(2,2)+Ki(3)*errors(1,2)+...
            Kd(3)*(r2(3)-y0(10));
560     ydot(sum(cnts(1:4))+3,1) = Kp(4)*errors(2,3)+...
            Ki(4)*(errors(1,3));
562     ydot(sum(cnts(1:4))+4,1) = Kp(6)*(errors(2,4))+...
            Ki(6)*(errors(1,4)+TrimV(2)*0);
564     ydot(sum(cnts(1:4))+5,1) = Kp(8)*(errors(2,5)+TrimV(3)*0)+...
            Ki(8)*(errors(1,5)+TrimV(3));
566     y0=ydot;
        % update waitbar
568     if t> hstep1
            waitbar( hstep1/tfinal/2, h)
570         hstep1 = hstep1 +.1*tfinal;
        end
572 end
    %---------- NonSlip idler vel. derive function
574 function dv = NonSlipVelDer(tens,v,ind)
        dv = 1/Jn(ind)*(-Bfn(ind)*v+ Rn(ind)^2*([-1 1]*tens)...
576         - ParasiticTorque(ind)*Rn(ind));
    end
578 %---------- Slip idler vel. deriv function
    function dv = SlipIdlerVelDer(tens,v,ind)
```

```
580         Fn = [-cos(theta_in(ind)) cos(theta_on(ind))]*tens(1:2,1);
            if v < 0.001
582             Pars = 0;
            else
584             Pars = ParasiticTorque(ind);
            end
586         dv = 1/Jn(ind)*(-Bfn(ind)*v + Ind(ind)*Mun(ind)*Fn*Rn(ind)^2 ...
                - Pars*Rn(ind));
588     end
        %---------- NonSlip Tension deriv span 1
590     %Just use Span.m like normal
        %---------- Slip Tension deriv span 1
592     function dt = SlipTensionIncoming(Props)
            %Taken from WTS discussion of Turn bars
594         %Props = [L1+L2*e^(mu*alpha), E*A, t0, T2, V1, V3]
            L1 = Props(1); EA = Props(2); Tnm1 = Props(3); Tnp1 = Props(4);
596         Vn = Props(5); Vnp2 = Props(6);
            dt =1/L1*(Vnp2*(EA - Tnp1) - Vn*(EA - Tnm1));
598     end
        %---------- Slip Tension deriv span 2
600     function dt = SlipTensionOutgoing(Props)
            %Taken from WTS discussion of Turn bars
602         %Props = [dT1, exp(mu2*alpha2)];
            dT1 = Props(1); emu = Props(2);
604         dt = dT1*emu;
        end
606     %---------- Events function
        function [value,isterminal,direction] = events1(t,y)
608 % Locate the time when tension ratio violates capstan equation
    % and stop integration.
610     %slipflag = 1;
        %Ind(2)  = 0;
612     nm1 = slipind-1;
        n = slipind;
614     testforTension = find(y(rolls+[1:spans])<0);
        value = [ ...%y(sum(cnts(1:2))+1) - 50/12, ...
616         %y(sum(cnts(1:2))+1) + 5/12, ...
            y(rolls+3), ...%
618         y(rolls+4) , ...
            y(rolls+1)+20, ...
620         y(rolls+2)+20
            ]';
622     isterminal = [1 1 1 1]';   % stop the integration
        direction = [-1 1 -1 -1]';   % [positive negative] direction
624     end
        function z = steadystate(x)
626         c = state_derive(0,[...
                ...%Initial_Vel(1)
628             x(1:9,1);
```

```
            Initial_Vel(10:11)'; ...
630         x(12:15);...
            Initial_Vel(16);...%x(16);...
632         x(17:rolls);...
            x(rolls+[1:2]);...
634         (f_da*5.032/12-x(29)*cos(theta_on(4))*(15.925-1.5)/12)/...
                ((15.925+1.5)/12)/cos(pi-theta_in(4));... %span tensions
636         x(25+[4:9]);...
            x(25+[9,11])'*[.5 .5]';...
638         x(25+[11:13]);...
            2*lineten-x(25+15);...
640         x(25+[15:16]);...
            2*lineten2-x(25+18,1);...
642         x(25+[18:24],1);
            0;... %dancer pos
644         0;... %dancer vel
            x(eccent)/Rn(eccent);... %omega(1)
646         ],...%
            x(sum(cnts(1:4))+[1:cnts(5)]));
648     z = c;
        %z(5) = Initial_Vel(5);
650     %z(9) = f_da-z(10);
        %z(12) = 2*lineten-z(13);
652     %z(14:15) = [0 0];
        z(sum(cnts(1:4))) = x(sum(cnts(1:4))) - z(eccent)/Rn(eccent);
654     %z(17:19) = x(17:19);
    end
656 %-------------------------------------------------------------------------
    %      B E A R I N G   F R I C T I O N
658 %-------------------------------------------------------------------------
    function bv = FrictionCalc(v,t,op1,val1,op2,val2)
660     % calculate bearing friction and parasitic torque if applicable
        if nargin == 6
662         switch op2
                case 'sliding_friction'
664                 val2 = Bsldfn'.*Rn' ;
                case 'Parasitic_Torque'
666                 val2 = Rn'.*ParasiticTorque';
                case 'viscous_friction'
668                 val2 = Bfn'.*v;
            end
670         switch op1
                case 'sliding_friction'
672                 val1 = Bsldfn'.*Rn' ;
                case 'Parasitic_Torque'
674                 val1 = Rn'.*ParasiticTorque';
                case 'viscous_friction'
676                 val1 = Bfn'.*v;
            end
```

```
678         elseif nargin < 5
                val2 = v*0;
680             switch op1
                    case 'sliding_friction'
682                     val1 = Bsldfn'.*Rn' ;
                    case 'Parasitic_Torque'
684                     val1 = Rn'.*ParasiticTorque';
                    case 'viscous_friction'
686                     val1 = Bfn'.*v;
                end
688         elseif nargin < 3
                val1 = v*0;
690         end
            tens = ([t; tf]-[t0; t]);
692         test1 = -(val1 + val2) + Rn'.^2.*(tens);
            m = size(v,1);
694         bv = zeros(m,1);
            for i= 1:m
696             if v(i)<=0 && test1(i)<= 0
                else
698                 bv(i) = -(val1(i) + val2(i)) + Rn(i)^2*(tens(i));
                end
700         end
        end
702 %-------------------------------------------------------------------------
    %      P O S T   P R O C E S S I N G
704 %-------------------------------------------------------------------------
        function [time, zppout, Events] = PostProcessedSlip(time,z)
706         %go through all the data and check for slip the normal way.  Update
            %the tensions if slip is found.
708         %
            % Requires an initialized RecordEvents function.
710         funnames{1} = @NonSlipVelDer;
            funnames{2} = @span2;
712         funnames{3} = @span2;
            Ind(20) = 0;
714         flipped = 0; %transpose boolean
            m = size(z);
716         if m(1) > m(2)
                %for processing during operation
718             z = z';
                flipped = 1; %it was flipped
720         end
            m = min(m); %max(m); %during vs. post processing
722         if nip10
                %if roller 10 is being modeled with a nip, do not consider it to
724             %slip.
                rollers = [2 3 4 5 6 7 8 9 ];
726         else
```

```
        rollers = [2 3 4 5 6 7 8 9 10];
    end
    if nip17
        %if roller 17 is being modeled with a nip, do not consider it to
        %slip.
        rollers = [rollers 11:15 17:rolls-1 ];
    else
        rollers = [rollers 11:16 17:rolls-1];
    end
    IND = zeros(m,numel(rollers)+1);
    etime = [];
    ez = [];
    eevent = [];
    zppout = z;
    slip_flagged = 0; %boolean to indicate that slip was found in the set
    ind_saved = 0*IND(1,:);
    I=0;
    while I<4
    for i = 1:m
       for j = 1:numel(rollers)
           p = rollers(j);
           test1 = z(i,rolls+p)-z(i,rolls+p-1)*exp(-Mun(p)*...
                theta_wrap(p));
                %<0, web slower than roller
           test2 = z(i,rolls+p)-z(i,rolls+p-1)*exp(Mun(p)*...
                theta_wrap(p));
                %>0, web faster than roller
           if test1 < 0
               IND(i,j) = -1;
               slip_flagged = 1;
           elseif test2 > 0
               IND(i,j) = 1;
               slip_flagged = 1;
           end
       end
       if ind_saved == IND(i,:)
           %indicator set is unchanged from one round of the while loop
           %to the next
           I=10;
           break
       else
           ind_saved = IND(i,:);
       end
           %apparent indicator set complete
           %
       if slip_flagged && numel(find(IND(i,:)~=0))>1
           for j = 1:numel(rollers)
               p = rollers(j);
               %test Gamma if slip_flagged true
```

```
776                     if abs(IND(i,j+1)-IND(i,j)) == 2
                            %from Whitworth Ph.D. thesis pp. 97 of pdf (73 by
778                         %pagenumber)
                        Gamma = log(z(i,rolls+p+1)/z(i,rolls+p-1))*IND(i,j) + ...
780                         (Mun(p+1)*theta_wrap(p+1) - Mun(p-1)*theta_wrap(p-1));
                        if Gamma > 0
782                         IND(i,j+1) = 0;
                        elseif Gamma < 0
784                         IND(i,j) = 0;
                        end
786                     end
                    end
788             end
                %
790             for j = 1:numel(rollers)
                    p = rollers(j);
792                 if IND(i,j) == 0
                    else
794                     if p == 20
                            %predict DG traction assuming PR speed is web speed
796                         mu = DG.predict(z(i,16),z(i,p),z(i,rolls+p),Mun(p));
                        else
798                         %use constant coeff.s elsewhere...
                            mu = Mun(p);
800                     end
                        zppout(i,rolls+p-1) = Whitworth(z(i,rolls+p),z(i,rolls+...
802                         p-1),L(p),L(p-1),IND(i,j)*mu*theta_wrap(p));
                        zppout(i,rolls+p) = zppout(i,rolls+...
804                          p-1)*exp(IND(i,j)*mu*theta_wrap(p));
                        if SFDR && p == 20
806                         funnames{1} = @SlipIdlerVelDer;
                            funnames{2} = @SlipTensionIncoming;
808                         funnames{3} = @SlipTensionOutgoing;
                            Ind(20) = IND(i,j);
810                     end
                    end
812             end
            end
814         I = I+1;
        end
816     if slip_flagged
            RecordEvents(time,z,IND(1,1:end-1))
818     end
        if nargout>2
820         %need for loop
            %save non-slip state for the event
822                 etime = [etime time(i)];
                    ez(i,:) = z(i,:);
824                 eevent = [eevent IND(i,j)*p];
```

```
            Events = {etime,ez,eevent};
        end
        if flipped
            zppout = zppout';
        end
    end
        function newTen = Whitworth(T,Tnm1,L,Lnm1,INDmualpha)
        %applies the Whitworth equation to the tensions
        %INDmualpha = IND(i,j)*Mun(p)*theta_wrap(p) where IND(i,j) is
        %the +/-1 value from the test1 or test2 above, Mun is the
        %coefficient of friction for roller p, and theta_wrap(p) is the
        %wrap angle in radians for roller p.
            newTen = (Tnm1*Lnm1+T*L)/(Lnm1+L*exp(INDmualpha));
        end
        function phi = PHIi(phi_nm1,ind_n,mu_n,alpha_n)
            %Multiplier in slip calculation
            phi = phi_nm1*exp(ind_n*mu_n*alpha_n);
        end
%------------ Plotting ------------------------------------------------------
    function l = legendmaker(list,prefix,postfix)
        m = numel(list);
        l = cell(m,1);
        if nargin<3
            postfix = [];
        end
        for i = 1:m
            l(i) = {[prefix num2str(list(i)) postfix]};
        end
    end
%% ---------------------
% P L O T T I N G
%---------------------

velocities_out = z(:,1:rolls)*60; %fpm
tensions_out = z(:,rolls+1:rolls+spans);
dancer_out = z(:,rolls+spans+[1])*180/pi;
motor_inputs_out = z(:,sum(cnts(1:4))+[1:cnts(5)]);
% Rollers and Spans of interest
velindex = [16 19 20 21 rolls];
tenindex = [1 4 8 11 15 17 18];
waitbar( .6, h)
r1 = IndScurveInput(0,z0(1),deltaV(1),accel_rate,jerk_rate,1);
refspeed = zeros(length(time),1);
shutdown_flag = 0;
for i= 1:length(time)
    t=time(i);
    if (t > 20) && ~shutdown_flag
        % put scheduling info here later
        % for unwind speed
```

```
874             r1 = IndScurveInput(t,400/60,0,accel_rate,jerk_rate,1);
                shutdown_flag = 1;
876         end
            if t> 20
878             r1 = IndScurveInput(t,400/60,0,30/60,30/60);
            else
880             r1 = IndScurveInput(t,z0(1),deltaV(1),30/60,30/60);
            end
882         refspeed(i) = r1(1);
    end
884 figure(2);
    subplot(2,2,1)
886 plot(time,velocities_out(:,velindex)-repmat(refspeed,1,5)*60)
    title('velocity');
888 legs = legendmaker(velindex,'R ');
    legend(legs)
890 grid;

892 subplot(2,2,2)
    plot(time,tensions_out(:,tenindex))
894 title('tension');
    legs = legendmaker(tenindex,'t ');
896 legend(legs);
    grid;
898 subplot(2,2,3)
    plot(time,dancer_out);%,time,dancer_out(:,1))
900 title('dancer');
    legend('x (deg)','location','best')%,'xdot');
902 grid;
    subplot(2,2,4)
904 plot(time,motor_inputs_out);%,time,dancer_out(:,1))
    title('Motor Inputs');
906 legend('Unwind','SWL','SWF','PR','Rewind','location','best')%,'xdot');
    ylabel('Amps')
908 grid;
    waitbar( 1, h)
910 if nargout>4
    %     if ismember('EEvents',who)
912 %         Events = EEvents;
    %     else
914 %     Events = {teout, yeout, ieout};
    %     end
916     Events = RecordEvents(0,0,0,2);
    end
918 close(h)
    clear RecordEvents DG
920 end
```

## J.3 Gain Calculation Simulation

The next three simulations were created for [87]. They execute at steady-state and have three pairs of unwind and rewind roll diameters to select for a simulation. The code in Listing J.1 simulates the EWL with fixed tension gains while varying the speed gains based on roll diameter. The code in Listing J.2 computes gains for each loop separately. The code in Listing J.3 computes the tension gains with speed loop included per the RA method.

### J.3.1 Gain Calculation with Fixed Tension Gains and Variable Speed Gains

Listing J.1: The GainSched08 Code

```
function [ SS, SSder, time, z, Events ] = GainSched08( tfinal,case_ds )
%Euclid line simplified Slip model with 5 rollers unwind, idler, driven,
% idler, rewind.  Added eccentricity to driven #3 roll.  Making the roller
% velocity derivative and span tension derivative all callable functions.
% Then, will change which one is called based on the condition of the event
% for  slip.  This model initially (and all previous versions: 01, 02, 03)
% used the diameter instead of the radius.  Version 05 amends that.
%   Detailed explanation goes here
%
%   Version 16 - Using 15 and updating the frictions and bearing drags for
%   the new arrangement of rollers.  This is from the SlipModel16()
%   function.
%
%   Version 01 - Took SlipModel16() and removed worries about slip.  Added
%   gain calculation methods from Gain Calc paper and from Euclid line in
%   order to use Pramode's gains.
%
%   Version 01a - Copied 01, Includes time varying radii. Set up for load
%   cell control at rollers 9 and 19.
%
%   Version 01b - Copied 01a, Added simulation of a shutdown. Fixing up the
%   dancer control side of the script.
%
%   Version 02 - Copied 01b,  Going to apply pure tension control to unwind
%   and rewind motors following RS paper.
%
%   Version 03 - Copied 02,  Going to apply 2rev diameter check
%
%   Version 04 - Copied 03, Converted to linearized dynamics, load cells
%   moved to initial idle roller. Method 1
%
%   Version 05 - Copied 04, Still using linearized dynamics, apply my
%   method to find tension gains.
```

```
%
%   Version 06 - Copied 04, going to add speed control loop to the mix,
%   gains calc'ed as a tension loop by itself and a speed loop by itself
%
%   Version 07 - Copied 06, going to calc tension gains including effect of
%   speed gains.  Used Gain Calc paper's method.  Added case_ds to select
%   from the function call what diameter pair to simulate. Method 2
%
%   Version 08 - Copied 07, Now going to fix tension gains and only vary
%   the speed controller gains.  Method 3
%
%
%

if nargin <1
    case_ds = 1;
    tfinal = 30;
elseif nargin <2
    case_ds =1;
end

file='C:\Users\quietman\Documents\MATLAB\WTS\EUCLID_FULL_LINE2.Dat';

dancers = 1; %number of dancers in system
%                           filename, number of rolls, number of spans
[r,s,c]=ElementPropReadIn(file,25,24);
Rolls = r{1};
Rolls = Rolls(:,:);
Spans = s{1};
Spans = Spans(:,:);
funnames = {@NonSlipVelDer,@Span2,@Span2,@Span2,@Span2};

RecordEvents(0,0,0,1);

%---------- Run Parameters --------------------
rolls = 25;
spans = 24;
deltaV = ones(1,rolls)*1/60; %increase in speed in ft/s
Initial_Vel = ones(1,rolls)*100/60; %initial speeds of rollers in ft/s
%Initial_Vel(17) = 399.7/60;
deltaV = deltaV*4/4;
Initial_Vel = Initial_Vel*4/4;
t0 = 10; %lbf wound in tension
tf = 10.; %outgoing tension in lbf
lineten = 10; % line tension setpoint
lineten2 = lineten; %line tension for process section (unmeasured)
lineten3 = 10; %line tension for rewind section (Tension ON case)
dancer = [];
acc = [];
```

```
unwind = [];
eccent = [];
Ind = zeros(rolls,1);
GR = 1./[3.795 13.95 13.95 7.006 3.795];  %gear ratios for drives

LC_unwind_control = 1; %set to 1 for unwind to be controlled by the load
%   cell at roller 9, set to 0 for dancer control
if LC_unwind_control
    cnts = [rolls spans 2 2 5]; %[25, 24, 2variable radii, 2eccent roller,
    %5motor currents]
else
    %dancer controlled
    %[25 24, 2dancer states, 2variable radii, 2eccent roller, 5motor
    % currents]
    cnts = [rolls spans 2 2 2 5];
    dancer = [4];
end
slipind = 3;
nip10 = 1; % 1 use nip, 0 don't use nip

g = 32.174; %ft/s^2 gravity
slipflag = 0; % signal to indicate that there is slip...
y0 = zeros(40,1); %y0 = [y0; -0.005];
simrun = 0;
%---------- Roller params ----------------------

L = Spans(4,:)/12; %feet
width_w = Spans(3,1)/12;%feet
rho_w = Spans(5,:)*12^3/g;
%lbm/in^3*(12in)^3/ft^3 / (lbm*ft/(s^2*lbf)) = lbf*s^2/ft^4
A = Spans(2,1)/1000/12 * width_w; %ft^2
E = Spans(1,1)*1000*12^2;%lbf/ft^2

%Rollers
Rn = Rolls(3,:)/12/2;% radius in ft for all rollers
Jn = Rolls(13,:); %lbf*ft*s^2
Cmn = Rolls(19,:); % Torque/speed constant of a motor
Kmn = Rolls(18,:); % Torque constant of a motor, lbf*ft/amp
Bfn = Rolls(20,:)*(1-0); % lbf*ft*sec, Bearing friction   ----------<
theta_arm = Rolls(11,:)*pi/180; % dancer arm angle from horizontal in rad.
Mn = Rolls(10,:)/g; %slugs
Mun = [ones(1,17)]*.2403; % static friction max from measurements
Mun([9 13]) = [0.1185 0.15];
e_3 = 0.0/12; %ft eccentricity
Jpn = 0.209; %inertia of the dancer arm

Kmn([1 rolls]) = [Kmn(1)*GR(1) Kmn(rolls)*GR(5)];
Cmn(1) = .001/GR(1);
Cmn(rolls) = 0.001/GR(5);
```

```
Cdn = [zeros(1,3) 0.15 0 0 0 0 0 0 0 0];
Kn = [zeros(1,3) 0 0 0 0 0 0 0 0 0];
f_da = 33.86; %36.2lbf external force on dancer.
%   34 works out to 5.36lbf in the unwind section
Rn_n = [zeros(1,5) 0 0 0 0 3 0 0]/24; %Nip radius in feet
Jn_n = [zeros(1,5) 0 0 0 0 4.635e-3 0 0 ]; %nip inertias
Bfn_n = [zeros(1,5) 0 0 0 0 1 0 0]*0.0006; %nip bearing friction

Bfn = [0.00060729 0.00016691 0.00126336 0.00017803 0.00004665 ...
    0.00002655 0.00050257 0.00062522 0.00061941 0.00049733  ...
    0.00061733*ones(1,15)] ;

bearing_sliding_friction = 0;  %boolean to toggle between
%                       viscous and sliding friction models
Bsldfn = [.1 0.05227297 0.41489625      0.17357908      ...
    0.23150706  0.13421975 0.16057590   0.20012461 ...
    0.20024913 .1 .1 0.15779922*ones(1,14)]; %mu's from SpinDownTest.m
Bsldfn = Bsldfn*diag([38*(1.5/24) 12.89*.915/12*[1 1] ...
    1.188*.915/12 .876*.915/12*[1 1] 12.89*.915/12*[1 1 1] ...
    78.454*.915/12*[1 1] 12.89*.915/12*[1 1 1 1 1] 42*1.5/24 ...
    12.89*.915/12*[1 1 1 1 1 1 1] 38*(1.5/24)])*.05;
theta_arm = zeros(1,25);
theta_arm(4) = pi/2;
Larm = [zeros(1,3),15.925,zeros(1,13)]/12;
ParasiticTorque = [0 0*Rn(2) 0 0.00*Rn(4) 0 0 0 0 Rn(9)*0 zeros(1,16)];

%------------ Span params ------------------------
A = 6.07/12*0.00536/12; % cross-sectional area in ft^2
rho = 1.0740000E-0002/g*144*12 ; %density of web in slug/ft^3
rot_dir = [-1 1 -1 1 -1 -1 -1 1 -1 -1 1 1 -1 1 -1 ...
    1 -1 1 -1 1 -1 1 -1 1 -1 ]; %forward web direction CW = -1, CCW = 1
[sdta, theta_in,theta_on] = PrintLine(file,25,rot_dir,5,theta_arm);
theta_in=theta_in(1:rolls);
theta_on = theta_on(1:rolls);
theta_wrap = WrapAngle(sdta,Rn);

if LC_unwind_control
    %Gain calc paper LC control gains zeta=0.9, t_r=0.3 sec
%unwind tension loop gains

case_d = [18.5/24 7.483/12 3.315/12 18.5/24 18.5/24;...  %Rn(1)
          3/24    6/12     9/12     3/24    3/24; ...    %Rn(rolls)
          18.5/24 7.483/12 3.315/12 7.483/12 3.315/12;...  %Rn(1) fixed
          3/24    6/12     9/12     6/12     9/12; ...    %Rn(rolls) fixed
          ];
% Rn(1) = 18.5/24; %set to bare roller on rewind, unwind:full roll
% Rn(1) = 7.483/12; %set to 6in on rewind, unwind:7.483in
% Rn(1) = 3.315/12; %set to 9in on rewind, unwind:3.315in
%case_ds = 1;
```

```
    Rn(1) = case_d(1,case_ds);
182 %--------Gain Scheduled Speed
    temp = GainSched(Rn(1),100/60,.9,20.5,1);
184 % negative sign due to loadcell being down stream of controlled motor
    Kp(1) = temp(1);
186 Ki(1) = temp(2)/1;
    Kd(1)=0;%
188 %unwind speed loop gains
    Kp(2)= temp(3);
190 Ki(2)= temp(4);
    Kd(2)=0.0;
192 %Driven speed loop gains
    Kp(3) = 5.5421; %
194 Ki(3) = 32.0624;
    Kd(3) = 0;
196 %Swrap Follow speed loop gains
    Kp(4) = 5.5421;%
198 Ki(4) = 32.0624;%
    Kd(4) = 0;%0;
200 %Pull Roll speed loop gains
    Kp(5) = 5.9853;%
202 Ki(5) = 34.6265;%
    Kd(5) = 0;%0;
204 %rewind tension loop gains

206 Rn(rolls) = case_d(2,case_ds);
    %--------Gain Scheduled speed
208 temp = GainSched(Rn(rolls),100/60,.9,20.5,rolls);
    Kp(6) = temp(1);
210 Ki(6) = temp(2)/1;
    Kd(6) = 0.0;%0;
212 %rewind speed loop gains
    Kp(7) = temp(3)/1;  %
214 Ki(7) = temp(4)/1;
    Kd(7) = 0.0;%0;
216 else
        %Gain Calc paper Dancer Control zeta=0.9, t_r=0.3sec
218 %unwind tension loop gains
    Kp(1)=0.4735/10;
220 Ki(1)= 0.2742;
    Kd(1)=0;%
222 %unwind speed loop gains
    Kp(2)=2.7928;
224 Ki(2)=16.1576;
    Kd(2)=0.0;
226 %Driven speed loop gains
    Kp(3) = 5.5421;
228 Ki(3) = 32.0624;
    Kd(3) = 0;
```

```
%Swrap Follow speed loop gains
Kp(4) = 5.5421;%
Ki(4) = 32.0624;%
Kd(4) = 0;%
%Pull Roll speed loop gains
Kp(5) = 5.9853;%
Ki(5) = 34.6265;%
Kd(5) = 0;%0;
%rewind tension loop gains
Kp(6) = .1718/2;%
Ki(6) = 0.0001552*600;
Kd(6) = 0.0;%0;
%rewind speed loop gains
Kp(7) =  3.91/30;
Ki(7) = 22.6207/40;
Kd(7) = 0.0;
end

Rn(1) = case_d(3,case_ds);
Rn(rolls) =case_d(4,case_ds);
angle_travel_1=0; %stores last unwind position angle
angle_travel_2 =0; %stores last rewind position angle
z0=zeros(58,1);

%------------ Initial Conditions
if LC_unwind_control
    %LC control at roller 9
z = fsolve(@steadystateLC,(...
    rand(58,1)-.5)+[ones(rolls,1)*Initial_Vel(1); ones(spans,1)*lineten;...
        zeros(9,1)],  optimoptions('fsolve','Algorithm',...
        'Levenberg-Marquardt','TolFun',1e-16,'MaxIter',4000,'TolX',1e-6));
z(10:11,1) = Initial_Vel(10:11)';
z(1,1) = Initial_Vel(1)';
z([17,rolls],1) = Initial_Vel([17,rolls])';
z(rolls+8,1) = 2*lineten-z(rolls+9,1);
z(rolls+10,1) = [.5 .5]*z(rolls+[9 11],1);
z(rolls+14,1) = 2*lineten2-z(rolls+15,1);
z(rolls+0+18,1) = 2*lineten3-z(rolls+0+19,1);
%z(20) = (f_da*5.032/12-z(21)*cos(theta_on(4))*(15.925-1.5)/12)/...
    ((15.92+1.5)/12)/cos(pi-theta_in(4));
z(50:51) = [Rn(1) Rn(rolls)]'; % set initial roll diameters
z(52) = z(1)/Rn(1);
z(53) = z(rolls)/Rn(rolls);

else
    %Dancer control at roller 4
    z = fsolve(@steadystateDan,rand(60,1),  optimoptions('fsolve',...
      'Algorithm','Levenberg-Marquardt','TolFun',1e-16,'MaxIter',...
      4000,'TolX',1e-6));
```

```
z(1,1) = Initial_Vel(1);
z(10:11,1) = Initial_Vel(10:11)';
z([17,rolls],1) = Initial_Vel([17 rolls])';
z(rolls+10,1) = [.5 .5]*z(rolls+[9 11],1);
z(rolls+14,1) = 2*lineten2-z(rolls+15,1);
z(rolls+18,1) = 2*lineten3-z(rolls+19,1);
z(rolls+3,1) = (f_da*5.032/12-z(rolls+4)*cos(theta_on(4))*(15.925-1.5)/12)...
    /((15.92+1.5)/12)/cos(pi-theta_in(4));
z(rolls+spans+[2],1) = [0]';
z(52:53) = [Rn(1) Rn(rolls)]'; % set initial roll diameters
z(54,1) = z(1)/Rn(1);
z(55,1) = z(rolls)/Rn(rolls);
end

velocities_ss = z(1:rolls,1);
%velocities_ss(driven) = Initvel(driven);
tensions_ss = z(rolls+[1:spans]);% 3.4*ones(14,1); z(rolls+[20:22])];
if LC_unwind_control
    %nothing
    dancer_ss = [];
    motors_ss = z(54:58);
else
    dancer_ss = [z(50) 0]';
    motors_ss = z(56:60);
end

% Solve stuff
if LC_unwind_control
    z0 = [velocities_ss; ...
        tensions_ss; ...
        z(50:51);...
        z(52:53);...
        motors_ss]; %7x1
else
    z0 = [velocities_ss; ...
    tensions_ss; ...
    dancer_ss;...
    z(52:53);...
    z(54:55);...
    motors_ss]; %7x1
end

figure(1);
subplot(2,2,1),plot(velocities_ss*60,'>')
title('Velocities')
xlabel('Roller #')
ylabel('fpm')
subplot(2,2,2),plot(tensions_ss,'o')
title('Tensions')
```

```
328 xlabel('Span #')
    ylabel('lbf')
330 %ylim([9.2 10.2])
    if ~LC_unwind_control
332 subplot(2,2,3),bar(dancer_ss*12)
    title('Dancer Position (in) and Velocity (in/s)')
334 xlabel('')
    ylabel('magnitude')
336 end
    subplot(2,2,4),bar(motors_ss)
338 title('Motor Inputs (amps)')
    xlabel('Motor #')
340 ylabel('Magnitude')

342 % Set the output Steady-state
    SS = z0;
344 % Set the output Steady-state derivative
    h =waitbar(0, 'Please Wait');
346 hstep1 = 0.05*tfinal;
    waitbar(0,h);
348 tstart = 0;
    SSder = State_Controller(tstart,z0);
350
    tstart=0:0.005:tfinal;
352 refine = 4;
    %for ode45
354 options = odeset('Events',@events1,'Refine',refine,'RelTol',1e-5);%for
    %for odeBR4
356 %options = {'EventFunction',@PostProcessedSlip;'TimeStep',0.005;...
        'SlipTrue',0};
358 tout = 0;
    zout = z0.';
360 teout = [];
    yeout = [];
362 ieout = [];
    istep = 0; % limit the number of bounces off the full extension or full
364 simrun=1;
    slipflag = 0;
366 flag1 = 0;
    flag2 = 0;
368 shutdown_flag = 0; %for resetting the IndScurve function once

370 timetest = 0.005; % ---- updated each timestep to compare and
    %       current time and wait until the next one to switch equations
372 %           close situation to 150 (arbitrary).
    while tout(end) < tfinal && istep < 5500
374
        [time,z] = ode45(@State_Controller,...
376         tstart([1 end]),z0);%,options); ,te,ye,ie
```

```
ie=[];
    nt = length(time);
  tout = [tout; time(2:nt)];    % dummy variable to store data
  zout = [zout; z(2:nt,:)];     % cummy variable to store data
%  teout = [teout; te];          % Events at tstart are never reported.
%  yeout = [yeout; ye];  %values of event function (1 of them will be 0 )
  if size(ie,1)>1
    ieout = [ieout; ie+10*slipflag];  %
  else
      ieout = [ieout; ie'+10*slipflag];
  end

  if time(nt) < tfinal %check to see if integration needs to continue
      %find which event caused the stop of integration
      if isempty(ie)
          options = odeset(options,'MaxStep',0.005);
          tstart1 = [time(nt) (.005-rem(time(nt),.005))+time(nt)];
          tstart = [tstart1(1) tstart1(2):0.005:tfinal];

          [time,z] = ode45(@State_Controller,...
        tstart,z(end,:),options);
         nt = length(time);
     tout = [tout; time(2:nt)];    % dummy variable to store data
    zout = [zout; z(2:nt,:)];     % dummy variable to store data
    teout = [teout; te];          % Events at tstart are never reported.
    yeout = [yeout; ye];  %values of event function (1 of them will be 0 )
    ieout = [ieout; ie];  %
      else
           z0 = z(nt,:);
          for i=1:numel(ie)
              eventcase = ie(i);
        if slipflag == 0
              switch eventcase;
          case 1
              %T3 < T2*exp(-Mun*theta_wrap(3))
              Ind(slipind) = -1;
              options = odeset(options,'MaxStep',.005); %@events2);
                nm1=slipind-1;
                n = slipind;
              z0(rolls+nm1) = (z0(rolls+nm1)*L(nm1)+z0(rolls+n)*...
                  L(n))/(L(nm1)+L(n)*exp(Ind(n)*Mun(n)*theta_wrap(n)));
              z0(rolls+n) = z0(rolls+nm1)*exp(Ind(n)*Mun(n)*theta_wrap(n));
              %slipflag = 1;
          case 2
              %T3 < T2*exp(Mun*theta_wrap(3))
              Ind(slipind) = 1;
              options = odeset(options,'MaxStep',.005);%@events2);
                nm1=slipind-1;
                n = slipind;
```

```
426                 z0(rolls+nm1) = (z0(rolls+nm1)*L(nm1)+z0(rolls+n)*...
                        L(n))/(L(nm1)+L(n)*exp(Ind(n)*Mun(n)*theta_wrap(n)));
428                 z0(rolls+n) = z0(rolls+nm1)*exp(Ind(n)*Mun(n)*theta_wrap(n));
                    %slipflag = 1;
430                 end
              else
432                 switch eventcase;
               case 1
434                 % Returning from slip condition 1
                    funnames{1} = @NonSlipVelDer;
436                 funnames{2} = @Span2;
                    funnames{3} = @Span2;
438                 Ind(2) = 0;
                    options = odeset('Events',@events1);
440                 %slipflag = 0;
               case 2
442                 % Returning from slip condition 2
                    funnames{1} = @NonSlipVelDer;
444                 funnames{2} = @Span2;
                    funnames{3} = @Span2;
446                 Ind(2) = 0;
                    options = odeset('Events',@events1);
448                 %slipflag = 0;
                    end
450             end
                    switch eventcase
452             case 3
                    istep = 5500;
454             case 4
                    istep = 5500;
456                 end
              end
458         end
            z0=z0';
460         tstart1 = [time(nt) (.005-rem(time(nt),.005))+time(nt)];
            tstart = [tstart1(1) tstart1(2):0.005:tfinal];
462         istep = istep + 1;
      end
464 end

466 time = tout; % Return expected value names
    z = zout;  % Return expected value names
468
    %-----------------------------------------------------------------------
470 %           S T A T E   D E R I V A T I V E   F U N C T I O N
    %-----------------------------------------------------------------------
472     function xdot = state_derive(T,x,u)

474         v = x(1:rolls,1);
```

```
        t = x(rolls+[1:spans],1);
476 %         theta1 = mod(x(sum(cnts(1:3))+1,1),2*pi());
    %         [r_1, r_1dot,r_1ddot,r_1ddot2] = Rollshape(theta1,...
478 %             x(sum(cnts(1:3))+2),Rn(unwind),rshape);
        xdot = x*0;
480
        % ---- roller velocity derivs General
482     if bearing_sliding_friction
            xdot(1:rolls,1) = 1./Jn'.*(-Bsldfn'.*Rn' + Rn'.^2.* ...
484             ([t; tf]-[t0; t])- Rn'.*ParasiticTorque');
        else
486         xdot(1:rolls,1) = 1./Jn'.*(-Bfn'.*v+ Rn'.^2.* ...
               ([t; tf]-[t0; t]) - Rn'.*ParasiticTorque');
488     end
        % specific derivatives for certain rollers
490     xdot(9,1) = feval(funnames{1},t(8:9),v(9),9); %5-24-19 uses
        %                 viscous friction model
492
        if LC_unwind_control
494         Jn(1) = inertia(x(50));
        xdot(1,1) = (1/Jn(1)*(-(Bfn(1)+Cmn(1))*v(1)/(2*pi*...
496         GR(1)*x(50))*GR(1)+...
            x(50)*(t(1)*GR(1)) + Kmn(1)*(u(1)) + Mn(1)*e_3*g...
498         *sin(x(rolls+spans+3,1))))*(2*pi*GR(1)*x(50)); % sin(T*omega1));
        else
500         %dancer control added 2 states
        xdot(1,1) = (1/0.056*(-Bfn(1)*v(1)/(2*pi*GR(1)*x(52))*GR(1)+...
502         x(52)*(t(1)*GR(1)) + Kmn(1)*(u(1)) + Mn(1)*e_3*g...
            *sin(x(sum(cnts(1:3))+1,1))))*(2*pi*GR(1)*x(52));
504 %         xdot(1,1) = 1/Jn(1)*(-Bfn(1)*v(1)+ x(52)^2.*(t(1))...
    %         +Kmn(1)*(u(1)*x(52))+x(52)*Mn(1)*e_3*g...
506 %           *sin(x(rolls+spans+5,1)));
        end
508     if nip10
            xdot(10,1) = 1/(Rn(10)^2*Jn_n(10)+Rn_n(10)^2*Jn(10))*( ...
510             -(Rn_n(10)^2*Bfn(10)+Rn(10)^2*Bfn_n(10))*v(10) + ...
                Rn_n(10)^2*Rn(10)^2*(t(10) - t(9))+Rn_n(10)^2*...
512             Rn(10)*Kmn(10)*u(2));
        else
514         xdot(10,1) = xdot(10,1) + 1/Jn(10)*(Kmn(10)*(u(2)*Rn(10)));
        end
516         xdot(11,1) = xdot(11,1) + 1/Jn(11)*(Kmn(11)*(u(3)*Rn(11)));
            xdot(17,1) = 1/(Rn(17)^2*Jn_n(10)+Rn_n(10)^2*Jn(17))*( ...
518             -(Rn_n(10)^2*Bfn(17)+Rn(17)^2*Bfn_n(10))*v(17) +  ...
                Rn_n(10)^2*Rn(17)^2*(t(17) - t(16))+Rn_n(10)^2*...
520             Rn(17)*Kmn(17)*u(4));
        if LC_unwind_control
522         Jn(rolls) = inertia(x(51));
            xdot(rolls,1) = (1/Jn(rolls)*(-(Bfn(rolls)+Cmn(rolls))*...
```

```
524                 v(rolls)/(2*pi*GR(5)*x(51))-x(51)*(t(spans)*GR(5))+...
                    Kmn(rolls)*(u(5))))*GR(5)*2*pi*x(51);
526         else
                xdot(rolls,1) = (1/.055*(-Bfn(rolls)*v(rolls)/(2*pi*x(53))-...
528                 x(53)*(t(spans)*GR(5))+Kmn(rolls)*(u(5))))*GR(5)*2*pi*x(53);
    %           xdot(rolls,1) = 1/Jn(rolls)*(-Bfn(rolls)*v(rolls)-...
530 %               x(53)^2.*(t(spans))+Kmn(rolls)*(u(5)*x(53)));
        end
532     if ~LC_unwind_control
        % ---- Dancer derivs
534     xdot(rolls+spans+1,1) = x(rolls+spans+2,1);
        xdot(rolls+spans+2,1) = 1/(Jpn)*(f_da*5.032/12 - ...  %/(Mn(dancer))
536         t(3)*cos(pi-theta_in(dancer))*(15.925+1.5)/12  -...
            t(4)*cos(theta_on(dancer))*((15.925-1.5)/12) - ...
538         Kn(4)*x(rolls+spans+1) - Cdn(4)*x(rolls+spans+2) - ...
                13.95/12*.317*g*sin(x(rolls+spans+1)));
540     end
        % ---- tension derivs
542     tens = [t0; t]; %include wound in tension
        for i = 1:spans
544         if ~LC_unwind_control && i == dancer
                [temp(3), temp(4)] = PendDancerLeng(x(sum(cnts(1:2))+...
546                 1),-x(sum(cnts(1:2))+2),L(i),Larm(i),...
                    1.38/12,1.5/12,1.5/12); %.24
548             sp0 = [v(i+[0 1])', tens(i+[0 1],1)', L(i)+...
                    temp(3), ... %Length does not change
550                     temp(4), theta_on(i)]';
            elseif ~LC_unwind_control && i+1==dancer
552             [temp(1), temp(2)] = PendDancerLeng(x(sum(cnts(1:2))+1)...
                    ,-x(sum(cnts(1:2))+2),L(i),Larm(i+1),...
554                 3.084/12,1.5/12,1.5/12);
                sp0 = [v(i+[0 1])', tens(i+[0 1],1)', L(i)+...
556                 temp(1),... %Length does not change
                    temp(2), theta_in(i+1)]';
558
            elseif i == unwind
560             % d1 calc'ed on line 24
                l1 = sqrt(d1 - (r_1+Rn(2))^2);
562             dl1 = -(r_1+Rn(2))*r_1dot*theta1dot/sqrt(d1-...
                    (r_1+Rn(2))^2); %y0(rolls+1,1)
564             sp0 = [v(i+[0 1])',tens(i+[0 1],1)',l1,dl1]';
            else
566             sp0 = [v(i+[0 1])' tens(i+[0 1],1)' L(i) 0 theta_on(i)]';
            end
568         % for span2Lin sp0 = [sp0(from above) vel1_ss vel2_ss]
            sp0 = [sp0; z0(i+[0 1])];
570         xdot(rolls+i,1) = span2Lin(T, sp0, E, A);
        end
572
```

```
        if LC_unwind_control
574         xdot(rolls+spans+[1 2],1) = [-0.00536/12/(2*pi)...
                *(x(1,1)/x(50));...%(x(rolls+spans+3)-angle_travel_1);...
576             0.00536/12/(2*pi)*(x(rolls,1)/x(51,1))];

578         %angular speeds
            xdot(rolls+spans+3,1) = x(1,1)/x(50);
580         xdot(rolls+spans+4,1) = x(rolls,1)/x(51);
        else
582         %dancer controlled
            xdot(rolls+spans+2+[1 2],1) = [-0.00536/12/(2*pi)...
584             *(x(1,1)/x(52));...
                0.00536/12/(2*pi)*(x(rolls,1)/x(53))];
586
            %angular speeds
588         xdot(rolls+spans+5,1) = x(1,1)/Rn(1);
            xdot(rolls+spans+6,1) = x(rolls,1)/Rn(rolls);
590     end
        % xdot = xdot';
592

    end
594 %-------------------------------------------------------------------------
    %           S T A T E   C O N T R O L L E R   F U N C T I O N
596 %-------------------------------------------------------------------------
    function [ydot] = State_Controller(t,y)
598     %[t,y] = PostProcessedSlip(t,y);
        ydot = y*0;
600     % Setup inputs
%   Steady-state operation only...
602     if t<0.5
            %Step in Tension
604     r1 = [100 0 0]'/60;
        r2 = r1;
606     else
            %Step in Tension
608         r1 = [100 0 0]'/60;
            r2 = r1;
610         lineten = 11;
        end
612     % get state derivative
        if LC_unwind_control
614         % update the steady-state velocities with r1 and r2
            z0(1,1) = r1(1);
616         z0(2:rolls,1)= r2(1)*ones(rolls-1,1);
            ydot(1:sum(cnts(1:4)),1) = state_derive(t,y(1:sum(cnts(1:4)),...
618             1),y(sum(cnts(1:4))+[1:cnts(5)],1));
            %tension errors
620         F(1)=[.5 .5]*y(rolls+[8 9],1);
            Fdot(1)=[.5 .5]*ydot(rolls+[8 9],1);
```

```
622             F(2)=[.5 .5]*y(rolls+0+[18 19],1);
                Fdot(2)=[.5 .5]*ydot(rolls+0+[18 19],1);
624             %speed errors
                errors = [ r1(1)-y(1), r2(1)-y(10), r2(1)-y(11), r2(1)-y(17),...
626                 r2(1)-y(rolls);...
                    r1(2)-ydot(1), r2(2)-ydot(10), r2(2)-ydot(11), r2(2)-...
628                 ydot(17), r2(2)-ydot(rolls)];
                %trim speed calc
630             TrimV(1) = (Kp(1)*(-Fdot(1))+Ki(1)*(lineten-F(1)));
                TrimV(2) = (Kp(6)*(-Fdot(2))+Ki(6)*(lineten-F(2)));
632             %current calc
                ydot(sum(cnts(1:4))+1,1) = (Kp(2)*(errors(2,1)+TrimV(1)*0)+...
634                 Ki(2)*(errors(1,1)+TrimV(1)));
                ydot(sum(cnts(1:4))+2,1) = Kp(3)*errors(2,2)+Ki(3)*errors(1,2);
636             ydot(sum(cnts(1:4))+3,1) = Kp(4)*errors(2,3)+Ki(4)*errors(1,3);
                ydot(sum(cnts(1:4))+4,1) = Kp(5)*errors(2,4)+Ki(5)*errors(1,4);
638             ydot(sum(cnts(1:4))+5,1) =  Kp(7)*(errors(2,5)+TrimV(2)*0)+...
                    Ki(7)*(errors(1,5)+TrimV(2));
640         else
                %dancer Controlled
642             ydot(1:sum(cnts(1:5)),1) = state_derive(t,y(1:sum(cnts(1:5)),...
                    1),y(sum(cnts(1:5))+[1:cnts(6)],1));
644
                F(1)= -y(rolls+spans+1,1);
646             Fdot(1)=-y(rolls+spans+2,1);
                F(2)=[.5 .5]*y(rolls+[18 19],1);
648             Fdot(2)=[.5 .5]*ydot(rolls+[18 19],1);

650             % find errors in speed
                errors = [ r1(1)-y(1), r2(1)-y(10), r2(1)-y(11), r2(1)-...
652                 y(17) r2(1)-y(rolls);...
                    r1(2)-ydot(1), r2(2)-ydot(10), r2(2)-ydot(11), r2(2)-...
654                 ydot(17) r2(2)-ydot(rolls)];

656             TrimV(1) = (Kp(1)*(Fdot(1))+Ki(1)*(F(1)));
                TrimV(2) = (Kp(6)*(-Fdot(2))+Ki(6)*(lineten-F(2)));
658             %current calc
                ydot(sum(cnts(1:5))+1,1) = (Kp(2)*(errors(2,1)+TrimV(1)*0)+...
660                 Ki(2)*(errors(1,1)+TrimV(1)));
                ydot(sum(cnts(1:5))+2,1) = Kp(3)*errors(2,2)+Ki(3)*errors(1,2);
662             ydot(sum(cnts(1:5))+3,1) = Kp(4)*errors(2,3)+Ki(4)*errors(1,3);
                ydot(sum(cnts(1:5))+4,1) = Kp(5)*errors(2,4)+Ki(5)*errors(1,4);
664             ydot(sum(cnts(1:5))+5,1) = Kp(7)*(errors(2,5)+TrimV(2)*0)+...
                    Ki(7)*(errors(1,5)+TrimV(2));
666         end

668         y0=ydot;
            % update waitbar
670         if t> hstep1
```

```
            waitbar( hstep1/tfinal/2, h)
672             hstep1 = hstep1 +.1*tfinal;
            end
674     end
        %---------- NonSlip idler vel. derive function
676     function dv = NonSlipVelDer(tens,v,ind)
            dv = 1/Jn(ind)*(-Bfn(ind)*v+ Rn(ind)^2*([-1 1]*tens)...
678             - ParasiticTorque(ind)*Rn(ind));
        end
680     %---------- Slip idler vel. deriv function
        function dv = SlipIdlerVelDer(tens,v,ind)
682         Fn = [-cos(theta_in(ind)) cos(theta_on(ind))]*tens(1:2,1);
            if v < 0.001
684             Pars = 0;
            else
686             Pars = ParasiticTorque(ind);
            end
688         dv = 1/Jn(ind)*(-Bfn(ind)*v + Ind(ind)*Mun(ind)*Fn*Rn(ind)^2 ...
                - Pars*Rn(ind));
690     end
        %---------- NonSlip Tension deriv span 1
692     %Just use Span.m like normal
        %---------- Slip Tension deriv span 1
694     function dt = SlipTensionIncoming(Props)
            %Taken from WTS discussion of Turn bars
696         %Props = [L1+L2*e^(mu*alpha), E*A, t0, T2, V1, V3]
            L1 = Props(1); EA = Props(2); Tnm1 = Props(3); Tnp1 = Props(4);
698         Vn = Props(5); Vnp2 = Props(6);
            dt =1/L1*(Vnp2*(EA - Tnp1) - Vn*(EA - Tnm1));
700     end
        %---------- Slip Tension deriv span 2
702     function dt = SlipTensionOutgoing(Props)
            %Taken from WTS discussion of Turn bars
704         %Props = [dT1, exp(mu2*alpha2)];
            dT1 = Props(1); emu = Props(2);
706         dt = dT1*emu;
        end
708     %---------- Events function
        function [value,isterminal,direction] = events1(t,y)
710 % Locate the time when tension ratio violates capstan equation
    % and stop integration.
712     %slipflag = 1;
        %Ind(2)  = 0;
714     nm1 = slipind-1;
        n = slipind;
716     testforTension = find(y(rolls+[1:spans])<0);

718     value = [ ...%y(sum(cnts(1:2))+1) - 50/12, ...
            %y(sum(cnts(1:2))+1) + 5/12, ...
```

```
720         y(rolls+3), ...%
            y(rolls+4) , ...
722         y(rolls+1)+20, ...
            y(rolls+2)+20
724         ]';
        isterminal = [1 1 1 1]';   % stop the integration
726     direction = [-1 1 -1 -1]';   % [positive negative] direction
        end
728
        function z = steadystateLC(x)
730         c = state_derive(0,[...
                Initial_Vel(1);...
732             x(2:9,1);
                Initial_Vel(10:11)'; ...
734             x(12:16);...
                Initial_Vel(17);...
736             x(18:24);... %end of speeds
                Initial_Vel(25);...
738          %   2*lineten-x(27,1);...
                x(26:28);...
740 %             (f_da*5.032/12-x(29)*cos(theta_on(4))*(15.925-1.5)/12)/...
    %                 ((15.925+1.5)/12)/cos(pi-theta_in(4));... %span tensions
742             x(29:32,1);...  %29=span 4
                2*lineten-x(34,1);...  %33 =span 8
744             x(34);...
                [.5 .5]*x([34,36],1);
746             x(36:38);...
                2*lineten2-x(40,1);...
748             x(40:42,1);... %42= span 17
                2*lineten3-x(44,1);...  %43 = span 18
750             x(44:49,1);...
                [Rn(1) Rn(rolls)]';... %variable radii
752             x(1)/Rn(1);... %omega(1)
                x(rolls)/Rn(rolls);...
754             ],...%
                x(54:58)); %motor currents
756         z = c;
            z(52) = x(52) - z(1)/Rn(1);
758         z(53) = x(53) - z(rolls)/Rn(rolls);
        end
760     function z = steadystateDan(x)
            c = state_derive(0,[...
762             Initial_Vel(1);...
                x(2:9,1);
764             Initial_Vel(10:11)'; ...
                x(12:16);...
766             Initial_Vel(17);...
                x(18:24);...
768             Initial_Vel(25);... %end of speeds
```

```
            x(26:27);...
770         (f_da*5.032/12-x(29)*cos(theta_on(4))*(15.925-1.5)/12)/...
                ((15.925+1.5)/12)/cos(pi-theta_in(4));... %span tensions
772         x(29:33,1);...  %29=span 4
%             2*lineten-x(34,1);...  %33 =span 8
774         x(34);...
            [.5 .5]*x([34,36],1);
776         x(36:38);...
            2*lineten2-x(40,1);...
778         x(40:42,1);... %42= span 17
            2*lineten3-x(44,1);...  %43 = span 18
780         x(44:49,1);...
            x(50,1);...
782         0;...
            [Rn(1) Rn(rolls)]';... %variable radii
784         x(1)/Rn(1);... %omega(1)
            x(rolls)/Rn(rolls);...
786         ],...%
            x(56:60)); %motor currents
788     z = c;
        z(54) = x(54) - z(1)/Rn(1);
790     z(55) = x(55) - z(rolls)/Rn(rolls);
    end
792         function newTen = Whitworth(T,Tnm1,L,Lnm1,INDmualpha)
            %applies the Whitworth equation to the tensions
794         %INDmualpha = IND(i,j)*Mun(p)*theta_wrap(p) where IND(i,j) is
            %the +/-1 value from the test1 or test2 above, Mun is the
796         %coefficient of friction for roller p, and theta_wrap(p) is the
            %wrap angle in radians for roller p.
798             newTen = (Tnm1*Lnm1+T*L)/(Lnm1+L*exp(INDmualpha));
            end
800
        function phi = PHIi(phi_nm1,ind_n,mu_n,alpha_n)
802         %Multiplier in slip calculation
            phi = phi_nm1*exp(ind_n*mu_n*alpha_n);
804     end

806 function [k] = GainSched(R,V0,z,om,ind)
        %quickly calc Euclid method for gains.  Need to know the tension
808     %gains first:  calculate based on fixed diameter or make user
        %input?
810     tmp = -TenGainSched(V0,.9,20.5,ind)/1; %go calc fixed gains for
        %     tension loop
812     if ind == rolls
            ind=ind-1;
814         Li =sum(L(ind-[0:5]));
            ind=ind+1;
816         J = inertia(R);
        else
```

```
818             Li = sum(L(1:8));
                J = inertia(R);
820         end
            %speed gains first
822         kp_speed = (-(Bfn(ind)+Cmn(ind))+2*z*om*J)/(GR(1)*Rn(ind)*Kmn(ind));
            ki_speed = (om^2*J)/(GR(1)*Rn(ind)*Kmn(ind)); %GR(1)*Rn(ind)*
824
            k=[tmp(1)/20,tmp(2)/20,kp_speed,ki_speed]/60;
826     end
        function x = TenGainSched(V0,z,om,ind)
828         if ind == rolls
                R = 3/24; %ft for tension calc
830             ind=ind-1;
                Li =sum(L(ind-[0:5]));
832             ind=ind+1;
                J = inertia(R);%1+(.055*R_b^4)*GR(5)^2;
834             num=(Kmn(ind)*A*E*R*GR(1)/sum(Li)/J);
            else
836             R = 15/24; %ft for tension calc
                Li = sum(L(1:8));
838             J = inertia(R);%1+(.055*R_b^4)*GR(1)^2;
                num=-(Kmn(ind)*A*E*R*GR(1)/sum(Li)/J);
840         end
            den = conv([1 V0/Li],[1 (Bfn(ind)+Cmn(ind))/J]);
842         a = [z*om          0       den(2)-2*z*om;...
                 2*om^2*z^2    -num    den(3)-om^2];
844         arf = rref(a);
            kr = arf(1,3);
846         kp = arf(2,3);
            ki = kr*z*om^3/num;  %
848         x=[kp,ki];
        end
850     function J = inertia(R)
           % calculate inertia based on Euclid parames and radius R (ft)
852        I_spind = 9.942e-3; %slug*ft^2 at spindle
           I_motor = 0.048; %slug*fT^2 at motor
854        I_gr = 6.695e-3; %slug*ft^2 at motor
           J = (I_spind + rho*width_w/2*pi*(-(1.5/12)^4+(R)^4))*GR(1)^2+...
856            I_motor+I_gr;
        end
858
    %---------- Plotting ----------------------------------------------------
860 velocities_out = z(:,1:rolls)*60; %fpm
    tensions_out = z(:,rolls+1:rolls+spans);
862 if LC_unwind_control
        dancer_out = z(:,rolls+[8,9,18,19])*[.5 .5 0 0; 0 0 .5 .5]';
864     %dancer_out = z(:,rolls+[8,9,18+5,19+5])*[.5 .5 0 0; 0 0 .5 .5]';
        motor_inputs_out = z(:,sum(cnts(1:4))+[1:cnts(5)]);
866     % Rollers and Spans of interest
```

```
        velindex = [1 10 11 17 25];
868     tenindex = [1 8 9 18 19];
      % tenindex = [1 8 9 18+5 19+5];
870 else
        dancer_out = z(:,rolls+spans+[1])*180/pi;
872     motor_inputs_out = z(:,sum(cnts(1:5))+[1:cnts(6)]);
          % Rollers and Spans of interest
874     velindex = [1 10 11 17 25];
        tenindex = [1 3 4 18 19];
876 end
    waitbar( .6, h)
878 figure(2);
    subplot(2,2,1)
880 plot(time,velocities_out(:,velindex))
    title('velocity');
882 legend(['v' num2str(velindex(1))],['v' num2str(velindex(2))],...
        ['v' num2str(velindex(3))],['v' num2str(velindex(4))],['v' ...
884     num2str(velindex(5))],'location','best');
    grid;
886
    subplot(2,2,2)
888 plot(time,tensions_out(:,tenindex))
    title('tension');
890 legend(['t' num2str(tenindex(1))],['t' num2str(tenindex(2))],...
        ['t' num2str(tenindex(3))],['t' num2str(tenindex(4))],['t' ...
892     num2str(tenindex(5))],'location','best');
    grid;
894
    subplot(2,2,3)
896 if LC_unwind_control

898 title('Load Cells');
    legend('Unwind LC','Rewind LC','location','best')%,'xdot');
900 grid;
    else
902 plot(time,dancer_out);%,time,dancer_out(:,1))
    title('dancer');
904 legend('x (deg)','location','best')%,'xdot');
    grid;
906 end

908 subplot(2,2,4)
    plot(time,motor_inputs_out);%,time,dancer_out(:,1))
910 title('Motor Inputs');
    legend('Unwind','SWL','SWF','PR','Rewind','location','best')%,'xdot');
912 ylabel('Amps')
    grid;
914 waitbar( 1, h)
    if nargout>4
```

```
916     Events = RecordEvents(0,0,0,2);
    end
918 close(h)
    end
```

### J.3.2 Gain Calculations Separate

Listing J.2: The GainSched06 Code

```
1  function [ SS, SSder, time, z, Events ] = GainSched06( tfinal )
   %Euclid line simplified Slip model with 5 rollers unwind, idler, driven,
3  % idler, rewind.  Added eccentricity to driven #3 roll.  Making the roller
   % velocity derivative and span tension derivative all callable functions.
5  % Then, will change which one is called based on the condition of the event
   % for  slip.  This model initially (and all previous versions: 01, 02, 03)
7  % used the diameter instead of the radius.  Version 05 amends that.
   %   Detailed explanation goes here
9  %
   %   Version 16 - Using 15 and updating the frictions and bearing drags for
11 %   the new arrangement of rollers.  This is from the SlipModel16()
   %   function.
13 %
   %   Version 01 - Took SlipModel16() and removed worries about slip.  Added
15 %   gain calculation methods from Gain Calc paper and from Euclid line in
   %   order to use Pramode's gains.
17 %
   %   Version 01a - Copied 01, Includes time varying radii. Set up for load
19 %   cell control at rollers 9 and 19.
   %
21 %   Version 01b - Copied 01a, Added simulation of a shutdown. Fixing up the
   %   dancer control side of the script.
23 %
   %   Version 02 - Copied 01b,  Going to apply pure tension control to unwind
25 %   and rewind motors following RS paper.
   %
27 %   Version 03 - Copied 02,  Going to apply 2rev diameter check
   %
29 %   Version 04 - Copied 03, Converted to linearized dynamics, load cells
   %   moved to initial idle roller.
31 %
   %   Version 05 - Copied 04, Still using linearized dynamics, apply my
33 %   method to find tension gains.
   %
35 %   Version 06 - Copied 04, going to add speed control loop to the mix,
   %   gains calc'ed as a tension loop by itself and a speed loop by itself
37 %
   %
39
   if nargin <1
41     tfinal = 30;
```

```
end

file='C:\Users\quietman\Documents\MATLAB\WTS\EUCLID_FULL_LINE2.Dat';

dancers = 1; %number of dancers in system
%                          filename, number of rolls, number of spans
[r,s,c]=ElementPropReadIn(file,25,24);
Rolls = r{1};
Rolls = Rolls(:,:);
Spans = s{1};
Spans = Spans(:,:);
funnames = {@NonSlipVelDer,@Span2,@Span2,@Span2,@Span2};

RecordEvents(0,0,0,1);

%---------- Run Parameters ----------
rolls = 25;
spans = 24;
deltaV = ones(1,rolls)*1/60; %increase in speed in ft/s
Initial_Vel = ones(1,rolls)*100/60; %initial speeds of rollers in ft/s
%Initial_Vel(17) = 399.7/60;
deltaV = deltaV*4/4;
Initial_Vel = Initial_Vel*4/4;
t0 = 10; %lbf wound in tension
tf = 10.; %outgoing tension in lbf
lineten = 10; % line tension setpoint
lineten2 = lineten; %line tension for process section (unmeasured)
lineten3 = 10; %line tension for rewind section (Tension ON case)
dancer = [];
acc = [];
unwind = [];
eccent = [];
Ind = zeros(rolls,1);
GR = 1./[3.795 13.95 13.95 7.006 3.795];  %gear ratios for drives

LC_unwind_control = 1; %set to 1 for unwind to be controlled by the load
%   cell at roller 9, set to 0 for dancer control
if LC_unwind_control
    cnts = [rolls spans 2 2 5]; %[25, 24, 2variable radii, 2eccent roller,
    %5motor currents]
else
    %dancer controlled
    %[25 24, 2dancer states, 2variable radii, 2eccent roller, 5motor
    % currents]
    cnts = [rolls spans 2 2 2 5];
    dancer = [4];
end
slipind = 3;
nip10 = 1; % 1 use nip, 0 don't use nip
```

```
91
   g = 32.174; %ft/s^2 gravity
93 slipflag = 0; % signal to indicate that there is slip...
   y0 = zeros(40,1); %y0 = [y0; -0.005];
95 simrun = 0;
   %---------- Roller params --------------------
97
   L = Spans(4,:)/12; %feet
99 width_w = Spans(3,1)/12;%feet
   rho_w = Spans(5,:)*12^3/g;
101 %       lbm/in^3*(12in)^3/ft^3 / (lbm*ft/(s^2*lbf)) = lbf*s^2/ft^4
   A = Spans(2,1)/1000/12 * width_w; %ft^2
103 E = Spans(1,1)*1000*12^2;%lbf/ft^2

105 %Rollers
   Rn = Rolls(3,:)/12/2;% radius in ft for all rollers
107 Jn = Rolls(13,:); %lbf*ft*s^2
   Cmn = Rolls(19,:); % Torque/speed constant of a motor
109 Kmn = Rolls(18,:); % Torque constant of a motor, lbf*ft/amp
   Bfn = Rolls(20,:)*(1-0); % lbf*ft*sec, Bearing friction   ------------------<
111 theta_arm = Rolls(11,:)*pi/180;% dancer arm angle from the horizontal in rad.
   Mn = Rolls(10,:)/g; %slugs
113 Mun = [ones(1,17)]*.2403; % static friction max from measurements
   Mun([9 13]) = [0.1185 0.15];
115 e_3 = 0.0/12; %ft eccentricity
   Jpn = 0.209; %inertia of the dancer arm
117 %Jn = Jn*diag([1 .4 .4 .4 1 .4 1]);

119 Kmn([1 rolls]) = [Kmn(1)*GR(1) Kmn(rolls)*GR(5)];
   Cmn(1) = .001/GR(1);
121 Cmn(rolls) = 0.001/GR(5);
   Cdn = [zeros(1,3) 0.15 0 0 0 0 0 0 0 0];
123 %Rn(1) = 15/24;

125 %omega1 = Initial_Vel(1)/Rn(1); %radians/sec for eccentricity
   Kn = [zeros(1,3) 0 0 0 0 0 0 0 0 0];
127
   f_da = 33.86; %36.2lbf external force on dancer.
129 %   34 works out to 5.36lbf in the unwind section
   Rn_n = [zeros(1,5) 0 0 0 0 3 0 0]/24; %Nip radius in feet
131 Jn_n = [zeros(1,5) 0 0 0 0 4.635e-3 0 0 ]; %nip inertias
   Bfn_n = [zeros(1,5) 0 0 0 0 1 0 0]*0.0006; %nip bearing friction
133
   Bfn = [0.00060729 0.00016691 0.00126336 0.00017803 0.00004665 ...
135     0.00002655 0.00050257 0.00062522 0.00061941 0.00049733  ...
       0.00061733*ones(1,15)] ;
137
   bearing_sliding_friction = 0;  %boolea0n to toggle between viscous
139 %                               and sliding friction models
```

```
Bsldfn = [.1 0.05227297 0.41489625 0.17357908 0.23150706 0.13421975 ...
    0.16057590  0.20012461      0.20024913 .1 .1 0.15779922*ones(1,14)];
      %mu's from SpinDownTest.m
Bsldfn = Bsldfn*diag([38*(1.5/24) 12.89*.915/12*[1 1] ...
    1.188*.915/12 .876*.915/12*[1 1] 12.89*.915/12*[1 1 1] ...
    78.454*.915/12*[1 1] 12.89*.915/12*[1 1 1 1 1] 42*1.5/24 ...
    12.89*.915/12*[1 1 1 1 1 1 1] 38*(1.5/24)])*.05;
theta_arm = zeros(1,25);
theta_arm(4) = pi/2;
Larm = [zeros(1,3),15.925,zeros(1,13)]/12;

ParasiticTorque = [0 0*Rn(2) 0 0.00*Rn(4) 0 0 0 0 Rn(9)*0 zeros(1,16)];


%------------ Span params ------------------------
%E = 68000*144; %Young's modulus in lbf/ft^2
E=E; %-------------------------------------------------------------HERE!
A = 6.07/12*0.00536/12; % cross-sectional area in ft^2
rho = 1.0740000E-0002/g*144*12 ; %density of web in slug/ft^3
%L = [7.2132318E+1 2.9821722E+1 2.9793893E+1 6.8992896E+1 3.7417148E+1...
%      5.7479186E+1]/12; %span length in ft
rot_dir = [-1 1 -1 1 -1 -1 -1 1 -1 -1 1 1 -1 1 -1 1 -1 1 -1 1 -1 1 -1 ];
%        forward web direction CW = -1, CCW = 1
[sdta, theta_in,theta_on] = PrintLine(file,25,rot_dir,5,theta_arm);
theta_in=theta_in(1:rolls);
theta_on = theta_on(1:rolls);
theta_wrap = WrapAngle(sdta,Rn);


if LC_unwind_control
    %Gain calc paper LC control gains zeta=0.9, t_r=0.3 sec
%unwind tension loop gains

 Rn(1) = 18.5/24; %set to bare roller on rewind, unwind:full roll
% Rn(1) = 7.483/12; %set to 6in on rewind, unwind:7.483in
% Rn(1) = 3.315/12; %set to 9in on rewind, unwind:3.315in
%temp = -GainSched(Rn(1),1/60,.9,20.5,1);%--------Gain Scheduled Speed
temp = -GainSched(Rn(1),100/60,.9,20.5,1)/60;%--------Gain Scheduled tension
% negative sign due to loadcell being down stream of controlled motor
Kp(1) = temp(1)/1;
Ki(1) = temp(2)/1;

Kd(1)=0;%
%unwind speed loop gains
temp = SpdGainSched(Rn(1),.9,20.5,1);
Kp(2)= temp(1); %2.7928; %18.95/60;%1.8;%
Ki(2)= temp(2); %16.15768; %51.14/60;%5.19;%
Kd(2)=0.0;%0.005/100;%
%Driven speed loop gains
```

```
Kp(3) = 5.5421; %345.232/60; %
Ki(3) = 32.0624; %265.354/60; %
Kd(3) = 0;
%Swrap Follow speed loop gains
Kp(4) = 5.5421;%1.8; %
Ki(4) = 32.0624;%5.19; %
Kd(4) = 0;%0;
%Pull Roll speed loop gains
Kp(5) = 5.9853;%1.8; %
Ki(5) = 34.6265;%5.19; %
Kd(5) = 0;%0;
%rewind tension loop gains

 Rn(rolls) = 3/24; %set to bare roller on rewind
% Rn(rolls) = 6/12; %set to bare roller on rewind
% Rn(rolls) = 9/12; %set to bare roller on rewind
%temp = GainSched(Rn(rolls),1/60,.9,20.5,rolls)*1;%----Gain Scheduled speed
temp = GainSched(Rn(rolls),100/60,.9,20.5,rolls)/60;%--Gain Scheduled tension
Kp(6) = temp(1)/1;
Ki(6) = temp(2)/1;
% Kp(6) = .1718/2;%1.8; %
% Ki(6) = .0077618*10; %0.007761;%5.19; %
Kd(6) = 0.0;%0;
%rewind speed loop gains
temp = SpdGainSched(Rn(rolls),.9,20.5,1);
Kp(7) = temp(1);  %3.917;%1.8; %
Ki(7) = temp(2);  %22.6207;%5.19; %
Kd(7) = 0.0;%0;
else
    %Gain Calc paper Dancer Control zeta=0.9, t_r=0.3sec
%unwind tension loop gains
Kp(1)=0.4735/10;  %4/12;%
Ki(1)= 0.2742; % 9.4/12;%
Kd(1)=0;%
%unwind speed loop gains
Kp(2)=2.7928; %18.95/60;%1.8;%
Ki(2)=16.1576; %51.14/60;%5.19;%
Kd(2)=0.0;%0.005/100;%
%Driven speed loop gains
Kp(3) = 5.5421; %345.232/60; %
Ki(3) = 32.0624; %265.354/60; %
Kd(3) = 0;
%Swrap Follow speed loop gains
Kp(4) = 5.5421;%1.8; %
Ki(4) = 32.0624;%5.19; %
Kd(4) = 0;%0;
%Pull Roll speed loop gains
Kp(5) = 5.9853;%1.8; %
Ki(5) = 34.6265;%5.19; %
```

```
Kd(5) = 0;%0;
%rewind tension loop gains
Kp(6) = .1718/2;%1.8; %
Ki(6) = 0.0001552*600;%5.19; %
Kd(6) = 0.0;%0;
%rewind speed loop gains
Kp(7) =  3.91/30;%1.8; %
Ki(7) = 22.6207/40;%5.19; %
Kd(7) = 0.0;%0;
end

% Rn(1) = 7.483/12; %set to 6in on rewind, unwind:7.483in
 Rn(1) = 3.315/12; %set to 9in on rewind, unwind:3.315in
% Rn(rolls) = 6/12; %set to bare roller on rewind
 Rn(rolls) = 9/12; %set to bare roller on rewind
angle_travel_1=0; %stores last unwind position angle
angle_travel_2 =0; %stores last rewind position angle
z0=zeros(58,1);

%------------ Initial Conditions

if LC_unwind_control
    %LC control at roller 9
z = fsolve(@steadystateLC,(...
    rand(58,1)-.5)+[ones(rolls,1)*Initial_Vel(1); ones(spans,1)*lineten;...
    zeros(9,1)], optimoptions('fsolve','Algorithm',...
    'Levenberg-Marquardt','TolFun',1e-16,'MaxIter',4000,'TolX',1e-6));
z(10:11,1) = Initial_Vel(10:11)';
z(1,1) = Initial_Vel(1)';
z([17,rolls],1) = Initial_Vel([17,rolls])';
z(rolls+1,1) = 2*lineten-z(rolls+1,1);
z(rolls+10,1) = [.5 .5]*z(rolls+[9 11],1);
z(rolls+14,1) = 2*lineten2-z(rolls+15,1);
z(rolls+5+18,1) = 2*lineten3-z(rolls+5+19,1);
%z(20) = (f_da*5.032/12-z(21)*cos(theta_on(4))*(15.925-1.5)/12)...
%            /((15.92+1.5)/12)/cos(pi-theta_in(4));
z(50:51) = [Rn(1) Rn(rolls)]'; % set initial roll diameters
z(52) = z(1)/Rn(1);
z(53) = z(rolls)/Rn(rolls);

else
    %Dancer control at roller 4
    z = fsolve(@steadystateDan,rand(60,1), optimoptions('fsolve',...
    'Algorithm','Levenberg-Marquardt','TolFun',1e-16,'MaxIter',...
    4000,'TolX',1e-6));
z(1,1) = Initial_Vel(1);
z(10:11,1) = Initial_Vel(10:11)';
z([17,rolls],1) = Initial_Vel([17 rolls])';
z(rolls+10,1) = [.5 .5]*z(rolls+[9 11],1);
```

```
z(rolls+14,1) = 2*lineten2-z(rolls+15,1);
z(rolls+18,1) = 2*lineten3-z(rolls+19,1);
z(rolls+3,1) = (f_da*5.032/12-z(rolls+4)*cos(theta_on(4))*(15.925-1.5)/12)...
    /((15.92+1.5)/12)/cos(pi-theta_in(4));
z(rolls+spans+[2],1) = [0]';
z(52:53) = [Rn(1) Rn(rolls)]'; % set initial roll diameters
z(54,1) = z(1)/Rn(1);
z(55,1) = z(rolls)/Rn(rolls);
end

velocities_ss = z(1:rolls,1);
%velocities_ss(driven) = Initvel(driven);
tensions_ss = z(rolls+[1:spans]);% 3.4*ones(14,1); z(rolls+[20:22])];
if LC_unwind_control
    %nothing
    dancer_ss = [];
    motors_ss = z(54:58);
else
    dancer_ss = [z(50) 0]';
    motors_ss = z(56:60);
end
%motor_inputs_ss(1) = motor_inputs_ss(1);
%state_derivative_ss = f_z(z);


% Solve stuff
if LC_unwind_control
    z0 = [velocities_ss; ...
        tensions_ss; ...
        z(50:51);...
        z(52:53);...
        motors_ss]; %7x1
else
    z0 = [velocities_ss; ...
    tensions_ss; ...
    dancer_ss;...
    z(52:53);...
    z(54:55);...
    motors_ss]; %7x1
end

figure(1);
subplot(2,2,1),plot(velocities_ss*60,'>')
title('Velocities')
xlabel('Roller #')
ylabel('fpm')
subplot(2,2,2),plot(tensions_ss,'o')
title('Tensions')
xlabel('Span #')
```

```
ylabel('lbf')
%ylim([9.2  10.2])
if ~LC_unwind_control
subplot(2,2,3),bar(dancer_ss*12)
title('Dancer Position (in) and Velocity (in/s)')
xlabel('')
ylabel('magnitude')
end
subplot(2,2,4),bar(motors_ss)
title('Motor Inputs (amps)')
xlabel('Motor #')
ylabel('Magnitude')

% Set the output Steady-state
SS = z0;
% Set the output Steady-state derivative
h =waitbar(0, 'Please Wait');
hstep1 = 0.05*tfinal;
waitbar(0,h);
tstart = 0;
SSder = State_Controller(tstart,z0);

%RecordEvents(0,SS,1);



tstart=0:0.005:tfinal;
refine = 4;
options = odeset('Events',@events1,'Refine',refine,'RelTol',1e-5);%for
%ode45
%options = {'EventFunction',@PostProcessedSlip;'TimeStep',...
    0.005;'SlipTrue',0}; %for odeBR4
tout = 0;
zout = z0.';
teout = [];
yeout = [];
ieout = [];
istep = 0; % limit the number of bounces off the full extension or full
simrun=1;
slipflag = 0;
flag1 = 0;
flag2 = 0;
shutdown_flag = 0; %for resetting the IndScurve function once

timetest = 0.005; % ---- updated each timestep to compare and current
%           time and wait until the next one to switch equations
%           close situation to 5500 (arbitrary).
while tout(end) < tfinal && istep < 5500
```

```
385      [time,z] = ode45(@State_Controller,...
             tstart([1 end]),z0);%,options); ,te,ye,ie
387  ie=[];
         nt = length(time);
389    tout = [tout; time(2:nt)];    % dummy variable to store data
       zout = [zout; z(2:nt,:)];     % cummy variable to store data
391  %  teout = [teout; te];          % Events at tstart are never reported.
     %  yeout = [yeout; ye];  %values of event function (1 of them will be 0 )
393    if size(ie,1)>1
         ieout = [ieout; ie+10*slipflag];  %
395    else
           ieout = [ieout; ie'+10*slipflag];
397    end

399    if time(nt) < tfinal %check to see if integration needs to continue
           %find which event caused the stop of integration
401        if isempty(ie)
               options = odeset(options,'MaxStep',0.005);
403            tstart1 = [time(nt) (.005-rem(time(nt),.005))+time(nt)];
               tstart = [tstart1(1) tstart1(2):0.005:tfinal];
405

               [time,z] = ode45(@State_Controller,...
407          tstart,z(end,:),options);
              nt = length(time);
409       tout = [tout; time(2:nt)];    % dummy variable to store data
        zout = [zout; z(2:nt,:)];     % dummy variable to store data
411     teout = [teout; te];          % Events at tstart are never reported.
        yeout = [yeout; ye];  %values of event function (1 of them will be 0 )
413     ieout = [ieout; ie];  %
          else
415            z0 = z(nt,:);
               for i=1:numel(ie)
417                eventcase = ie(i);
           if slipflag == 0
419                switch eventcase;
             case 1
421              %T3 < T2*exp(-Mun*theta_wrap(3))
                 Ind(slipind) = -1;
423  %               funnames{1} = @SlipIdlerVelDer;
     %               funnames{2} = @SlipTensionIncoming;
425  %               funnames{3} = @SlipTensionOutgoing;
                 options = odeset(options,'MaxStep',.005); %@events2);
427                nm1=slipind-1;
                   n = slipind;
429              z0(rolls+nm1) = (z0(rolls+nm1)*L(nm1)+z0(rolls+n)*L(n))/...
                     (L(nm1)+L(n)*exp(Ind(n)*Mun(n)*theta_wrap(n)));
431              z0(rolls+n) = z0(rolls+nm1)*exp(Ind(n)*Mun(n)*theta_wrap(n));
                 %slipflag = 1;
433          case 2
```

```
                %T3 < T2*exp(Mun*theta_wrap(3))
435             Ind(slipind) = 1;
%                 funnames{1} = @SlipIdlerVelDer;
437 %               funnames{2} = @SlipTensionIncoming;
%                 funnames{3} = @SlipTensionOutgoing;
439             options = odeset(options,'MaxStep',.005);%@events2);
                  nm1=slipind-1;
441               n = slipind;
                z0(rolls+nm1) = (z0(rolls+nm1)*L(nm1)+z0(rolls+n)*L(n))...
443                /(L(nm1)+L(n)*exp(Ind(n)*Mun(n)*theta_wrap(n)));
                z0(rolls+n) = z0(rolls+nm1)*exp(Ind(n)*Mun(n)*theta_wrap(n));
445             %slipflag = 1;
                end
447         else
                switch eventcase;
449          case 1
                % Returning from slip condition 1
451             funnames{1} = @NonSlipVelDer;
                funnames{2} = @Span2;
453             funnames{3} = @Span2;
                Ind(2) = 0;
455             options = odeset('Events',@events1);
                %slipflag = 0;
457          case 2
                % Returning from slip condition 2
459             funnames{1} = @NonSlipVelDer;
                funnames{2} = @Span2;
461             funnames{3} = @Span2;
                Ind(2) = 0;
463             options = odeset('Events',@events1);
                %slipflag = 0;
465             end
            end
467             switch eventcase
             case 3
469             istep = 5500;
             case 4
471             istep = 5500;
                end
473          end
        end
475     z0=z0';
        tstart1 = [time(nt) (.005-rem(time(nt),.005))+time(nt)];
477     tstart = [tstart1(1) tstart1(2):0.005:tfinal];
        istep = istep + 1;
479   end
  end
481
  time = tout; % Return expected value names
```

```
z = zout;  % Return expected value names
%[time,z,EEvents] = PostProcessedSlip(time,z);
%-------------------------------------------------------------------------
%           S T A T E   D E R I V A T I V E   F U N C T I O N
%-------------------------------------------------------------------------
    function xdot = state_derive(T,x,u)

        v = x(1:rolls,1);
        t = x(rolls+[1:spans],1);
%         theta1 = mod(x(sum(cnts(1:3))+1,1),2*pi());
%         [r_1, r_1dot,r_1ddot,r_1ddot2] = Rollshape(theta1,...
%             x(sum(cnts(1:3))+2),Rn(unwind),rshape);
        xdot = x*0;

        % ---- roller velocity derivs General
        if bearing_sliding_friction
            xdot(1:rolls,1) = 1./Jn'.*(-Bsldfn'.*Rn' + Rn'.^2.*...
                ([t; tf]-[t0; t]) - Rn'.*ParasiticTorque');
        else
            xdot(1:rolls,1) = 1./Jn'.*(-Bfn'.*v+ Rn'.^2.*([t; tf]-[t0; t])...
                - Rn'.*ParasiticTorque');
        end
        % specific derivatives for certain rollers
        %5-24-19 uses viscous friction model
        xdot(9,1) = feval(funnames{1},t(8:9),v(9),9);

        if LC_unwind_control
            Jn(1) = inertia(x(50));
        xdot(1,1) = (1/Jn(1)*(-(Bfn(1)+Cmn(1))*v(1)/(2*pi*GR(1)...
            *x(50))*GR(1)+ x(50)*(t(1)*GR(1))+Kmn(1)*(u(1))+...
            Mn(1)*e_3*g*sin(x(rolls+spans+3,1))))...
            *(2*pi*GR(1)*x(50));
        else
            %dancer control added 2 states
        xdot(1,1) = (1/0.056*(-Bfn(1)*v(1)/(2*pi*GR(1)*x(52))*GR(1)+...
            x(52)*(t(1)*GR(1)) +Kmn(1)*(u(1))+Mn(1)*e_3*g...
            *sin(x(sum(cnts(1:3))+1,1))))*(2*pi*GR(1)*x(52));
        end
        if nip10
            xdot(10,1) = 1/(Rn(10)^2*Jn_n(10)+Rn_n(10)^2*Jn(10))*( ...
                -(Rn_n(10)^2*Bfn(10)+Rn(10)^2*Bfn_n(10))*v(10)+Rn_n(10)^2*...
                Rn(10)^2*(t(10) - t(9))+Rn_n(10)^2*Rn(10)*Kmn(10)*u(2));
        else
            xdot(10,1) = xdot(10,1) + 1/Jn(10)*(Kmn(10)*(u(2)*Rn(10)));
        end
            xdot(11,1) = xdot(11,1) + 1/Jn(11)*(Kmn(11)*(u(3)*Rn(11)));
            xdot(17,1) = 1/(Rn(17)^2*Jn_n(10)+Rn_n(10)^2*Jn(17))*( ...
                -(Rn_n(10)^2*Bfn(17)+Rn(17)^2*Bfn_n(10))*v(17)+Rn_n(10)^2*...
                Rn(17)^2*(t(17) - t(16))+Rn_n(10)^2*Rn(17)*Kmn(17)*u(4));
```

```
        if LC_unwind_control
533         Jn(rolls) = inertia(x(51));
            xdot(rolls,1) = (1/Jn(rolls)*(-(Bfn(rolls)+Cmn(rolls))...
535             *v(rolls)/(2*pi*GR(5)*x(51))-...
                x(51)*(t(spans)*GR(5))+Kmn(rolls)*(u(5))))*GR(5)*2*pi*x(51);
537     else
            xdot(rolls,1) = (1/.055*(-Bfn(rolls)*v(rolls)/(2*pi*x(53))-...
539             x(53)*(t(spans)*GR(5))+Kmn(rolls)*(u(5))))*GR(5)*2*pi*x(53);
    %       xdot(rolls,1) = 1/Jn(rolls)*(-Bfn(rolls)*v(rolls)-...
541 %           x(53)^2.*(t(spans))+Kmn(rolls)*(u(5)*x(53)));
        end
543     if ~LC_unwind_control
        %-- Dancer derivs
545     xdot(rolls+spans+1,1) = x(rolls+spans+2,1);
        xdot(rolls+spans+2,1) = 1/(Jpn)*(f_da*5.032/12 - ...
547         t(3)*cos(pi-theta_in(dancer))*(15.925+1.5)/12 - ...
            t(4)*cos(theta_on(dancer))*((15.925-1.5)/12) - ...
549         Kn(4)*x(rolls+spans+1) - Cdn(4)*x(rolls+spans+2) -...
            13.95/12*.317*g*sin(x(rolls+spans+1)));
551     end
        %-- tension derivs
553     tens = [t0; t]; %include wound in tension
        for i = 1:spans
555         if ~LC_unwind_control && i == dancer
                [temp(3), temp(4)] = PendDancerLeng(x(sum(cnts(1:2))+1),...
557                 -x(sum(cnts(1:2))+2),L(i),Larm(i),...
                    1.38/12,1.5/12,1.5/12);
559             sp0 = [v(i+[0 1])', tens(i+[0 1],1)', L(i)+...
                    temp(3), ...
561                     temp(4), theta_on(i)]';
            elseif ~LC_unwind_control && i+1==dancer
563             [temp(1), temp(2)] = PendDancerLeng(x(sum(cnts(1:2))+1),-...
                    x(sum(cnts(1:2))+2),L(i),Larm(i+1),...
565                 3.084/12,1.5/12,1.5/12);
                sp0 = [v(i+[0 1])', tens(i+[0 1],1)', L(i)+...
567                 temp(1),...
                    temp(2), theta_in(i+1)]';
569
            elseif i == unwind
571             % d1 calc'ed on line 24
                l1 = sqrt(d1 - (r_1+Rn(2))^2);
573             dl1 = -(r_1+Rn(2))*r_1dot*theta1dot/sqrt(d1-(r_1+Rn(2))^2);
                sp0 = [v(i+[0 1])',tens(i+[0 1],1)',l1,dl1]';
575         else
                sp0 = [v(i+[0 1])' tens(i+[0 1],1)' L(i) 0 theta_on(i)]';
577         end
            % for span2Lin sp0 = [sp0(from above) vel1_ss vel2_ss]
579         sp0 = [sp0; z0(i+[0 1])];
            xdot(rolls+i,1) = span2Lin(T, sp0, E, A);
```

```
581         end
583         if LC_unwind_control
                xdot(rolls+spans+[1 2],1) = [-0.00536/12/(2*pi)...
585                 *(x(1,1)/x(50));...
                    0.00536/12/(2*pi)*(x(rolls,1)/x(51,1))];
587
                %angular speeds
589             xdot(rolls+spans+3,1) = x(1,1)/x(50);
                xdot(rolls+spans+4,1) = x(rolls,1)/x(51);
591         else
                %dancer controlled
593             xdot(rolls+spans+2+[1 2],1) = [-0.00536/12/(2*pi)...
                    *(x(1,1)/x(52));...
595                 0.00536/12/(2*pi)*(x(rolls,1)/x(53))];
597             %angular speeds
                xdot(rolls+spans+5,1) = x(1,1)/Rn(1);
599             xdot(rolls+spans+6,1) = x(rolls,1)/Rn(rolls);
            end
601         % xdot = xdot';
603     end
%-------------------------------------------------------------------------
605 %           S T A T E   C O N T R O L L E R   F U N C T I O N
%-------------------------------------------------------------------------
607     function [ydot] = State_Controller(t,y)
            %[t,y] = PostProcessedSlip(t,y);
609         ydot = y*0;
            % Setup inputs
611
            if t<0.5
613             %Step in Tension
            r1 = [100 0 0]'/60;
615         r2 = r1;
            else
617             %Step in Tension
                r1 = [100 0 0]'/60;
619             r2 = r1;
                lineten = 11;
621         end
            % get state derivative
623         if LC_unwind_control
                % update the steady-state velocities with r1 and r2
625             z0(1,1) = r1(1);
                z0(2:rolls,1)= r2(1)*ones(rolls-1,1);
627             ydot(1:sum(cnts(1:4)),1) = state_derive(t,y(1:sum(cnts(1:4)),...
                    1),y(sum(cnts(1:4))+[1:cnts(5)],1));
629 %             if mod(y(52),2*pi)>pi && flag1 == 0
```

```
%                 flag1 = 1;
%             end
%             if abs(mod(y(52),2*pi)+.5/180*pi)>2*pi && flag1
%                 if t>0.5
%                 temp = -GainSched(y(50),r1(1),.9,20.5,1);
%                 Kp(1) = temp(1);
%                 Ki(1) = temp(2);
%                 flag1 = 0;
%                 RecordEvents(t,y,[1 0]);
%                 end
%             end
%             if mod(y(53),2*pi)>pi && flag2 == 0
%                flag2 = 1;
%             end
%             if abs(mod(y(53),2*pi)+.5/180*pi)>2*pi && flag2
%                 if t>0.5
%                 temp = GainSched(y(51),r2(1),.9,20.5,rolls);
%                 Kp(6) = temp(1);
%                 Ki(6) = temp(2);
%                flag2 = 0;
%                RecordEvents(t,y,[1 0]);
%                 end
%          end
            %tension errors
            F(1)=[.5 .5]*y(rolls+[1 2],1);
            Fdot(1)=[.5 .5]*ydot(rolls+[1 2],1);
            F(2)=[.5 .5]*y(rolls+5+[18 19],1);
            Fdot(2)=[.5 .5]*ydot(rolls+5+[18 19],1);
            %speed errors
            errors = [ r1(1)-y(1), r2(1)-y(10), r2(1)-y(11), r2(1)-y(17), ...
                    r2(1)-y(rolls);...
                r1(2)-ydot(1), r2(2)-ydot(10), r2(2)-ydot(11), r2(2)-...
                    ydot(17), r2(2)-ydot(rolls)];

            %trim speed calc
            TrimV(1) = (Kp(1)*(-Fdot(1))+Ki(1)*(lineten-F(1)));
            TrimV(2) = (Kp(6)*(-Fdot(2))+Ki(6)*(lineten-F(2)));
            %current calc
            ydot(sum(cnts(1:4))+1,1) = (Kp(2)*(errors(2,1)+TrimV(1)*0)+...
                Ki(2)*(errors(1,1)+TrimV(1)));
            ydot(sum(cnts(1:4))+2,1) = Kp(3)*errors(2,2)+Ki(3)*errors(1,2);
            ydot(sum(cnts(1:4))+3,1) = Kp(4)*errors(2,3)+Ki(4)*errors(1,3);
            ydot(sum(cnts(1:4))+4,1) = Kp(5)*errors(2,4)+Ki(5)*errors(1,4);
            ydot(sum(cnts(1:4))+5,1) =  Kp(7)*(errors(2,5)+TrimV(2)*0)+...
                Ki(7)*(errors(1,5)+TrimV(2));

        else
            %dancer Controlled
            ydot(1:sum(cnts(1:5)),1) = state_derive(t,y(1:sum(cnts(1:5))),...
```

```
679                 1),y(sum(cnts(1:5))+[1:cnts(6)],1));

681
                F(1)= -y(rolls+spans+1,1);
683             Fdot(1)=-y(rolls+spans+2,1);
                F(2)=[.5 .5]*y(rolls+[18 19],1);
685             Fdot(2)=[.5 .5]*ydot(rolls+[18 19],1);

687             % find errors in speed
                errors = [ r1(1)-y(1), r2(1)-y(10), r2(1)-y(11), r2(1)-y(17) ...
689                     r2(1)-y(rolls);...
                    r1(2)-ydot(1), r2(2)-ydot(10), r2(2)-ydot(11), r2(2)-...
691                     ydot(17) r2(2)-ydot(rolls)];

693             TrimV(1) = (Kp(1)*(Fdot(1))+Ki(1)*(F(1)));
                TrimV(2) = (Kp(6)*(-Fdot(2))+Ki(6)*(lineten-F(2)));
695             %current calc
                ydot(sum(cnts(1:5))+1,1) = (Kp(2)*(errors(2,1)+TrimV(1)*0)+...
697                 Ki(2)*(errors(1,1)+TrimV(1)));
                ydot(sum(cnts(1:5))+2,1) = Kp(3)*errors(2,2)+Ki(3)*errors(1,2);
699             ydot(sum(cnts(1:5))+3,1) = Kp(4)*errors(2,3)+Ki(4)*errors(1,3);
                ydot(sum(cnts(1:5))+4,1) = Kp(5)*errors(2,4)+Ki(5)*errors(1,4);
701             ydot(sum(cnts(1:5))+5,1) = Kp(7)*(errors(2,5)+TrimV(2)*0)+...
                    Ki(7)*(errors(1,5)+TrimV(2));
703         end

705         y0=ydot;
            % update waitbar
707         if t> hstep1
                waitbar( hstep1/tfinal/2, h)
709             hstep1 = hstep1 +.1*tfinal;
            end
711     end
        %---------- NonSlip idler vel. derive function
713     function dv = NonSlipVelDer(tens,v,ind)
            dv = 1/Jn(ind)*(-Bfn(ind)*v+ Rn(ind)^2*([-1 1]*tens)...
715             - ParasiticTorque(ind)*Rn(ind));
        end
717     %---------- Slip idler vel. deriv function
        function dv = SlipIdlerVelDer(tens,v,ind)
719         Fn = [-cos(theta_in(ind)) cos(theta_on(ind))]*tens(1:2,1);
            if v < 0.001
721             Pars = 0;
            else
723             Pars = ParasiticTorque(ind);
            end
725         dv = 1/Jn(ind)*(-Bfn(ind)*v + Ind(ind)*Mun(ind)*Fn*Rn(ind)^2 ...
                - Pars*Rn(ind));
727     end
```

```
    %---------- NonSlip Tension deriv span 1
    %Just use Span.m like normal
    %---------- Slip Tension deriv span 1
    function dt = SlipTensionIncoming(Props)
        %Taken from WTS discussion of Turn bars
        %Props = [L1+L2*e^(mu*alpha), E*A, t0, T2, V1, V3]
        L1 = Props(1); EA = Props(2); Tnm1 = Props(3); Tnp1 = Props(4);
        Vn = Props(5); Vnp2 = Props(6);
        dt =1/L1*(Vnp2*(EA - Tnp1) - Vn*(EA - Tnm1));
    end
    %---------- Slip Tension deriv span 2
    function dt = SlipTensionOutgoing(Props)
        %Taken from WTS discussion of Turn bars
        %Props = [dT1, exp(mu2*alpha2)];
        dT1 = Props(1); emu = Props(2);
        dt = dT1*emu;
    end
    %---------- Events function
    function [value,isterminal,direction] = events1(t,y)
% Locate the time when tension ratio violates capstan equation
% and stop integration.
    %slipflag = 1;
    %Ind(2)  = 0;
    nm1 = slipind-1;
    n = slipind;
    testforTension = find(y(rolls+[1:spans])<0);

    value = [ ...%y(sum(cnts(1:2))+1) - 50/12, ...
        %y(sum(cnts(1:2))+1) + 5/12, ...
        y(rolls+3), ...%
        y(rolls+4) , ...
        y(rolls+1)+20, ...
        y(rolls+2)+20
        ]';
    isterminal = [1 1 1 1]';   % stop the integration
    direction = [-1 1 -1 -1]';   % [positive negative] direction
    end

    function z = steadystateLC(x)
        c = state_derive(0,[...
            Initial_Vel(1);...
            x(2:9,1);
            Initial_Vel(10:11)'; ...
            x(12:16);...
            Initial_Vel(17);...
            x(18:24);... %end of speeds
            Initial_Vel(25);...
            2*lineten-x(27,1);...
            x(27:28);...
```

```
%             (f_da*5.032/12-x(29)*cos(theta_on(4))*(15.925-1.5)/12)/...
%                 ((15.925+1.5)/12)/cos(pi-theta_in(4));... %span tensions
            x(29:33,1);...  %29=span 4
%            2*lineten-x(34,1);...  %33 =span 8
            x(34);...
            [.5 .5]*x([34,36],1);
            x(36:38);...
            2*lineten2-x(40,1);...
            x(40:47,1);... %42= span 17
            2*lineten3-x(49,1);...  %43 = span 18
            x(49,1);...
            [Rn(1) Rn(rolls)]';... %variable radii
            x(1)/Rn(1);... %omega(1)
            x(rolls)/Rn(rolls);...
            ],...%
            x(54:58)); %motor currents
        z = c;
        %z(5) = Initial_Vel(5);
        %z(9) = f_da-z(10);
        %z(12) = 2*lineten-z(13);
        %z(14:15) = [0 0];
        z(52) = x(52) - z(1)/Rn(1);
        z(53) = x(53) - z(rolls)/Rn(rolls);
        %z(17:19) = x(17:19);
    end
    function z = steadystateDan(x)
        c = state_derive(0,[...
            Initial_Vel(1);...
            x(2:9,1);
            Initial_Vel(10:11)'; ...
            x(12:16);...
            Initial_Vel(17);...
            x(18:24);...
            Initial_Vel(25);... %end of speeds
            x(26:27);...
            (f_da*5.032/12-x(29)*cos(theta_on(4))*(15.925-1.5)/12)/...
                ((15.925+1.5)/12)/cos(pi-theta_in(4));... %span tensions
            x(29:33,1);...  %29=span 4
%            2*lineten-x(34,1);...  %33 =span 8
            x(34);...
            [.5 .5]*x([34,36],1);
            x(36:38);...
            2*lineten2-x(40,1);...
            x(40:42,1);... %42= span 17
            2*lineten3-x(44,1);...  %43 = span 18
            x(44:49,1);...
            x(50,1);...
            0;...
            [Rn(1) Rn(rolls)]';... %variable radii
```

```
            x(1)/Rn(1);... %omega(1)
827         x(rolls)/Rn(rolls);...
            ],...%
829         x(56:60)); %motor currents
        z = c;
831     %z(5) = Initial_Vel(5);
        %z(9) = f_da-z(10);
833     %z(12) = 2*lineten-z(13);
        %z(14:15) = [0 0];
835     z(54) = x(54) - z(1)/Rn(1);
        z(55) = x(55) - z(rolls)/Rn(rolls);
837     %z(17:19) = x(17:19);
    end
839


841

        function newTen = Whitworth(T,Tnm1,L,Lnm1,INDmualpha)
843     %applies the Whitworth equation to the tensions
        %INDmualpha = IND(i,j)*Mun(p)*theta_wrap(p) where IND(i,j) is
845     %the +/-1 value from the test1 or test2 above, Mun is the
        %coefficient of friction for roller p, and theta_wrap(p) is the
847     %wrap angle in radians for roller p.
            newTen = (Tnm1*Lnm1+T*L)/(Lnm1+L*exp(INDmualpha));
849     end

851     function phi = PHIi(phi_nm1,ind_n,mu_n,alpha_n)
            %Multiplier in slip calculation
853         phi = phi_nm1*exp(ind_n*mu_n*alpha_n);
        end
855
    function [k] = GainSched(R,V0,z,om,ind)
857     %quickly calc my method of finding gains.

859     if ind == rolls
            ind=ind-1;
861         Li =sum(L(ind-[0:0]));
            R_b=R/1.5*12;
863         ind=ind+1;
        J = inertia(R);%1+(.055*R_b^4)*GR(5)^2;
865         num=-(Kmn(ind)*A*E*R/sum(Li)/J);
        else
867         Li = sum(L(1:1));
            R_b=R/1.5*12;
869     J = inertia(R);%1+(.055*R_b^4)*GR(1)^2;
            num=-(Kmn(ind)*A*E*R/sum(Li)/J);
871     end
        den = conv([1 V0/Li],[1 (Bfn(ind)+Cmn(ind))/J]);
873     a = [z*om 0 den(2)-2*z*om; 2*om^2*z^2 -num den(3)-om^2];
        arf = rref(a);
```

```
875         kr = arf(1,3);
            kp = arf(2,3);
877         ki = kr*z*om^3/num;  %
            k=[abs(kp),ki];
879     end
        function [k] = SpdGainSched(R,z,om,ind)
881         %quickly calc my method of finding gains.
            if ind == rolls
883             ind=ind-1;
                Li =sum(L(ind-[0:0]));
885             R_b=R/1.5*12;
                ind=ind+1;
887         J = inertia(R);%1+(.055*R_b^4)*GR(5)^2;
            else
889             Li = sum(L(1:1));
                R_b=R/1.5*12;
891         J = inertia(R);%1+(.055*R_b^4)*GR(1)^2;
            end
893         kp = (-(Bfn(ind)+Cmn(ind))+2*z*om*J)/(Rn(ind)*Kmn(ind));
            ki = (om^2*J)/(Rn(ind)*Kmn(ind));
895         k=[abs(kp),ki];
        end
897     function J = inertia(R)
           % calculate inertia based on Euclid parames and radius R (ft)
899        I_spind = 9.942e-3; %slug*ft^2 at spindle
           I_motor = 0.048; %slug*fT^2 at motor
901        I_gr = 6.695e-3; %slug*ft^2 at motor
           J = (I_spind + rho*width_w/2*pi*(-(1.5/12)^4+(R)^4))*GR(1)^2+...
903            I_motor+I_gr;
        end
```

### J.3.3 Gain Calculation with Variable Tension Gains and Variable Speed Gains

Listing J.3: The GainSched07 Code

```
function [ SS, SSder, time, z, Events ] = GainSched07( tfinal,case_ds )
2 %Euclid line simplified Slip model with 5 rollers unwind, idler, driven,
  % idler, rewind.  Added eccentricity to driven #3 roll.  Making the roller
4 % velocity derivative and span tension derivative all callable functions.
  % Then, will change which one is called based on the condition of the event
6 % for  slip.  This model initially (and all previous versions: 01, 02, 03)
  % used the diameter instead of the radius.  Version 05 amends that.
8 %   Detailed explanation goes here
  %
10 %   Version 16 - Using 15 and updating the frictions and bearing drags for
  %   the new arrangement of rollers.  This is from the SlipModel16()
12 %   function.
  %
14 %   Version 01 - Took SlipModel16() and removed worries about slip.  Added
  %   gain calculation methods from Gain Calc paper and from Euclid line in
```

```
%   order to use Pramode's gains.
%
%   Version 01a - Copied 01, Includes time varying radii. Set up for load
%   cell control at rollers 9 and 19.
%
%   Version 01b - Copied 01a, Added simulation of a shutdown. Fixing up the
%   dancer control side of the script.
%
%   Version 02 - Copied 01b,  Going to apply pure tension control to unwind
%   and rewind motors following RS paper.
%
%   Version 03 - Copied 02,  Going to apply 2rev diameter check
%
%   Version 04 - Copied 03, Converted to linearized dynamics, load cells
%   moved to initial idle roller.
%
%   Version 05 - Copied 04, Still using linearized dynamics, apply my
%   method to find tension gains.
%
%   Version 06 - Copied 04, going to add speed control loop to the mix,
%   gains calc'ed as a tension loop by itself and a speed loop by itself
%
%   Version 07 - Copied 06, going to calc tension gains including effect of
%   speed gains.  Used Gain Calc paper's method.  Added case_ds to select
%   from the function call what diameter pair to simulate.
%
%
%


if nargin <1
    case_ds = 1;
    tfinal = 30;
elseif nargin <2
    case_ds =1;
end

file='C:\Users\quietman\Documents\MATLAB\WTS\EUCLID_FULL_LINE2.Dat';

dancers = 1; %number of dancers in system
%                           filename, number of rolls, number of spans
[r,s,c]=ElementPropReadIn(file,25,24);
Rolls = r{1};
Rolls = Rolls(:,:);
Spans = s{1};
Spans = Spans(:,:);
funnames = {@NonSlipVelDer,@Span2,@Span2,@Span2,@Span2};

RecordEvents(0,0,0,1);

```

```
%---------- Run Parameters ----------
rolls = 25;
spans = 24;
deltaV = ones(1,rolls)*1/60; %increase in speed in ft/s
Initial_Vel = ones(1,rolls)*100/60; %initial speeds of rollers in ft/s
%Initial_Vel(17) = 399.7/60;
deltaV = deltaV*4/4;
Initial_Vel = Initial_Vel*4/4;
t0 = 10; %lbf wound in tension
tf = 10.; %outgoing tension in lbf
lineten = 10; % line tension setpoint
lineten2 = lineten; %line tension for process section (unmeasured)
lineten3 = 10; %line tension for rewind section (Tension ON case)
dancer = [];
acc = [];
unwind = [];
eccent = [];
Ind = zeros(rolls,1);
GR = 1./[3.795 13.95 13.95 7.006 3.795];  %gear ratios for drives

LC_unwind_control = 1; %set to 1 for unwind to be controlled by the load
%   cell at roller 9, set to 0 for dancer control
if LC_unwind_control
    cnts = [rolls spans 2 2 5]; %[25, 24, 2variable radii, 2eccent roller,
    %5motor currents]
else
    %dancer controlled
    %[25 24, 2dancer states, 2variable radii, 2eccent roller, 5motor
    % currents]
    cnts = [rolls spans 2 2 2 5];
    dancer = [4];
end
slipind = 3;
nip10 = 1; % 1 use nip, 0 don't use nip

g = 32.174; %ft/s^2 gravity
slipflag = 0; % signal to indicate that there is slip...
y0 = zeros(40,1); %y0 = [y0; -0.005];
simrun = 0;
%---------- Roller params ----------

L = Spans(4,:)/12; %feet
width_w = Spans(3,1)/12;%feet
rho_w = Spans(5,:)*12^3/g;%lbm/in^3*(12in)^3/ft^3 / (lbm*ft/(s^2*lbf))
%                               = lbf*s^2/ft^4
A = Spans(2,1)/1000/12 * width_w; %ft^2
E = Spans(1,1)*1000*12^2;%lbf/ft^2

%Rollers
```

```
Rn = Rolls(3,:)/12/2;% radius in ft for all rollers
Jn = Rolls(13,:); %lbf*ft*s^2
Cmn = Rolls(19,:); % Torque/speed constant of a motor
Kmn = Rolls(18,:); % Torque constant of a motor, lbf*ft/amp
Bfn = Rolls(20,:)*(1-0); % lbf*ft*sec, Bearing friction
theta_arm = Rolls(11,:)*pi/180; % dancer arm angle from the horizontal
%                                 in rad.
Mn = Rolls(10,:)/g; %slugs
Mun = [ones(1,17)]*.2403; % static friction max from measurements
Mun([9 13]) = [0.1185 0.15];
e_3 = 0.0/12; %ft eccentricity
Jpn = 0.209; %inertia of the dancer arm

Kmn([1 rolls]) = [Kmn(1)*GR(1) Kmn(rolls)*GR(5)];
Cmn(1) = .001/GR(1);
Cmn(rolls) = 0.001/GR(5);
Cdn = [zeros(1,3) 0.15 0 0 0 0 0 0 0 0];

Kn = [zeros(1,3) 0 0 0 0 0 0 0 0 0];

f_da = 33.86; %36.2lbf external force on dancer.
%   34 works out to 5.36lbf in the unwind section
Rn_n = [zeros(1,5) 0 0 0 0 3 0 0]/24; %Nip radius in feet
Jn_n = [zeros(1,5) 0 0 0 0 4.635e-3 0 0 ]; %nip inertias
Bfn_n = [zeros(1,5) 0 0 0 0 1 0 0]*0.0006; %nip bearing friction

Bfn = [0.00060729 0.00016691 0.00126336 0.00017803 0.00004665 ...
    0.00002655 0.00050257 0.00062522 0.00061941 0.00049733  ...
    0.00061733*ones(1,15)] ;

bearing_sliding_friction = 0;  %boolea0n to toggle between viscous
%                                and sliding friction models
Bsldfn = [.1 0.05227297 0.41489625 0.17357908 0.23150706 0.13421975 ...
    0.16057590 0.20012461 0.20024913 .1 .1 0.15779922*ones(1,14)];
    %mu's from SpinDownTest.m
Bsldfn = Bsldfn*diag([38*(1.5/24) 12.89*.915/12*[1 1] ...
    1.188*.915/12 .876*.915/12*[1 1] 12.89*.915/12*[1 1 1] ...
    78.454*.915/12*[1 1] 12.89*.915/12*[1 1 1 1 1] 42*1.5/24 ...
    12.89*.915/12*[1 1 1 1 1 1 1] 38*(1.5/24)])*.05;
theta_arm = zeros(1,25);
theta_arm(4) = pi/2;
Larm = [zeros(1,3),15.925,zeros(1,13)]/12;

ParasiticTorque = [0 0*Rn(2) 0 0.00*Rn(4) 0 0 0 0 Rn(9)*0 zeros(1,16)];
%------------ Span params ------------------------
%E = 68000*144; %Young's modulus in lbf/ft^2
A = 6.07/12*0.00536/12; % cross-sectional area in ft^2
rho = 1.0740000E-0002/g*144*12 ; %density of web in slug/ft^3
%L = [7.2132318E+1  2.9821722E+1  2.9793893E+1  6.8992896E+1 ...
```

```
%   3.7417148E+1  5.7479186E+1]/12; %span length in ft
rot_dir = [-1 1 -1 1 -1 -1 -1 1 -1 -1 1 1 -1 1 -1 1 -1 1 -1 1 -1 1...
    -1 1 -1 ]; %forward web direction CW = -1, CCW = 1
[sdta, theta_in,theta_on] = PrintLine(file,25,rot_dir,5,theta_arm);
theta_in=theta_in(1:rolls);
theta_on = theta_on(1:rolls);
theta_wrap = WrapAngle(sdta,Rn);

if LC_unwind_control
    %Gain calc paper LC control gains zeta=0.9, t_r=0.3 sec
%unwind tension loop gains

case_d = [18.5/24 7.483/12 3.315/12 18.5/24 18.5/24;...  %Rn(1)
          3/24    6/12     9/12     3/24    3/24; ...    %Rn(rolls)
          18.5/24 7.483/12 3.315/12 7.483/12 3.315/12;...  %Rn(1) fixed
          3/24    6/12     9/12     6/12     9/12; ...    %Rn(rolls) fixed
          ];
% Rn(1) = 18.5/24; %set to bare roller on rewind, unwind:full roll
% Rn(1) = 7.483/12; %set to 6in on rewind, unwind:7.483in
% Rn(1) = 3.315/12; %set to 9in on rewind, unwind:3.315in
%case_ds = 1;
Rn(1) = case_d(1,case_ds);
%temp = -GainSched(Rn(1),1/60,.9,20.5,1);%--------Gain Scheduled Speed
temp = GainSched(Rn(1),100/60,.9,20.5,1);%--------Gain Scheduled tension
% negative sign due to loadcell being down stream of controlled motor
Kp(1) = temp(1)*2;
Ki(1) = temp(2)/1;

Kd(1)=0;%
%unwind speed loop gains
%temp = SpdGainSched(Rn(1),.9,20.5,1);
Kp(2)= temp(3)/1; %2.7928; %18.95/60;%1.8;%
Ki(2)= temp(4)/1; %16.15768; %51.14/60;%5.19;%
Kd(2)=0.0;%0.005/100;%
%Driven speed loop gains
Kp(3) = 5.5421; %345.232/60; %
Ki(3) = 32.0624; %265.354/60; %
Kd(3) = 0;
%Swrap Follow speed loop gains
Kp(4) = 5.5421;%1.8; %
Ki(4) = 32.0624;%5.19; %
Kd(4) = 0;%0;
%Pull Roll speed loop gains
Kp(5) = 5.9853;%1.8; %
Ki(5) = 34.6265;%5.19; %
Kd(5) = 0;%0;
%rewind tension loop gains

% Rn(rolls) = 3/24; %set to bare roller on rewind
```

```
% Rn(rolls) = 6/12; %set to bare roller on rewind
% Rn(rolls) = 9/12; %set to bare roller on rewind
Rn(rolls) = case_d(2,case_ds);
%temp = GainSched(Rn(rolls),1/60,.9,20.5,rolls)*1;%--Gain Scheduled speed
temp = GainSched(Rn(rolls),100/60,.9,20.5,rolls);%-Gain Scheduled tension
Kp(6) = temp(1)*2;
Ki(6) = temp(2)/1; %/2% for LC at 24
% Kp(6) = .1718/2;%1.8; %
% Ki(6) = .0077618*10; %0.007761;%5.19; %
Kd(6) = 0.0;%0;
%rewind speed loop gains
%temp = SpdGainSched(Rn(rolls),.9,20.5,1);
Kp(7) = temp(3)/1;  %3.917;%1.8; %
Ki(7) = temp(4)/1;  %22.6207;%5.19; %
Kd(7) = 0.0;%0;
else
    %Gain Calc paper Dancer Control zeta=0.9, t_r=0.3sec
%unwind tension loop gains
Kp(1)=0.4735/10;  %4/12;%
Ki(1)= 0.2742; % 9.4/12;%
Kd(1)=0;%
%unwind speed loop gains
Kp(2)=2.7928; %18.95/60;%1.8;%
Ki(2)=16.1576; %51.14/60;%5.19;%
Kd(2)=0.0;%0.005/100;%
%Driven speed loop gains
Kp(3) = 5.5421; %345.232/60; %
Ki(3) = 32.0624; %265.354/60; %
Kd(3) = 0;
%Swrap Follow speed loop gains
Kp(4) = 5.5421;%1.8; %
Ki(4) = 32.0624;%5.19; %
Kd(4) = 0;%0;
%Pull Roll speed loop gains
Kp(5) = 5.9853;%1.8; %
Ki(5) = 34.6265;%5.19; %
Kd(5) = 0;%0;
%rewind tension loop gains
Kp(6) = .1718/2;%1.8; %
Ki(6) = 0.0001552*600;%5.19; %
Kd(6) = 0.0;%0;
%rewind speed loop gains
Kp(7) =  3.91/30;%1.8; %
Ki(7) = 22.6207/40;%5.19; %
Kd(7) = 0.0;%0;
end

% Rn(1) = 7.483/12; %set to 6in on rewind, unwind:7.483in
% Rn(1) = 3.315/12; %set to 9in on rewind, unwind:3.315in
```

```
% Rn(rolls) = 6/12; %set to bare roller on rewind
% Rn(rolls) = 9/12; %set to bare roller on rewind
Rn(1) = case_d(3,case_ds);
Rn(rolls) =case_d(4,case_ds);
angle_travel_1=0; %stores last unwind position angle
angle_travel_2 =0; %stores last rewind position angle
z0=zeros(58,1);

%------------ Initial Conditions

if LC_unwind_control
    %LC control at roller 9
z = fsolve(@steadystateLC,(...
    rand(58,1)-.5)+[ones(rolls,1)*Initial_Vel(1); ones(spans,1)*...
    lineten; zeros(9,1)],  optimoptions('fsolve','Algorithm',...
    'Levenberg-Marquardt','TolFun',1e-16,'MaxIter',4000,'TolX',1e-6));
z(10:11,1) = Initial_Vel(10:11)';
z(1,1) = Initial_Vel(1)';
z([17,rolls],1) = Initial_Vel([17,rolls])';
z(rolls+8,1) = 2*lineten-z(rolls+9,1);
z(rolls+10,1) = [.5 .5]*z(rolls+[9 11],1);
z(rolls+14,1) = 2*lineten2-z(rolls+15,1);
z(rolls+0+18,1) = 2*lineten3-z(rolls+0+19,1);
z(50:51) = [Rn(1) Rn(rolls)]'; % set initial roll diameters
z(52) = z(1)/Rn(1);
z(53) = z(rolls)/Rn(rolls);

else
    %Dancer control at roller 4
    z = fsolve(@steadystateDan,rand(60,1),  optimoptions('fsolve',...
    'Algorithm','Levenberg-Marquardt','TolFun',1e-16,'MaxIter',4000,...
    'TolX',1e-6));
z(1,1) = Initial_Vel(1);
z(10:11,1) = Initial_Vel(10:11)';
z([17,rolls],1) = Initial_Vel([17 rolls])';
z(rolls+10,1) = [.5 .5]*z(rolls+[9 11],1);
z(rolls+14,1) = 2*lineten2-z(rolls+15,1);
z(rolls+18,1) = 2*lineten3-z(rolls+19,1);
z(rolls+3,1) = (f_da*5.032/12-z(rolls+4)*cos(theta_on(4))*...
    (15.925-1.5)/12) /((15.92+1.5)/12)/cos(pi-theta_in(4));
z(rolls+spans+[2],1) = [0]';
z(52:53) = [Rn(1) Rn(rolls)]'; % set initial roll diameters
z(54,1) = z(1)/Rn(1);
z(55,1) = z(rolls)/Rn(rolls);
end

velocities_ss = z(1:rolls,1);
tensions_ss = z(rolls+[1:spans]);%
if LC_unwind_control
```

```
310     %nothing
        dancer_ss = [];
312     motors_ss = z(54:58);
    else
314     dancer_ss = [z(50) 0]';
        motors_ss = z(56:60);
316 end
    %motor_inputs_ss(1) = motor_inputs_ss(1);
318 %state_derivative_ss = f_z(z);


320
    % Solve stuff
322 if LC_unwind_control
        z0 = [velocities_ss; ...
324         tensions_ss; ...
            z(50:51);...
326         z(52:53);...
            motors_ss]; %7x1
328 else
        z0 = [velocities_ss; ...
330     tensions_ss; ...
        dancer_ss;...
332     z(52:53);...
        z(54:55);...
334     motors_ss]; %7x1
    end
336
    figure(1);
338 subplot(2,2,1),plot(velocities_ss*60,'>')
    title('Velocities')
340 xlabel('Roller #')
    ylabel('fpm')
342 subplot(2,2,2),plot(tensions_ss,'o')
    title('Tensions')
344 xlabel('Span #')
    ylabel('lbf')
346 %ylim([9.2 10.2])
    if ~LC_unwind_control
348 subplot(2,2,3),bar(dancer_ss*12)
    title('Dancer Position (in) and Velocity (in/s)')
350 xlabel('')
    ylabel('magnitude')
352 end
    subplot(2,2,4),bar(motors_ss)
354 title('Motor Inputs (amps)')
    xlabel('Motor #')
356 ylabel('Magnitude')

358 % Set the output Steady-state
```

```
SS = z0;
% Set the output Steady-state derivative
h =waitbar(0, 'Please Wait');
hstep1 = 0.05*tfinal;
waitbar(0,h);
tstart = 0;
SSder = State_Controller(tstart,z0);
%RecordEvents(0,SS,1);

tstart=0:0.005:tfinal;
refine = 4;
options = odeset('Events',@events1,'Refine',refine,'RelTol',1e-5);%for
%ode45
%options = {'EventFunction',@PostProcessedSlip;'TimeStep',0.005;...
%    'SlipTrue',0}; %for odeBR4
tout = 0;
zout = z0.';
teout = [];
yeout = [];
ieout = [];
istep = 0; % limit the number of bounces off the full extension or full
simrun=1;
slipflag = 0;
flag1 = 0;
flag2 = 0;
shutdown_flag = 0; %for resetting the IndScurve function once

timetest = 0.005; % ---- updated each timestep to compare and current
%           time and wait until the next one to switch equations
%           close situation to 150 (arbitrary).
while tout(end) < tfinal && istep < 5500

    [time,z] = ode45(@State_Controller,...
        tstart([1 end]),z0);%,options); ,te,ye,ie
ie=[];
    nt = length(time);
  tout = [tout; time(2:nt)];    % dummy variable to store data
  zout = [zout; z(2:nt,:)];     % cummy variable to store data
%  teout = [teout; te];          % Events at tstart are never reported.
%  yeout = [yeout; ye];  %values of event function (1 of them will be 0 )
  if size(ie,1)>1
    ieout = [ieout; ie+10*slipflag];  %
  else
      ieout = [ieout; ie'+10*slipflag];
  end

  if time(nt) < tfinal %check to see if integration needs to continue
      %find which event caused the stop of integration
      if isempty(ie)
```

```
408           options = odeset(options,'MaxStep',0.005);
              tstart1 = [time(nt) (.005-rem(time(nt),.005))+time(nt)];
410           tstart = [tstart1(1) tstart1(2):0.005:tfinal];

412           [time,z] = ode45(@State_Controller,...
          tstart,z(end,:),options);
414           nt = length(time);
       tout = [tout; time(2:nt)];    % dummy variable to store data
416    zout = [zout; z(2:nt,:)];     % dummy variable to store data
       teout = [teout; te];          % Events at tstart are never reported.
418    yeout = [yeout; ye];  %values of event function (1 of them will be 0 )
       ieout = [ieout; ie];  %
420      else
              z0 = z(nt,:);
422           for i=1:numel(ie)
                  eventcase = ie(i);
424         if slipflag == 0
                  switch eventcase;
426           case 1
                  %T3 < T2*exp(-Mun*theta_wrap(3))
428               Ind(slipind) = -1;
                  options = odeset(options,'MaxStep',.005); %@events2);
430                 nm1=slipind-1;
                    n = slipind;
432               z0(rolls+nm1) = (z0(rolls+nm1)*L(nm1)+z0(rolls+n)*...
                      L(n))/(L(nm1)+L(n)*exp(Ind(n)*Mun(n)*theta_wrap(n)));
434               z0(rolls+n) = z0(rolls+nm1)*exp(Ind(n)*Mun(n)*theta_wrap(n));
                  %slipflag = 1;
436           case 2
                  %T3 < T2*exp(Mun*theta_wrap(3))
438               Ind(slipind) = 1;
                  options = odeset(options,'MaxStep',.005);%@events2);
440                 nm1=slipind-1;
                    n = slipind;
442               z0(rolls+nm1) = (z0(rolls+nm1)*L(nm1)+z0(rolls+n)*...
                      L(n))/(L(nm1)+L(n)*exp(Ind(n)*Mun(n)*theta_wrap(n)));
444               z0(rolls+n) = z0(rolls+nm1)*exp(Ind(n)*Mun(n)*theta_wrap(n));
                  %slipflag = 1;
446               end
           else
448               switch eventcase;
            case 1
450               % Returning from slip condition 1
                  funnames{1} = @NonSlipVelDer;
452               funnames{2} = @Span2;
                  funnames{3} = @Span2;
454               Ind(2) = 0;
                  options = odeset('Events',@events1);
456               %slipflag = 0;
```

```
            case 2
458             % Returning from slip condition 2
                funnames{1} = @NonSlipVelDer;
460             funnames{2} = @Span2;
                funnames{3} = @Span2;
462             Ind(2) = 0;
                options = odeset('Events',@events1);
464             %slipflag = 0;
                end
466         end
                switch eventcase
            case 3
468             istep = 5500;
            case 4
                istep = 5500;
472             end
            end
474     end
        z0=z0';
476     tstart1 = [time(nt) (.005-rem(time(nt),.005))+time(nt)];
        tstart = [tstart1(1) tstart1(2):0.005:tfinal];
478     istep = istep + 1;
  end
480 end

482 time = tout; % Return expected value names
  z = zout;  % Return expected value names
484 %[time,z,EEvents] = PostProcessedSlip(time,z);
  %-----------------------------------------------------------------------
486 %           S T A T E   D E R I V A T I V E   F U N C T I O N
  %-----------------------------------------------------------------------
488     function xdot = state_derive(T,x,u)

490         v = x(1:rolls,1);
            t = x(rolls+[1:spans],1);
492 %         theta1 = mod(x(sum(cnts(1:3))+1,1),2*pi());
  %         [r_1, r_1dot,r_1ddot,r_1ddot2] = Rollshape(theta1,...
494 %             x(sum(cnts(1:3))+2),Rn(unwind),rshape);
            xdot = x*0;
496
            % ---- roller velocity derivs General
498         if bearing_sliding_friction
                xdot(1:rolls,1) = 1./Jn'.*(-Bsldfn'.*Rn' + Rn'.^2.*([t; tf]...
500                 -[t0; t]) - Rn'.*ParasiticTorque');
            else
502             xdot(1:rolls,1) = 1./Jn'.*(-Bfn'.*v+ Rn'.^2.*([t; tf]-...
                    [t0; t]) - Rn'.*ParasiticTorque');
504         end
            % specific derivatives for certain rollers
```

```
506         xdot(9,1) = feval(funnames{1},t(8:9),v(9),9); %5-24-19 uses
            %                                          viscous friction model
508
            if LC_unwind_control
510             Jn(1) = inertia(x(50));
            xdot(1,1) = (1/Jn(1)*(-(Bfn(1)+Cmn(1))*v(1)/(2*pi*GR(1)*...
512             x(50))*GR(1)+ x(50)*(t(1)*GR(1)) + Kmn(1)*(u(1))+...
                Mn(1)*e_3*g*sin(x(rolls+spans+3,1))))*(2*pi*GR(1)*x(50));
514         else
                %dancer control added 2 states
516         xdot(1,1) = (1/0.056*(-Bfn(1)*v(1)/(2*pi*GR(1)*x(52))*GR(1)+...
                x(52)*(t(1)*GR(1)) +Kmn(1)*(u(1))+Mn(1)*e_3*g...
518             *sin(x(sum(cnts(1:3))+1,1))))*(2*pi*GR(1)*x(52));
            end
520         if nip10
                xdot(10,1) = 1/(Rn(10)^2*Jn_n(10)+Rn_n(10)^2*Jn(10))*( ...
522                 -(Rn_n(10)^2*Bfn(10)+Rn(10)^2*Bfn_n(10))*v(10)  ...
                    + Rn_n(10)^2*Rn(10)^2*(t(10) - t(9))+Rn_n(10)^2*...
524                 Rn(10)*Kmn(10)*u(2));
            else
526             xdot(10,1) = xdot(10,1) + 1/Jn(10)*(Kmn(10)*(u(2)*Rn(10)));
            end
528         xdot(11,1) = xdot(11,1) + 1/Jn(11)*(Kmn(11)*(u(3)*Rn(11)));
            xdot(17,1) = 1/(Rn(17)^2*Jn_n(10)+Rn_n(10)^2*Jn(17))*( ...
530             -(Rn_n(10)^2*Bfn(17)+Rn(17)^2*Bfn_n(10))*v(17)  ...
                + Rn_n(10)^2*Rn(17)^2*(t(17) - t(16))+Rn_n(10)^2*...
532             Rn(17)*Kmn(17)*u(4));
            if LC_unwind_control
534             Jn(rolls) = inertia(x(51));
                xdot(rolls,1) = (1/Jn(rolls)*(-(Bfn(rolls)+Cmn(rolls))*...
536                 v(rolls)/(2*pi*GR(5)*x(51))-...
                    x(51)*(t(spans)*GR(5))+Kmn(rolls)*(u(5))))*GR(5)*2*pi*x(51);
538         else
                xdot(rolls,1) = (1/.055*(-Bfn(rolls)*v(rolls)/(2*pi*x(53))-...
540                 x(53)*(t(spans)*GR(5))+Kmn(rolls)*(u(5))))*GR(5)*2*pi*x(53);
    %           xdot(rolls,1) = 1/Jn(rolls)*(-Bfn(rolls)*v(rolls)-...
542 %               x(53)^2.*(t(spans))+Kmn(rolls)*(u(5)*x(53)));
            end
544         if ~LC_unwind_control
            % ---- Dancer derivs
546         xdot(rolls+spans+1,1) = x(rolls+spans+2,1);
            xdot(rolls+spans+2,1) = 1/(Jpn)*(f_da*5.032/12 - ...
548             t(3)*cos(pi-theta_in(dancer))*(15.925+1.5)/12  -...
                t(4)*cos(theta_on(dancer))*((15.925-1.5)/12) - ...
550             Kn(4)*x(rolls+spans+1) - Cdn(4)*x(rolls+spans+2)  -...
                13.95/12*.317*g*sin(x(rolls+spans+1)));
552         end
            % ---- tension derivs
554         tens = [t0; t]; %include wound in tension
```

```
for i = 1:spans
    if ~LC_unwind_control && i == dancer
        [temp(3), temp(4)] = PendDancerLeng(x(sum(cnts(1:2))+1),...
            -x(sum(cnts(1:2))+2),...
            L(i),Larm(i),1.38/12,1.5/12,1.5/12); %.24
        sp0 = [v(i+[0 1])', tens(i+[0 1],1)', L(i)+...
            temp(3), ... %Length does not change
                temp(4), theta_on(i)]';
    elseif ~LC_unwind_control && i+1==dancer
        [temp(1), temp(2)] = PendDancerLeng(x(sum(cnts(1:2))+1),...
            -x(sum(cnts(1:2))+2),...
            L(i),Larm(i+1),3.084/12,1.5/12,1.5/12);
        sp0 = [v(i+[0 1])', tens(i+[0 1],1)', L(i)+...
            temp(1),... %Length does not change
            temp(2), theta_in(i+1)]';

    elseif i == unwind
        % d1 calc'ed on line 24
        l1 = sqrt(d1 - (r_1+Rn(2))^2);
        dl1 = -(r_1+Rn(2))*r_1dot*theta1dot/sqrt(d1-(r_1+Rn(2))^2);
        sp0 = [v(i+[0 1])',tens(i+[0 1],1)',l1,dl1]';
    else
        sp0 = [v(i+[0 1])' tens(i+[0 1],1)' L(i) 0 theta_on(i)]';
    end
    % for span2Lin sp0 = [sp0(from above) vel1_ss vel2_ss]
    sp0 = [sp0; z0(i+[0 1])];
    xdot(rolls+i,1) = span2Lin(T, sp0, E, A);
end

if LC_unwind_control
    xdot(rolls+spans+[1 2],1) = [-0.00536/12/(2*pi)...
        *(x(1,1)/x(50));...%(x(rolls+spans+3)-angle_travel_1);...
        0.00536/12/(2*pi)*(x(rolls,1)/x(51,1))];

    %angular speeds
    xdot(rolls+spans+3,1) = x(1,1)/x(50);
    xdot(rolls+spans+4,1) = x(rolls,1)/x(51);
else
    %dancer controlled
    xdot(rolls+spans+2+[1 2],1) = [-0.00536/12/(2*pi)...
        *(x(1,1)/x(52));...
        0.00536/12/(2*pi)*(x(rolls,1)/x(53))];

    %angular speeds
    xdot(rolls+spans+5,1) = x(1,1)/Rn(1);
    xdot(rolls+spans+6,1) = x(rolls,1)/Rn(rolls);
end
% xdot = xdot';
```

```
604     end
%-------------------------------------------------------------------------
606 %           S T A T E   C O N T R O L L E R   F U N C T I O N
%-------------------------------------------------------------------------
608     function [ydot] = State_Controller(t,y)
        %[t,y] = PostProcessedSlip(t,y);
610         ydot = y*0;
        % Setup inputs
612         if t<0.5
            %Step in Tension
614         r1 = [100 0 0]'/60;
            r2 = r1;
616         else
            %Step in Tension
618             r1 = [100 0 0]'/60;
                r2 = r1;
620             lineten = 11;
            end
622         % get state derivative
            if LC_unwind_control
624             % update the steady-state velocities with r1 and r2
                z0(1,1) = r1(1);
626             z0(2:rolls,1)= r2(1)*ones(rolls-1,1);
                ydot(1:sum(cnts(1:4)),1) = state_derive(t,y(1:sum(cnts(1:4)),...
628                 1),y(sum(cnts(1:4))+[1:cnts(5)],1));
                %tension errors
630             F(1)=[.5 .5]*y(rolls+[8 9],1);
                Fdot(1)=[.5 .5]*ydot(rolls+[8 9],1);
632             F(2)=[.5 .5]*y(rolls+0+[18 19],1);
                Fdot(2)=[.5 .5]*ydot(rolls+0+[18 19],1);
634             %speed errors
                errors = [ r1(1)-y(1), r2(1)-y(10), r2(1)-y(11), r2(1)-...
636                     y(17), r2(1)-y(rolls);...
                    r1(2)-ydot(1), r2(2)-ydot(10), r2(2)-ydot(11), r2(2)-...
638                     ydot(17), r2(2)-ydot(rolls)];

640             %trim speed calc
                TrimV(1) = (Kp(1)*(-Fdot(1))+Ki(1)*(lineten-F(1)));
642             TrimV(2) = (Kp(6)*(-Fdot(2))+Ki(6)*(lineten-F(2)));
                %current calc
644             ydot(sum(cnts(1:4))+1,1) = (Kp(2)*(errors(2,1)+TrimV(1)*0)+...
                    Ki(2)*(errors(1,1)+TrimV(1)));
646             ydot(sum(cnts(1:4))+2,1) = Kp(3)*errors(2,2)+Ki(3)*errors(1,2);
                ydot(sum(cnts(1:4))+3,1) = Kp(4)*errors(2,3)+Ki(4)*errors(1,3);
648             ydot(sum(cnts(1:4))+4,1) = Kp(5)*errors(2,4)+Ki(5)*errors(1,4);
                ydot(sum(cnts(1:4))+5,1) =  Kp(7)*(errors(2,5)+TrimV(2)*0)+...
650                 Ki(7)*(errors(1,5)+TrimV(2));

652         else
```

```
            %dancer Controlled
654         ydot(1:sum(cnts(1:5)),1) = state_derive(t,y(1:sum(cnts(1:5)),...
                1),y(sum(cnts(1:5))+[1:cnts(6)],1));
656
658         F(1)= -y(rolls+spans+1,1);
            Fdot(1)=-y(rolls+spans+2,1);
660         F(2)=[.5 .5]*y(rolls+[18 19],1);
            Fdot(2)=[.5 .5]*ydot(rolls+[18 19],1);
662
            % find errors in speed
664         errors = [ r1(1)-y(1), r2(1)-y(10), r2(1)-y(11), r2(1)-...
                    y(17) r2(1)-y(rolls);...
666             r1(2)-ydot(1), r2(2)-ydot(10), r2(2)-ydot(11), r2(2)-...
                    ydot(17) r2(2)-ydot(rolls)];
668
            TrimV(1) = (Kp(1)*(Fdot(1))+Ki(1)*(F(1)));
670         TrimV(2) = (Kp(6)*(-Fdot(2))+Ki(6)*(lineten-F(2)));
            %current calc
672         ydot(sum(cnts(1:5))+1,1) = (Kp(2)*(errors(2,1)+TrimV(1)*0)+...
                Ki(2)*(errors(1,1)+TrimV(1)));
674         ydot(sum(cnts(1:5))+2,1) = Kp(3)*errors(2,2)+Ki(3)*errors(1,2);
            ydot(sum(cnts(1:5))+3,1) = Kp(4)*errors(2,3)+Ki(4)*errors(1,3);
676         ydot(sum(cnts(1:5))+4,1) = Kp(5)*errors(2,4)+Ki(5)*errors(1,4);
            ydot(sum(cnts(1:5))+5,1) = Kp(7)*(errors(2,5)+TrimV(2)*0)+...
678             Ki(7)*(errors(1,5)+TrimV(2));
        end
680
        y0=ydot;
682     % update waitbar
        if t> hstep1
684         waitbar( hstep1/tfinal/2, h)
            hstep1 = hstep1 +.1*tfinal;
686     end
    end
688 %---------- NonSlip idler vel. derive function
    function dv = NonSlipVelDer(tens,v,ind)
690     dv = 1/Jn(ind)*(-Bfn(ind)*v+ Rn(ind)^2*([-1 1]*tens)...
            - ParasiticTorque(ind)*Rn(ind));
692 end
    %---------- Slip idler vel. deriv function
694 function dv = SlipIdlerVelDer(tens,v,ind)
    Fn = [-cos(theta_in(ind)) cos(theta_on(ind))]*tens(1:2,1);
696 if v < 0.001
        Pars = 0;
698 else
        Pars = ParasiticTorque(ind);
700 end
    dv = 1/Jn(ind)*(-Bfn(ind)*v + Ind(ind)*Mun(ind)*Fn*Rn(ind)^2 ...
```

```
            - Pars*Rn(ind));
    end
    %---------- NonSlip Tension deriv span 1
    %Just use Span.m like normal
    %---------- Slip Tension deriv span 1
    function dt = SlipTensionIncoming(Props)
        %Taken from WTS discussion of Turn bars
        %Props = [L1+L2*e^(mu*alpha), E*A, t0, T2, V1, V3]
        L1 = Props(1); EA = Props(2); Tnm1 = Props(3); Tnp1 = Props(4);
        Vn = Props(5); Vnp2 = Props(6);
        dt =1/L1*(Vnp2*(EA - Tnp1) - Vn*(EA - Tnm1));
    end
    %---------- Slip Tension deriv span 2
    function dt = SlipTensionOutgoing(Props)
        %Taken from WTS discussion of Turn bars
        %Props = [dT1, exp(mu2*alpha2)];
        dT1 = Props(1); emu = Props(2);
        dt = dT1*emu;
    end
    %---------- Events function
    function [value,isterminal,direction] = events1(t,y)
% Locate the time when tension ratio violates capstan equation
% and stop integration.
    %slipflag = 1;
    %Ind(2)  = 0;
    nm1 = slipind-1;
    n = slipind;
    testforTension = find(y(rolls+[1:spans])<0);

    value = [ ...%y(sum(cnts(1:2))+1) - 50/12, ...
        %y(sum(cnts(1:2))+1) + 5/12, ...
        y(rolls+3), ...%
        y(rolls+4) , ...
        y(rolls+1)+20, ...
        y(rolls+2)+20
        ]';
    isterminal = [1 1 1 1]';   % stop the integration
    direction = [-1 1 -1 -1]';   % [positive negative] direction
    end

    function z = steadystateLC(x)
        c = state_derive(0,[...
            Initial_Vel(1);...
            x(2:9,1);
            Initial_Vel(10:11)'; ...
            x(12:16);...
            Initial_Vel(17);...
            x(18:24);... %end of speeds
            Initial_Vel(25);...
```

```
            x(26:28);...
752         x(29:32,1);...  %29=span 4
            2*lineten-x(34,1);...  %33 =span 8
754         x(34);...
            [.5 .5]*x([34,36],1);
756         x(36:38);...
            2*lineten2-x(40,1);...
758         x(40:42,1);... %42= span 17
            2*lineten3-x(44,1);...  %43 = span 18
760         x(44:49,1);...
            [Rn(1) Rn(rolls)]';... %variable radii
762         x(1)/Rn(1);... %omega(1)
            x(rolls)/Rn(rolls);...
764         ],...%
            x(54:58)); %motor currents
766     z = c;
        z(52) = x(52) - z(1)/Rn(1);
768     z(53) = x(53) - z(rolls)/Rn(rolls);
    end
770 function z = steadystateDan(x)
        c = state_derive(0,[...
772         Initial_Vel(1);...
            x(2:9,1);
774         Initial_Vel(10:11)'; ...
            x(12:16);...
776         Initial_Vel(17);...
            x(18:24);...
778         Initial_Vel(25);... %end of speeds
            x(26:27);...
780         (f_da*5.032/12-x(29)*cos(theta_on(4))*(15.925-1.5)/12)/...
                ((15.925+1.5)/12)/cos(pi-theta_in(4));... %span tensions
782         x(29:33,1);...  %29=span 4
%           2*lineten-x(34,1);...  %33 =span 8
784         x(34);...
            [.5 .5]*x([34,36],1);
786         x(36:38);...
            2*lineten2-x(40,1);...
788         x(40:42,1);... %42= span 17
            2*lineten3-x(44,1);...  %43 = span 18
790         x(44:49,1);...
            x(50,1);...
792         0;...
            [Rn(1) Rn(rolls)]';... %variable radii
794         x(1)/Rn(1);... %omega(1)
            x(rolls)/Rn(rolls);...
796         ],...%
            x(56:60)); %motor currents
798     z = c;
        z(54) = x(54) - z(1)/Rn(1);
```

```
800         z(55) = x(55) - z(rolls)/Rn(rolls);
        end
802         function newTen = Whitworth(T,Tnm1,L,Lnm1,INDmualpha)
            %applies the Whitworth equation to the tensions
804         %INDmualpha = IND(i,j)*Mun(p)*theta_wrap(p) where IND(i,j) is
            %the +/-1 value from the test1 or test2 above, Mun is the
806         %coefficient of friction for roller p, and theta_wrap(p) is the
            %wrap angle in radians for roller p.
808             newTen = (Tnm1*Lnm1+T*L)/(Lnm1+L*exp(INDmualpha));
            end
810
            function phi = PHIi(phi_nm1,ind_n,mu_n,alpha_n)
812             %Multiplier in slip calculation
                phi = phi_nm1*exp(ind_n*mu_n*alpha_n);
814         end

816     function [k] = GainSched(R,V0,z,om,ind)
            %quickly calc my method of finding gains.
818         if ind == rolls
                ind=ind-1;
820             Li =sum(L(ind-[0:5]));
                R_b=R/1.5*12;
822             ind=ind+1;
                J = inertia(R);%1+(.055*R_b^4)*GR(5)^2;
824             num=-(Kmn(ind)*A*E*R/sum(Li)/J);
            else
826             Li = sum(L(1:8));
                R_b=R/1.5*12;
828             J = inertia(R);%1+(.055*R_b^4)*GR(1)^2;
                num=-(Kmn(ind)*A*E*R/sum(Li)/J);
830         end
            %speed gains first
832         %Kmn(ind) = Kmn(ind)/GR(1)
            kp_speed = (-(Bfn(ind)+Cmn(ind))+2*z*om*J)/(Rn(ind)*Kmn(ind)*GR(1));
834         ki_speed = (om^2*J)/(GR(1)*Rn(ind)*Kmn(ind));
            % speed loop tf num/ den
836         num1 = GR(1)*Rn(ind)*Kmn(ind)/J;
            den1 = [ 1 (Bfn(ind)+Cmn(ind))/J];
838         num1a = conv(num1,[kp_speed ki_speed]);
            den1a = [den1 0];
840         num1loop = num1a;
            den1loop = [den1a ] + [0  num1a(1) num1a(2)];
842       % span tension transfer
          if ind == 25
844           num2 = E*A/Li;
          else
846           num2 = -E*A/Li;
          end
848       den2 = [1 V0/Li];
```

```
            num3 = conv(num2,num1loop);
850         den3 = conv(den2,den1loop);
            %Find the 2nd order approx. of the 37th order TF
852         [r_num,r_den] = RouthApprox(num3,den3,2);
            if ind == 1
854             Again = [om*z -r_num(end-1) -2*z*om+r_den(2);...
             (2*z^2*om^2-r_num(end-1)/r_num(end)*om^3*z) -r_num(end) ...
856              -om^2+r_den(end)];
                rAgain = rref(Again);
858             kr = rAgain(1,3)
                kp_ten = rAgain(2,3);
860             ki_ten = kr*om^3*z/r_num(end);
            else
862             again =[(-2*z^2*om^2+(r_num(end-1)*z*om^3)/r_num(end)), ...
                    (r_num(end)), -r_den(end)+om^2;...
864                  -z*om, r_num(end-1), -r_den(end-1)+2*z*om];
                arf = rref(again);
866             kr = (arf(1,3));
                kp_ten = arf(2,3);
868             ki_ten = kr*z*om^3/r_num(end);
            end
870         k=[(kp_ten),ki_ten,kp_speed,ki_speed]/60;
        end
872     function J = inertia(R)
           % calculate inertia based on Euclid parames and radius R (ft)
874        I_spind = 9.942e-3; %slug*ft^2 at spindle
           I_motor = 0.048; %slug*fT^2 at motor
876        I_gr = 6.695e-3; %slug*ft^2 at motor
           J = (I_spind + rho*width_w/2*pi*(-(1.5/12)^4+(R)^4))*GR(1)^2+...
878            I_motor+I_gr;
        end
880 %------------ Plotting ------------------------------------------------

882 velocities_out = z(:,1:rolls)*60; %fpm
    tensions_out = z(:,rolls+1:rolls+spans);
884 if LC_unwind_control
        dancer_out = z(:,rolls+[8,9,18,19])*[.5 .5 0 0; 0 0 .5 .5]';
886     motor_inputs_out = z(:,sum(cnts(1:4))+[1:cnts(5)]);
        % Rollers and Spans of interest
888     velindex = [1 10 11 17 25];
        tenindex = [1 8 9 18 19];
890 else
        dancer_out = z(:,rolls+spans+[1])*180/pi;
892     motor_inputs_out = z(:,sum(cnts(1:5))+[1:cnts(6)]);
           % Rollers and Spans of interest
894     velindex = [1 10 11 17 25];
        tenindex = [1 3 4 18 19];
896 end
    r1 = IndScurveInput(0,z0(1),deltaV(1),30/60,30/60,1);
```

```
refspeed = zeros(length(time),1);
shutdown_flag = 0;
for i= 1:length(time)
    t=time(i);
    if (t > 20) && ~shutdown_flag
        % put scheduling info here later
        r1 = IndScurveInput(t,400/60,0,30/60,30/60,1); % for unwind speed
        shutdown_flag = 1;
    end
    if t> 20
        r1 = IndScurveInput(t,400/60,0,30/60,30/60);
    else
        r1 = IndScurveInput(t,z0(1),deltaV(1),30/60,30/60);
    end
    refspeed(i) = r1(1);
end
waitbar( .6, h)
figure(2);
subplot(2,2,1)
plot(time,velocities_out(:,velindex)-repmat(refspeed,1,5)*60)
title('velocity');
legend(['v' num2str(velindex(1))],['v' num2str(velindex(2))],...
    ['v' num2str(velindex(3))],['v' num2str(velindex(4))],['v'...
    num2str(velindex(5))],'location','best');
grid;
subplot(2,2,2)
plot(time,tensions_out(:,tenindex))
title('tension');
legend(['t' num2str(tenindex(1))],['t' num2str(tenindex(2))],...
    ['t' num2str(tenindex(3))],['t' num2str(tenindex(4))],['t'...
    num2str(tenindex(5))],'location','best');%
grid;
subplot(2,2,3)
if LC_unwind_control
    plot(time,dancer_out);
title('Load Cells');
legend('Unwind LC','Rewind LC','location','best')%,'xdot');
grid;
else
plot(time,dancer_out);%,time,dancer_out(:,1))
title('dancer');
legend('x (deg)','location','best')%,'xdot');
grid;
end
subplot(2,2,4)
plot(time,motor_inputs_out);%,time,dancer_out(:,1))
title('Motor Inputs');
legend('Unwind','SWL','SWF','PR','Rewind','location','best')%,'xdot');
ylabel('Amps')
```

```
grid;
waitbar( 1, h)
if nargout>4
    Events = RecordEvents(0,0,0,2);
end
close(h)
end
```

## J.4 High Speed Web Line Simulation

This simulation is the 6th iteration of the simulation code for the HSWL. It is a simulation that includes noise on the tension feedback loop. The code in Listing J.4 is specific to experiments with PET material. There were two more codes, a 'bn' and a 'cn', that were the same code with the material changed to 6 inch wide Tyvek and 24 inch wide Tyvek, respectively. This code was used to obtain the simulation data shown in chapter 4.

Listing J.4: The High-Speed Web Line Simulation Code

```
function [ SS, SSder, time, z, Events ] = HSWL06an( tfinal,Vmax )
%The HSWL06an function is a simulation of the High-Speed Web Line at OSU.
%The WHRC owns this line.  It is capable of 0-2500 FPM, web widths up to 30
%inches across, and tensions up to 90 lbf.
%   This function has several stages:  Setup, Line parameters, Run
%   Parameters, Steady-State, Integration, and Plotting.
%
%   Subfunctions which support the Setup stage are:  ElementPropReadIn()-
%   external, RecordEvents() - external, motor() - external (ParentRoll()
%   - called by motor() ), Gains() - external, BRLeadLag() - external,
%   DucoteyGood() - external,
%   Subfunctions which support the Line Parameters stage are:  none
%   Subfunctions which support the Run Parameters stage are:  PrintLine() -
%   external, WrapAngle() - external
%   Subfunctions which support the Steady-State stage are:  fsolve -
%   external and MATLAB, steadystate().
%   Subfunctions which support the Integration stage are:  ode45() -
%   external and MATLAB, odeBR4() - external (not used), Controller(),
%   IndScurveInput() - external, state_derive(), FrictionCalc(),
%   Whitworth() (not used), PostProcessedSlip() (not used), PHIi()
%   (not used), TorqueLimit(), LimitTrim(),
%   Subfunctions which support the Plotting stage are: legendmaker(),
%   IndScurveInput() - external
%
%   Version 01 - Startup 0-Vmax ft/min,  Unwind load cell feedback (R3),
%       Rewind load cell feedback (R27)
%
%   Version 02 - Startup 0-Vmax ft/min,  Unwind dancer feedback (R8),
```

```
%       Rewind load cell feedback (R27), f_da is the air pressure times the
%       surface area of the piston
%
%   Version 03 - Startup 0-Vmax ft/min,  Unwind dancer feedback (R8),
%       Rewind load cell feedback (R27), f_da is the air pressure times the
%       surface area of the piston
%
%   Version 03a - Switching the PID control back to traditional integrate
%       error and calculate inputs first before going into derivative
%
%   Version 04a - Copied 03a but will use gains calculated my way.
%
%   Version 05a - Copied 03a for PID method and will switch the Unwind
%       feedback device to a load cell at roller 3.  The Pull roll will be
%       set up with gains calculated by Rockwell.  Added 3 kinds of
%       friction model to develop the web velocity for tension change.
%       Made the Torque calculation depended on every 20ms update of the
%       unwind radius for calculating inertia and radius.
%
%   Version 06a - Copied 05a but will use gains calculated my way.
%
%   Version 06an - Copied 05a and added noise to LC feedback
%
%


%-------------------------------------------------------------------------
% S E T U P
%-------------------------------------------------------------------------
if nargin == 1
    Vmax = 400/60; %ft/sec
elseif nargin <1
    % Take care of the situation that the requested simulation duration is
    % not given
    tfinal = 30; %sec
    Vmax = 400/60; %ft/sec
end
if Vmax>42 %line speed maximum in ft/s
    Vmax=Vmax/60; %convert to ft/s from fpm
end
file='C:\Users\quietman\Documents\MATLAB\WTS\HSWL03.Dat';
%      filename, number of rolls, number of spans
[r,s,c]=ElementPropReadIn(file,29,28);
Rolls = r{1};
Spans = s{1};
funnames = {@NonSlipVelDer,@Span2,@Span2,@Span2,@Span2};

fourPlotTitle = '0-800FPM Startup, PET, Unwind-LC,Rewind-LC, Reish';

```

```
RecordEvents(0,0,0,1); %Records state when slip is found
% % DM01 is set up in software to run as a 36.9Hp motor at 1420 RPM
%DM01 = motor('Unwind L2875b'); % 6in wide Tyvek
DM01 = motor('Unwind L2875b PET');
DM01.set('R',14/12/2);
DM09 = motor('L2153');
%set up in software to run per nameplate values: 30Hp, 1150 RPM
DM29 = motor('Rewind L2875 PET');
DM29.set('R',2/12);
DM01.set('speed',Vmax*60);
DM09.set('speed',Vmax*60);
DM29.set('speed',Vmax*60);
%17.6375 ft to dancer, 4.4497 ft for loadcell
b_boulter = {'BW',30;'ratio_lead_to_crossover',1/4;...
    'omega_co',30}; %unw and rew motors
c_boulter = {'BW',15;'ratio_lead_to_crossover',1/4;'omega_co',15}; %PR motor
e_Boulter = {'ratio_lead_to_crossover',1/5;'Kp_ten',60;...
    'omega_co',10;'EC_radius',2/12};%load cell
Reish_speed = {'omega_n',16;'zeta',.79077};
Reish_ten = {'Gr',1;'omega_n',26;'zeta',.9;'tenLoop_sample_ratio',1/40;...
    'span spring',DM01.Roll.get('EA');'LS',Vmax;'L',4.4497;...
    'EC_radius',2/12;'final_value_adj',10};
G_Unwind_speed = Gains('TEST-03');
G_Unwind_Ten = Gains('TEST-02');
G_PR_speed = Gains('TEST-04');
G_Rewind_speed = Gains('test-01');
G_Rewind_Ten = Gains('test-01');
TenComp1 = BRLeadLag({'T1',1/1.4;'T2',1/14;'up sat',100;'low sat',-100});
TenComp2 = BRLeadLag({'T1',1/1.4;'T2',1/14;'up sat',100;'low sat',-100});
%-------------------------------------------------------------------------
% L I N E   P A R A M E T E R S
%-------------------------------------------------------------------------
rolls = 29;
spans = 28;
dancer = [];
% acc = [];
unwind = [1];
rewind = [rolls];
driven = [unwind, 9, rewind];
idlers = 1:rolls;
idlers = setdiff(idlers,driven); %roller numbers that are not driven
eccent = [1];
Ind = zeros(rolls,1);
cnts = [rolls spans 0 5 2 3];
% cnts is rolls; spans; 0 dancer states, 1 eccentric roller position unwind
% motor radians per second and summed with 3 lead-lag compensator state; 2
% tension error states, 3 motor speed error states

accel_rate = 80/60; %feet per min/sec in ft/s^2  Used in IndScurve
```

```
jerk_rate = 200/60; %feet per min/sec^2 in ft/s^3
%Use roller built into DucoteyGood object
DG = DucoteyGood('ReishWebPET-1','ReishRoller2');
%DG = DucoteyGood('ReishWebTyv-3','ReishRoller2');
order = [2:8,10:28]; %list of idlers for slip.
g = 32.174; %ft/s^2 gravity
radps2rpm = 30/pi;
rpm2radps = 1/radps2rpm;
y0 = zeros(60,1);
%-------------------------------------------------------------------------
% R U N   P A R A M E T E R S
%-------------------------------------------------------------------------
Initial_Vel = ones(1,rolls)*00/60; %initial speeds of rollers in ft/s
deltaV = ones(1,rolls)*Vmax; %increase in speed in ft/s
t0 = 30; %lbf wound in tension on the Unwind roll
tf = 24; %lbf wound in tension on the Rewind roll
lineten = 24; % line tension setpoint Unwind section
lineten2 = 24; %line tension setpoint of rewind section
nip9 = 1; % 1 use nip, 0 don't use nip NOTE-> HSWL requires use of this nip
SFDR = 0;  % 1 use SFDR in slip check, 0 normal Whitworth
%boolean to toggle between viscous and sliding friction models
Use_bearing_sliding_friction = 0;
%boolean to toggle between parasitic torque and none
Use_parasitic_torque = 0;
sim = 0; %turn off noise for initialization
%---------- Roller params ----------
%Rollers
Rn = Rolls(3,:)/12/2;% radius in ft for all rollers
Jn = Rolls(13,:); %lbf*ft*s^2
Cmn = Rolls(19,:); % motor drag constant
Cmn(1) = .02; %2% of speed is drag (windage)
Kmn = zeros(1,rolls);
Kmn(1) = DM01.get('K_m');
Kmn(9) = DM09.get('K_m');
Kmn(29) = DM29.get('K_m');
Bfn = Rolls(20,:)*(1-0); % lbf*ft*sec, Bearing friction   ----------<
Mn = Rolls(10,:)/g; %slugs
Mun = [ones(1,rolls)]*.2; % static friction max from measurements
e_3 = 0.0/12; %ft eccentricity
Rn(1) = DM01.get('R');
Rn(29) = DM29.get('R');
DM01.set('J_load','unwind');%1.3/g+0.016);
DM09.set('J_load',0.351756); % add inertia of the roller slug-ft^2
DM09.calculateJ_sec();
DM29.set('J_load','rewind');%1.3/g+0.016);
Jn(1) = DM01.calculateInertia(Rn(1),1);
Jn(9) = DM09.calculateInertia(Rn(9),1); %calc inertia from roll perspective
Jn(29) = DM29.calculateInertia(Rn(29),1); %
rated_torque = [DM01.get('rated torque') DM09.get('rated torque')...
```

```
    DM29.get('rated torque')];
GR = DM01.get('GR');
Rn_n = [zeros(1,8) 1 zeros(1,19) 1 ]*4/24; %Nip radius in feet
Jn_n = [zeros(1,8) 1 zeros(1,19) 1 ]*4.635e-3; %nip inertias
Bfn_n = [zeros(1,8) 1 zeros(1,19) 1 ]*0.0006; %nip bearing friction

% Bfn = [0.00060729 0.00016691 0.00126336 0.00017803 0.00004665 ...
%     0.00002655 0.00050257 0.00062522 0.00061941 0.00049733  ...
%     0.00061733*ones(1,15)] ;
Bsldfn = [1 0.05227297  0.41489625      0.17357908      0.23150706 ...
    0.13421975 0.16057590       0.20012461      0.20024913 .1 .1 ...
    0.15779922*ones(1,14)]; %mu's from SpinDownTest.m
Bsldfn = Bsldfn*diag([38*(1.5/24) 12.89*.915/12*[1 1] ...
    1.188*.915/12 .876*.915/12*[1 1] 12.89*.915/12*[1 1 1] ...
    78.454*.915/12*[1 1] 12.89*.915/12*[1 1 1 1 1] 42*1.5/24 ...
    12.89*.915/12*ones(1,8)])*.05;
theta_arm = zeros(1,rolls);
theta_arm(dancer) = pi;
ParasiticTorque = [0 0*Rn(2) 0 0.00*Rn(4) 0 0 0 0 Rn(9)*1.419*0 ...
    zeros(1,10) Rn(20)*0 zeros(1,9)];
friction_params = {};
if Use_bearing_sliding_friction
    friction_params{1} = 'sliding_friction';
else
    friction_params{1} = 'viscous_friction';
end
if Use_parasitic_torque
    friction_params{2} = 'Parasitic_Torque';
end
OverWrap = zeros(1,rolls);
OverWrap(1) = 1; %Set over wrap to 1 (-1 for under wrap) for unwind
OverWrap(29) = 1; %Set over wrap to 1 (-1 for under wrap) for rewind
%---------- Span params ----------
L = Spans(4,:)/12; %feet
L(7:8) = ([34 10]*0 + [44 20]*1)/12; %span length fully extended in LC con
EA = DM01.Roll.get('EA');
E = DM01.Roll.get('E');
A = EA/E;
web_thick = DM01.Roll.thickness;
rot_dir = [-1 -1 1 -1 -1 -1 1 -1 1 1 -1 1 -1 1 -1 1 1 ...
    -1 1 -1 1 -1 1 1 1 -1 1 -1 -1]; %forward web direction CW = -1, CCW = 1
[sdta, theta_in,theta_on] = PrintLine(file,rolls,rot_dir,5,theta_arm);
theta_in=theta_in(1:rolls);
theta_on = theta_on(1:rolls);
theta_wrap = WrapAngle(sdta,Rn);
DG.set('beta',theta_wrap(20));
frictChar= [4000,.0006,.0002]; %used in the web friction model
%_-----------------------------------------------------------------
%  SET CONTROLLER GAINS
```

```
225 %-------------------------------------------------------------------------
    I = DM01.calculateInertia(Rn(1),0);%DM01.get('J')+R01.processInertia(Rn(1));
227 G_Unwind_speed.processGains(8,I,Reish_speed);
    s_gains = G_Unwind_speed.calculate(I,DM01);
229 G_Unwind_Ten.processGains(9,I,Reish_ten);
    I = DM29.calculateInertia(Rn(29),0); %calculate total inertia at the motor
231 G_Rewind_speed.processGains(5,I,b_boulter);
    G_Rewind_Ten.processGains(6,I,e_Boulter);
233 G_Unwind_speed.set('R',DM01.R);
    t_gains = G_Unwind_Ten.calculate(Jn(1),G_Unwind_speed);
235
    G_PR_speed.processGains(5,DM09.calculateInertia(Rn(9),0),c_boulter);
237 TenComp1.set({'T1',1/39;'T2',1/8});
    %unwind tension loop gains
239 Kp(1)= t_gains(1);
    Ki(1)= t_gains(2);
241 Kd(1)=0/12;%
    %unwind speed loop gains
243 Kp(2)=s_gains(1);
    Ki(2)=s_gains(2);
245 Kd(2)=0.0/60;
    %Master Speed speed loop gains
247 s_gains = G_PR_speed.calculate(DM09.calculateInertia(Rn(9),0),DM09);
    Kp(3) = s_gains(1);
249 Ki(3) = s_gains(2);
    Kd(3) = 0/60;
251 s_gains = G_Rewind_speed.calculate(I,DM29);
    t_gains = G_Rewind_Ten.calculate(Jn(29),G_Rewind_speed);
253 TenComp2.set({'T1',1/40;'T2',1/20});
    %Rewind tension loop gains
255 Kp(4) = t_gains(1);
    Ki(4) = t_gains(2);
257 Kd(4) = 0/60;
    %Rewind speed loop gains
259 Kp(5) = s_gains(1);
    Ki(5) = s_gains(2);
261 Kd(5) = 0/60;
    disp(['J_sec: ' num2str(DM01.get('J_sec'))]);
263 disp([Kp;Ki])
    Tval =0;
265 yval =0;
    TrimV = [0 0 0];
267 ldlg1 = 0;
    ldlg3 = 0;
269 ldlg4 = 0;
    rad = Rn(1);
271 stepcnt = 1;
    lowspeed_start = 1;
273 below5 = 1;
```

```
started = 0;
hstep1 = 0.05*tfinal;
tstart=0:0.02:tfinal;
stepcntmax = length(tstart);
%------------ Initial Conditions --------------------------------------------
xi = [rand(1,rolls)*0,ones(1,spans)*lineten,rand(1,sum(cnts(3:4))),...
    0,0,0,0,0];
z = fsolve(@steadystate,xi',  optimoptions('fsolve','Algorithm',...
    'Levenberg-Marquardt','TolFun',1e-18,'MaxIter',4000,'TolX',1e-8));
z([driven],1) = Initial_Vel(driven)';
z(rolls+2,1) = 2*lineten-z(rolls+3);
z(rolls+26,1) = 2*lineten2-z(rolls+27,1);
z(sum(cnts(1:4))-4) = z(eccent)/Rn(eccent);
z(sum(cnts(1:4))-3) = z(unwind,1)/Rn(unwind)*radps2rpm;
z(sum(cnts(1:4))) = Rn(1);
z(sum(cnts(1:4))+1) = 0;
velocities_ss = z(1:rolls,1);
tensions_ss = z(rolls+[1:spans]);
motors_ss = z(sum(cnts(1:4))+[1:sum(cnts(5:6))]);
% Solve stuff
z0 = [velocities_ss; ...
    tensions_ss; ...
    ...%dancer_ss;...
    z(sum(cnts(1:4))-4);... %unwind rotational position
    z(sum(cnts(1:4))-3);... %unwind rotational rate in rpm
    [.5 .5]*tensions_ss(26:27);... %z(sum(cnts(1:4)));...
    [.5 .5]*tensions_ss(2:3);...
    z(sum(cnts(1:4))); ... % unwind radius in ft
    motors_ss]; %5x1
figure(1);
subplot(2,2,1),plot(velocities_ss*60,'>')
title('Velocities')
xlabel('Roller #')
ylabel('fpm')
subplot(2,2,2),plot(tensions_ss,'o')
title('Tensions')
xlabel('Span #')
ylabel('lbf')
subplot(2,2,3),bar([[.5 .5]*tensions_ss(2:3) [.5 .5]*tensions_ss(26:27)])
title('Load Cell (lbf)')
xlabel({'3','27'})
ylabel('magnitude')
subplot(2,2,4),bar(motors_ss)
title('Error States ')
ylabel('Magnitude')
SS = z0;
h =waitbar(0, 'Please Wait');
stepcnt=1;
waitbar(0,h);
```

```
RecordEvents(0,z0,[Kp Ki zeros(1,15) 10])
%-------------------------------------------------------------------------
% B E G I N   I N T E G R A T I O N
%-------------------------------------------------------------------------
sim = 1; %turn on noise for simulation
options = odeset('RelTol',1e-5);%for ode45
[time,z] = ode45(@Controller,tstart([1:end]),z0,options);
if nargout == 5
    Events = RecordEvents(0,0,0,2);
end
%-------------------------------------------------------------------------
% C O N T R O L L E R   F U N C T I O N
%-------------------------------------------------------------------------
    function ydot = Controller(T,y)
        ydot = y*0;
        % Setup inputs
        if T == 0  %Initialize IndScurveInput function
            % for unwind speed
            r1 = IndScurveInput(T,Initial_Vel(1),deltaV(1),accel_rate,...
                jerk_rate,1);
            % for Driven Speed
            r2 = IndScurveInput(T,Initial_Vel(9),deltaV(9),accel_rate,...
                jerk_rate,1);
            % for Rewind Speed
            r3 = IndScurveInput(T,Initial_Vel(rolls),deltaV(rolls),...
                accel_rate,jerk_rate,1);
        end
        % get reference values
        if T > 0 && T<10
        %for unwind speed
        r1 = IndScurveInput(T,Initial_Vel(1),deltaV(1),accel_rate,...
                jerk_rate);
        % for driven speed
        r2 = IndScurveInput(T,Initial_Vel(9),deltaV(9),accel_rate,...
                jerk_rate);
        % for Rewind Speed
        r3 = IndScurveInput(T,Initial_Vel(rolls),deltaV(rolls),...
                accel_rate,jerk_rate);
        else
            %for unwind speed
            r1 = IndScurveInput(T,Initial_Vel(1),deltaV(1),...
                accel_rate,jerk_rate);
            % for driven speed
            r2 = IndScurveInput(T,Initial_Vel(9),deltaV(9),...
                accel_rate,jerk_rate);
            % for Rewind Speed
            r3 = IndScurveInput(T,Initial_Vel(rolls),deltaV(rolls),...
                    accel_rate,jerk_rate);
        end
```

```
%limit motor inputs
DM01_sat = 0;
DM09_sat = 0;
DM029_sat = 0;
if y(sum(cnts(1:5))+1) > rated_torque(1)
    DM01_sat = 1;
elseif y(sum(cnts(1:5))+1) < -rated_torque(1)
    DM01_sat = 1;
end
if y(sum(cnts(1:5))+2) > 20.09
    DM09_sat = 1;
elseif y(sum(cnts(1:5))+2) < -20.09
    DM09_sat = 1;
end
if y(sum(cnts(1:5))+3) > rated_torque(3)
    DM029_sat = 1;
elseif y(sum(cnts(1:5))+3) < -rated_torque(3)
    DM029_sat = 1;
end
if T>=tstart(stepcnt)
    I = DM01.calculateInertia(rad,0);
    s_gains = G_Unwind_speed.calculate(I,DM01);
    t_gains = G_Unwind_Ten.calculate(DM01.calculateInertia(rad,1)...
        ,G_Unwind_speed);
    %unwind tension loop gains
    Kp(1)= t_gains(1);
    Ki(1)= t_gains(2);
    %unwind speed loop gains
    Kp(2)=s_gains(1);
    Ki(2)=s_gains(2);
    RecordEvents(T,y,[Kp Ki zeros(1,15) 10])
end
%calc load cell tension R3 with added Gaussian noise
F(1) = [.5 .5]*y(rolls+[2 3],1);
addnoise = randn(1)*(.05155);
if F(1)+ addnoise > 0 && sim
    if (y(1)/Vmax)<=1 && y(1)>0
        F(1) = F(1) + addnoise*(y(1)/Vmax);
    end
end
F(2) = 0;
%calc loadcell tension R27
F(3) = [.5 .5]*y(rolls+[26 27],1);
% lead-lag compensate the tension feedbacks
[F(1), ldlg1] = TenComp1.compensate(y(sum(cnts(1:4))-2),F(1));
[F(3), ldlg3] = TenComp2.compensate(y(sum(cnts(1:4))-1),F(3));
 % find trim speeds
TrimV(1) = (Kp(1)*-(lineten-F(1))+Ki(1)*(y(sum(cnts(1:4))+1,1)...
    ))/60/40;
```

```
421         TrimV(2) = (Kp(4)*(lineten2-F(3))+Ki(4)*(y(sum(cnts(1:4))+2))...
                )/60/40;
423         ydot(sum(cnts(1:4))+1,1) = -(lineten-F(1))/40/60;
            ydot(sum(cnts(1:4))+2,1) = (lineten2-F(3))/40/60;
425         %limit trim velocity to 30 like HSWL
            TrimV(1) = LimitTrim(TrimV(1),30);
427         TrimV(2) = LimitTrim(TrimV(2),30);
            % calc motor current derivatives
429         if abs(y(9))<5/60 && lowspeed_start
                below5 = 1; % turn off the integral control while at low speed
431             RecordEvents( T,y,[zeros(1,25) 3]);
            else
433             lowspeed_start = 0;
                below5 = 0;
435         end
            % find errors in speed
437         errors = [ r1(1)-y(1)+TrimV(1), r2(1)-y(9), r3(1)-y(rolls)+...
                       TrimV(2); ...
439                    r1(2)-ydot(1), r2(2)-ydot(9), r3(2)-ydot(rolls)];
            %Unwind
441           Torq_ref = DM01.calculateInertia(rad,0)...
                *r1(2)/rad*radps2rpm + (Bfn(1)+Cmn(1))*r1(1)/...
443             rad*radps2rpm - rad/GR*...
                (lineten-F(1));%(y(cnts(1)+1)) ;
445         U(1) = (Kp(2)*(errors(1,1)*radps2rpm/rad)+...
                Ki(2)*(y(sum(cnts(1:5))+1,1)*radps2rpm/rad...
447             *(1-below5))+...
                Kd(2)*(r2(3)-(ydot(1)-y0(1))/.005))*radps2rpm/...
449             rad + Torq_ref;
            ydot(sum(cnts(1:5))+1,1) = errors(1,1);
451         U(1) = TorqueLimit(U(1),y(sum(cnts(1:4))-2)*rpm2radps,DM01);
            %Saturation logic
453         if DM01_sat
                if U(1) >= rated_torque(1)
455                 %positive limit
                    U(1) = rated_torque(1);
457             elseif U(1)<= -rated_torque(1)
                    %negative limit
459                 U(1) = -rated_torque(1);
                end
461         end
            %master speed
463         U(2) = Kp(3)*errors(1,2) + ...
                Ki(3)*y(sum(cnts(1:5))+2)*(1-below5)+...
465             Kd(3)*(r2(2)-y0(9,1));
            ydot(sum(cnts(1:5))+2,1) = errors(1,2);
467         %Saturation logic
            if DM09_sat
469             if U(2) >= 20.09
```

```
            %positive limit
471             U(2) = 20.09;
            elseif U(2) <= -20.09
473             %negative limit
                U(2) = -20.09;
475         end
        end
477     %Rewind
        Torq_ref = Jn(29)*r3(2) + (Bfn(29)+Cmn(29))*r3(1) - ...
479         Rn(29)/GR*(F(3)-lineten2);
        U(3) = Kp(5)*errors(1,3)+...
481         Ki(5)*(y(sum(cnts(1:5))+3)*(1-below5)) + Torq_ref;
        ydot(sum(cnts(1:5))+3,1) = errors(1,3);
483     U(3) = TorqueLimit(U(3),y(rolls)/Rn(rolls),DM29);
        %Saturation logic
485     if DM029_sat
            if U(3) >= rated_torque(3)
487         %positive limit
                U(3) = rated_torque(3);
489         elseif U(3) <= -rated_torque(3)
                %negative limit
491             U(3) = -rated_torque(3);
            end
493     end
        Tval = T;
495     yval = y;
        % get state derivative
497     ydot(1:sum(cnts(1:4)),1)= state_derive(T,y(1:sum(cnts(1:4)),1),U);
        if below5
499         ydot(sum(cnts(1:5))+1,1) = 0;
            ydot(sum(cnts(1:5))+3,1) = 0;
501     end
        %update the lead-lag compensator state derivative
503     ydot(sum(cnts(1:4))-2) = ldlg1;
        ydot(sum(cnts(1:4))-1) = ldlg3;
505     y0=ydot;
        % update waitbar
507     if T> hstep1
            waitbar( hstep1/tfinal/2, h)
509         hstep1 = hstep1 +.1*tfinal;
        end
511 end
%-------------------------------------------------------------------------
513 % S T A T E   F U N C T I O N
%-------------------------------------------------------------------------
515 function xdot = state_derive(T,x,u)
        v = x(1:rolls,1); %separate velocities
517     t = x(rolls+[1:spans],1); % separate tensions
        %--- care for changing radius of the unwind
```

```
519         if T>=tstart(stepcnt)
                rad = x(sum(cnts(1:4)));
521             G_Unwind_speed.set('R',rad);
                if stepcnt < stepcntmax
523                 stepcnt = stepcnt+1;
                end
525         end
            raddot = -web_thick/2/pi*x(sum(cnts(1:4))-3)*rpm2radps;
527         xdot = x*0;
            %All rollers ------------------------------------------------------
529         if max(size(friction_params))==2 %this is selected earlier
              cv = FrictionCalc(v,t,friction_params{1}(1:end),0,...
531               friction_params{2}(1:end),0);
            else
533           cv = FrictionCalc(v,t,friction_params{1}(1:end),0);
            end
535         xdot(1:rolls,1) = 1./Jn'.*cv;
            %xdot(20,1) = feval(funnames{1},t(19:20),v(20),20);
537         %Unwind -----------------------------------------------------------
            xdot(sum(cnts(1:4))-3,1) = (xdot(unwind,1)/Rn(1) + ...
539             1/(DM01.calculateInertia(rad,0))*(-Cmn(1)*...
                x(sum(cnts(1:4))-3,1)+OverWrap(1)*(u(1)) + rad*t0*0 ...
541             +Mn(1)*e_3*g*sin(x(sum(cnts(1:3))+1,1))))*30/pi;
            %------ Unwind rotation rate in RPM
543         xdot(unwind,1) = rad*xdot(sum(cnts(1:4))-3,1)*pi/30+ ...
                raddot*x(sum(cnts(1:4))-3,1)*pi/30;
545         %Master Speed -----------------------------------------------------
            if nip9
547             %use nip roller equation for roller 9
                xdot(9,1) = 1/(Rn(9)^2*Jn_n(9)+Rn_n(9)^2*Jn(9))*( ...
549                 -(Rn_n(9)^2*Bfn(9)+Rn(9)^2*Bfn_n(9))*v(9) + Rn_n(9)^2* ...
                    Rn(9)^2*(t(9) - t(8))+Rn_n(9)^2*Rn(9)*Kmn(9)*u(2));
551         else
                %no nip << Nip required by HSWL so this is not going to be used
553             %except in debugging
                xdot(9,1) = xdot(9,1) + 1/Jn(9)*(Kmn(9)*(u(2)*Rn(9)));
555         end
            %Rewind -----------------------------------------------------------
557         xdot(rolls,1) = xdot(rolls,1) + 1/Jn(rolls)*(OverWrap(29)*(u(3)...
                *Rn(rolls)) - 0*Rn(rolls)^2*tf); %-Rn(rolls)^2*tf*0
559         %Web Speed for tension derivs-------------------------------------
            %  Added 4-18-20 from Dwivedula "Modeling Web slip on Roller
561         %  Dynamics"  ASME 2005
            for i = 1: length(order)
563             p = order(i);
                v(p) = v(p) - t(p-1)/frictChar(1)*(1-(frictChar(2)/...
565                 frictChar(3))*(1-exp(-frictChar(3)*theta_wrap(p)))...
                    -t(p)/t(p-1));
567             % increasing the 1000 to 2000 makes more oscillatory response
```

```
        end
        %Tension derivs ---------------------------------------------------
        tens = [t0; t]; %include wound in tension
        for i = 1:spans
            if i == dancer
                temp(3) = x(sum(cnts(1:2))+1);
                temp(4) = x(sum(cnts(1:2))+2);
                sp0 = [v(i+[0 1])', tens(i+[0 1],1)', L(i)+...
                    temp(3), ... %Length does not change
                        temp(4), theta_on(i)]';
            elseif i+1==dancer
                temp(1) = x(sum(cnts(1:2))+1);
                temp(2) = x(sum(cnts(1:2))+2);
                sp0 = [v(i+[0 1])', tens(i+[0 1],1)', L(i)+...
                    temp(1),... %Length does not change
                    temp(2), theta_in(i+1)]';
            else
                sp0 = [v(i+[0 1])' tens(i+[0 1],1)' L(i) 0 theta_on(i)]';
            end
            xdot(rolls+i,1) = span2(T, sp0, E, A);
        end
        %Update the eccentric roller's angular speed -----------------------
        xdot(sum(cnts(1:3))+1,1) = x(eccent,1)/Rn(eccent);
        %Update derivative of radius
        xdot(sum(cnts(1:4)),1)  = raddot;
    end
%-------------------------------------------------------------------------
% S T E A D Y - S T A T E   F U N C T I O N
%-------------------------------------------------------------------------
    function z = steadystate(x)
        stepcnt = 1;
        c = Controller(0,[...
            Initial_Vel(1)
            x(2:8,1);...
            Initial_Vel(9); ...
            x(10:rolls-1,1);...
            Initial_Vel(driven(2));...
            x(rolls+[1],1);...
            (2*lineten-x(rolls+3));... %span prior to load cell 1
            x(rolls+[3:25],1);...
            (2*lineten2-x(rolls+27,1));... %span prior to load cell 2
            x(rolls+[27:28],1);...
            x(eccent,1)/Rn(eccent);... %theta(1)
            x(unwind,1)/Rn(unwind)*30/pi;... %omega(1) in rpm
            lineten;...%0.5-(x(sum(cnts(1:2))+1,1)*12/11);... %filter state
            lineten2;...  %filter state 2
            Rn(1);... %radius of Unwind roll
            ...%
            x(sum(cnts(1:4))+1,1);...
```

```
617             0;...
                x(sum(cnts(1:5))+[1:cnts(6)])];
619         z = c;
            z([1,9]) = Initial_Vel(1,9)*0;
621         z(sum(cnts(1:4))-2) = x(sum(cnts(1:4))-2) - z(eccent)/Rn(eccent);
            z(sum(cnts(1:4))) = z(sum(cnts(1:4))) - (-web_thick/2/pi*...
623             x(sum(cnts(1:4))-3)*rpm2radps);
    end
625 %-------------------------------------------------------------------------
    %      B E A R I N G   F R I C T I O N
627 %-------------------------------------------------------------------------
    function bv = FrictionCalc(v,t,op1,val1,op2,val2)
629     % calculate bearing friction and parasitic torque if applicable
        if nargin == 6
631         switch op2
                case 'sliding_friction'
633                 val2 = Bsldfn'.*Rn' ;
                case 'Parasitic_Torque'
635                 val2 = Rn'.*ParasiticTorque';
                case 'viscous_friction'
637                 val2 = Bfn'.*v;
            end
639         switch op1
                case 'sliding_friction'
641                 val1 = Bsldfn'.*Rn' ;
                case 'Parasitic_Torque'
643                 val1 = Rn'.*ParasiticTorque';
                case 'viscous_friction'
645                 val1 = Bfn'.*v;
            end
647     elseif nargin > 2
            val2 = v*0;
649         switch op1
                case 'sliding_friction'
651                 val1 = Bsldfn'.*Rn' ;
                case 'Parasitic_Torque'
653                 val1 = Rn'.*ParasiticTorque';
                case 'viscous_friction'
655                 val1 = Bfn'.*v;
            end
657     elseif nargin < 3
            val1 = v*0;
659         val2 = val1;
        end
661     tens = ([t; tf]-[t0; t]);
        tens(unwind) = tens(unwind)+t0;
663     tens(rewind) = tens(rewind)-tf;
        test1 = -(val1 + val2) + Rn'.^2.*(tens);
665     %adding in the starting torque for DM01, 03, 09
```

```
        test1(driven(1)) = test1(1)-4.106*Rn(1)^2;
667         %4.106ft*lbf starting torque measured
        test1(driven(2)) = test1(9)-0.846*Rn(9)^2;
669         %0.846ft*lbf starting torque measured
        test1(driven(3)) = test1(29)-2.989*Rn(29)^2;
671         %2.989ft*lbf starting torque measured
        start_torque = [4.106*Rn(1)^2 0.846*Rn(9)^2 2.989*Rn(29)^2];
673     run_torque([driven]) = [2.774*Rn(1)^2 0.763*Rn(9)^2  ...
            2.4169*Rn(29)^2];
675        %running torques measured
       m = size(idlers',1);
677    bv = v*0; %zeros(m,1);
       for i= 1:m
679         j = idlers(i);
            if 0% abs(v(j))-5/60<=0 && test1(j)<= 0
681             %bv(i) = 0; becaue the bearing drag is larger than the
                %input torque
683             RecordEvents(Tval,yval,[zeros(1,25),6])
            else
685             bv(j) = -(val1(j) + val2(j)) + Rn(j)^2*(tens(j));
            end
687     end
       m = size(driven,1);
689    for i= 1:m
           j = driven(i);
691        %for driven elements sum friction and starting torque
           if abs(v(j)) - 5/60 <= 0 && ~started
693            bv(j) = -(val1(j) + val2(j) +...
                   (start_torque(i) -  (start_torque(i)-run_torque(j))*...
695                (1-exp(-v(j)*60)))  -...
                   Rn(j)^2*(tens(j)); %test1(j);
697            RecordEvents(Tval,yval,[zeros(1,25),7])
           else
699            started = 1;
               bv(j) = -(val1(j) + val2(j) + run_torque(j)) +...
701                Rn(j)^2*(tens(j));
           end
703    end
   end
705 function tor = TorqueLimit(U,v,mot)
       % v: rad/s, L: ft*lbf, U: ft*lbf, mot: motor object
707    % V: RPM,
       V = v/2/pi*60;
709    test1 = U*v/550;

711    if V < 1420
           volt = (V-0)/1420*(460-18)+18;
713    else
           volt = 460;
```

```
715         end

717         if abs(V) > 1150
                Li = mot.hp/v;
719         else
                Li = mot.torque_rated;
721         end
            amp=U*v/1.356/volt;
723         if amp*volt/745.69 > mot.hp
                disp('Power Limit')
725         end

727         if test1 > mot.hp
                tor = Li;
729         end
            if U > Li
731             tor = Li;
            elseif U < -Li
733             tor = -Li;
            else
735             tor = U;
            end
737     end
        function limV = LimitTrim(trim,Lim)
            if trim > Lim
                limV = Lim;
741             RecordEvents(Tval,yval,[zeros(1,25) 8]);
            elseif trim < -Lim
743             limV = -Lim;
                RecordEvents(Tval,yval,[zeros(1,25) -8]);
745         else
                limV = trim;
747         end
        end
749 %-------------------------------------------------------------------------
    % P L O T T I N G
751 %-------------------------------------------------------------------------
    % Unit conversions
753     function l = legendmaker(list,prefix,postfix)
            m = numel(list);
755         l = cell(m,1);
            if nargin<3
757             postfix = [];
            end
759         for i = 1:m
                l(i) = {[prefix num2str(list(i)) postfix]};
761         end
        end
763 velocities_out = z(:,1:rolls)*60; %fpm
```

```
tensions_out = z(:,rolls+1:rolls+spans);
LC_out = z(:,rolls+[2 3 26 27])*[.5 .5 0 0; 0 0 .5 .5]';
motor_inputs_out = z(:,sum(cnts(1:4))+[1:sum(cnts(5:6))]);

% Rollers and Spans of interest
velindex = [1  9 27 rolls];
tenindex = [1 2 3 26 27 28];
waitbar( .6, h)

r1 = IndScurveInput(0,z0(9),deltaV(1),accel_rate,jerk_rate,1);
refspeed = zeros(length(time),1);
shutdown_flag = 0;
for i= 1:length(time)
    t=time(i);
        r1 = IndScurveInput(t,z0(1),deltaV(1),accel_rate,jerk_rate);
    refspeed(i) = r1(1);
end

ha=figure(2);
subplot(2,2,1)
plot(time,velocities_out(:,velindex)-repmat(refspeed,1,length(velindex))*60);
ylabel('Velocity \Delta');
legs = legendmaker(velindex,'R ');
legend(legs)
s = get(ha,'children'); t = get(s(1),'title');
set(t,'string',fourPlotTitle)
set(t,'fontsize',16)
set(t,'position',[1 1.0763 1.0021])
grid;

subplot(2,2,2)
plot(time,tensions_out(:,tenindex))
ylabel('tension (lbf)');
legs = legendmaker(tenindex,'t ');
legend(legs);
grid;

subplot(2,2,3)
plot(time,LC_out);
title('Feedback');
legend('R3 (lbf)','R27 (lbf)','location','best')
grid;

subplot(2,2,4)
plot(time,motor_inputs_out);
title('Errors');
legend('Unw Ten','Rew Ten','Unwind','M. Speed','Rewind','location','best')
ylabel('Amps')
grid;
```

```
813 waitbar( 1, h)
    clf(figure(6))
815 figure(6)
    s = {'Unwind Speed Gains'; ...
817     ['K_p:   ' num2str(G_Unwind_speed.K_p)];...
        ['K_i:   ' num2str(G_Unwind_speed.K_i)];...
819     ['Bandwidth:   ' num2str(G_Unwind_speed.BW)];...
        };
821 annotation(6,'textbox',[0,0.2,.3,.3],'string',s)
    s = {'Unwind Tension Gains'; ...
823     ['K_p:   ' num2str(G_Unwind_Ten.K_p)];...
        ['K_i:   ' num2str(G_Unwind_Ten.K_i)];...
825     ['Bandwidth:   ' num2str(G_Unwind_Ten.BW)];...
        };
827 annotation(6,'textbox',[.3,.2,.3,.3],'string',s)
    SSder = Controller(0,SS);
829 % Finishing Up
    close(h)
831 clear RecordEvents R01 R29 DM01 DM09 DM29 TenComp1 G_Unwind_speed
    clear G_Unwind_Ten G_Rewind_speed G_Rewind_Ten
833 end
```

# APPENDIX K

# RSLOGIX CODE

This chapter contains the structured text code added to the RSLogix 5000® software running the HSWL to implement the Routh Approximation Method gain calculations. The code is split into three parts: the main code and two subroutines used to calculate coefficients for the Crammer's Rule solving in the main code. There were a few other additions to the RSLogix 5000® program, `OSU_HSRL_060520.ACD`, that were not in structured text format and do not translate well to this format. That code handled the switching between the Rockwell method and the RA method and the change of lead-lag filter frequencies. The code decided which method to use via a boolean tag, '`BR_UseRAMethod`', that the user can set to true or false when the line is stopped.

## K.1 Gain Calculation Method

```
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
(* Replaced by Ben Reish 3-6-2020                                  *)
(* Revision 0.1                                                    *)
(* Revision 0.2 4-2-2020                                           *)
(* Revision 0.3 5-7-2020                                           *)
(* Revision 0.4 5-19-2020                                           *)
(*                                                                 *)
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)

(* Look up the following values in the Controller Tags *)
(*    and verify that they are what you want them to be. *)
(* zBR_RiseTimeSpd_Sec *)
(*    zBR_DampingRatioSpd *)
(*    zBR_RiseTimeTen_Sec *)
(*    zBR_DampingRatioTen *)
(*    zBR_TenLoopSample_ratio = 40 *)
(*    zBR_Final_Value_adj = 10 *)
```

```
(*    zBR_RAMethodOutputNum (3 cell array) [ 0 91.9134 1512] *)
(*    zBR_RAMethodOutputDen (3 cell array)      [ 1 17.2253 12.9854] *)
(*    The example RAMethodOutput Num and Den values are  for *)
(*    a tension loop natural frequency of 20.57 rad/s and  *)
(*    a tension loop damping ratio of 0.9 *)

(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
(*                                                         *)
(* Calculate                                               *)
(* (Rated Motor Torque)                                    *)
(* PID F03P05R116                                          *)
(* Revision 1.01              Unchanged                    *)
(*                                                         *)
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)

(*                       MtrPwrRated [HP] * 5252     *)
(* MtrTrqRated_LbFt = ---------------------------    *)
(*                           MtrSpdBase [RPM]        *)
DM01_MtrTrqRated_LbFt := zDM01_MtrPwrRated_HP * 5252 / zDM01_MtrSpdBase_RPM ;


(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
(*                                                         *)
(* Calculate                                               *)
(* (Inertia, Center Winder)                                *)
(* PID F03P05R110                                          *)
(* Revision 1.01              Unchanged                    *)
(*                                                         *)
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)

(* RollVolume_Ft3 = (Pi * (Diam_In/24)**2 - Pi * (DiamMinEC_In/24)**2) *
      Wid_In/12 *)
(*                = Pi * ((Diam_In/24)**2 - (DiamMinEC_In/24)**2) * *)
(* Width_In/12    *)
DM01_RollVolume_Ft3 := ((DM01_RollDiam_In/24)**2 - (zDM01_DiamMinEC_In/
    24)**2) * Web_Width_SP / 3.8197 ;
(*3.8197 = 1/(pi/12) *)

(* RollWeight_Lb = RollVolume_Ft3 * Density_lbft3 *)
DM01_RollWeight_Lb := DM01_RollVolume_Ft3 * Material_Density_SP * 1728 ;


(* JRoll_LbFt2 = RollWeight_lb / 2 * ((Diam_In/24)**2 + (DiamMinEC_In/ *)
(* 24)**2) / GearRatio**2 *)
DM01_JRoll_LbFt2 := DM01_RollWeight_Lb / 2 * ((DM01_RollDiam_In/24)**2 +
      (zDM01_DiamMinEC_In/24)**2) / zDM01_GearRatio**2 ;


(* Calculate total reflected inertia [pound-feet**2] *)
```

```
67  DM01_J_LbFt2 := zDM01_JEC_LbFt2 + DM01_JRoll_LbFt2 ;

69
    (*            J [LbFt**2] * MtrSpdBase [RPM]      *)
71  (* J [sec] = ------------------------------------   *)
    (*                308 * MtrTrqRated [FtLb]         *)
73  DM01_J_Sec := DM01_J_LbFt2 * zDM01_MtrSpdBase_RPM / (308 *
        DM01_MtrTrqRated_LbFt) ;
75

77  (* Calculate normalized inertia for monitoring only (1.0 = empty core) *)
    DM01_J_PU := DM01_J_LbFt2 / zDM01_JEC_LbFt2 ;
79

81  (* * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
    (*                                                       *)
83  (* Calculate                                             *)
    (* (Torque per Tension)                                  *)
85  (* PID F03P05R111                                        *)
    (* Revision 1.01              Unchanged                  *)
87  (*                                                       *)
    (* * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
89

    (*                          Diam [In]          100                  1       *)
91  (* Conversion_PctPerLb = ------------ * ------------------- * ------------ *)
    (*                           12 * 2      MtrTrqRated [LbFt]     GearRatio  *)
93  DM01_Conversion_PctPerLb := DM01_RollDiam_In / zDM01_GearRatio /
        DM01_MtrTrqRated_LbFt / 0.24 ;
95

97  (* * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
    (*                                                       *)
99  (* Calculate                                             *)
    (* (Torque per Line Speed Rate)                          *)
101 (* PID F03P05R112                                        *)
    (* Revision 1.01              Unchanged                  *)
103 (*                                                       *)
    (* * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
105

    (*                                J [sec] * GearRatio * 100          *)
107 (* Conversion_PctPerFPMsec = ------------------------------------------ *)
    (*                           Pi * (Diam [in] / 12) * MtrSpdBase [RPM]  *)
109 DM01_Conversion_PctPerFPMSec := DM01_J_Sec * zDM01_GearRatio /
        DM01_RollDiam_In / zDM01_MtrSpdBase_RPM * 381.97 ;
111

113 (* * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
    (*                                                       *)
115 (* Calculate                                             *)
```

```
(* (Speed Regulator Tuning, 700S via DPI)                    *)
(* Replaced by Ben Reish 3-6-2020                            *)
(* Revision 0.1                                              *)
(* Revision 0.2  4-2-2020                                    *)
(*                                                           *)
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)

(* Look up the following values in the Controller Tags *)
(*    and verify that they are what you want them to be. *)
(* zBR_RiseTimeSpd_Sec *)
(*    zBR_DampingRatioSpd *)
(*    zBR_RiseTimeTen_Sec *)
(*    zBR_DampingRatioTen *)
(*    BR_DM01_J_slugFt2 *)
(*    zBR_RAMethodOutputNum (3 cell array) *)
(*    zBR_RAMethodOutputDen (3 cell array)       *)
(*    BR_Kdn_LbfFt - dancer spring constant, if any  *)
(*    BR_DanMass_Slug - dancer mass                  *)

(* Calculate motor total inertia in slug-ft^2 *)
BR_DM01_J_slugFt2 := DM01_J_LbFt2 / 32.176;

(* Calculate motor constant torque/amp *)
DM01_Km_FtLbfAmp := DM01_MtrTrqRated_LbFt / zDM01_MtrCurrentRated_Amps;

(* Trap Damping ratio between 0.1 and 0.9 for accuracy of estimation *)
IF (zBR_DampingRatioSpd < 0.1 ) THEN
        BR_DampingRatioSpd := 0.1;
ELSIF (zBR_DampingRatioSpd > 0.9 ) THEN
        BR_DampingRatioSpd := 0.9;
ELSE
        BR_DampingRatioSpd := zBR_DampingRatioSpd;
END_IF;

(* Calculate the Speed loop natural frequency from the rise time and *)
(* Damping ratio *)
BR_NaturalFreqSpd_RadSec := (zSy_Pi - ATAN(SQRT(1 - BR_DampingRatioSpd**2) /
        BR_DampingRatioSpd)) / (zBR_RiseTimeSpd_Sec * SQRT(1 -
        BR_DampingRatioSpd**2));

(* Calculate iniitegral speed gain *)
DM01_SpdRegKi := DM01_J_LbFt2 / 32.176 * BR_NaturalFreqSpd_RadSec**2;

(* Calculate proportional speed gain *)
DM01_SpdRegKp := 2 * BR_DampingRatioSpd * BR_NaturalFreqSpd_RadSec *
        DM01_J_LbFt2 / 32.176 ;

(* Limit speed regulator proportional gain *)
IF ( DM01_SpdRegKp > zDM01_SpdRegKpMax ) THEN
```

```
    DM01_SpdRegKp := zDM01_SpdRegKpMax ;
END_IF ;

(* Calculate actual speed regulator open-loop crossover frequency *)
DM01_SpdRegWco_Rad := DM01_SpdRegKi / DM01_SpdRegKp;

(* Calculate speed regulator lead frequency *)
(* Drop the integral off while in the Tension with Speed mode*)
IF (DM01_Mode = 2  AND DM01_Tension_On) Then
DM01_SpdRegWld_Rad := zDM01_SpdRegWld_Rad ;
ELSE
DM01_SpdRegWld_Rad := DM01_SpdRegWco_Rad / 4 ;
End_IF;


(* Copy File instruction for DPI format *)
COP(DM01_SpdRegKp,DM01_DrvSpdRegKp_DC,1) ;
COP(DM01_SpdRegKi,DM01_DrvSpdRegKi_DC,1) ;

(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
(*                                                           *)
(* Calculate                                                 *)
(* (Dancer Position Regulator Tuning, Speed Control)         *)
(* Replaced by Ben Reish 4-2-2020                            *)
(* Revision .01                                              *)
(*                                                           *)
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)

(* Convert master line speed to ft/s *)
BR_LS := Master_Ref_Input / 60;
BR_EA := Web_Width_SP * Web_Thickness_SP / 1000 * Material_Modulus_SP;

(* Set up the motor speed loop TF*)
BR_num[0] := 0;
BR_num[1] := DM01_SpdRegKp / BR_DM01_J_slugFt2;
BR_num[2] := DM01_SpdRegKi / BR_DM01_J_slugFt2;
BR_den[0] := 1;
BR_den[1] := DM01_SpdRegKp / BR_DM01_J_slugFt2;
BR_den[2] := DM01_SpdRegKi / BR_DM01_J_slugFt2;

(* Calculate transfer function to dancer position *)
(* Put coefficients in BR_PolyIn_num[2] and BR_PolyIn_den[6]  *)
BR_PolyIn_num[0] := 0;
BR_PolyIn_num[1] := DM01_RollDiam_In/24 / zBR_webLength_Ft[1] * BR_num[1] *
        BR_EA / BR_DanMass_Slug;
BR_PolyIn_num[2] := DM01_RollDiam_In/24 / zBR_webLength_Ft[1] * BR_num[2] *
        BR_EA / BR_DanMass_Slug;

BR_PolyIn_den[0] := 1;
```

```
BR_PolyIn_den[1] := BR_den[1] + BR_LS / zBR_webLength_Ft[1] + BR_Cdn_LbfS /
        BR_DanMass_Slug;
BR_PolyIn_den[2] := BR_den[1] * (BR_LS / zBR_webLength_Ft[1] + BR_Cdn_LbfS /
        BR_DanMass_Slug ) + BR_den[2] + BR_LS / zBR_webLength_Ft[1] *
        BR_Cdn_LbfS / BR_DanMass_Slug + BR_Kdn_LbfFt / BR_DanMass_Slug;
BR_PolyIn_den[3] := BR_den[2] * (BR_LS / zBR_webLength_Ft[1] + BR_Cdn_LbfS /
        BR_DanMass_Slug) + BR_Kdn_LbfFt / BR_DanMass_Slug * BR_LS /
        zBR_webLength_Ft[1] + BR_den[1] * (BR_LS / zBR_webLength_Ft[1] *
        BR_Cdn_LbfS / BR_DanMass_Slug + BR_Kdn_LbfFt / BR_DanMass_Slug);
BR_PolyIn_den[4] := BR_den[2] * (BR_LS / zBR_webLength_Ft[1] * BR_Cdn_LbfS
        / BR_DanMass_Slug + BR_Kdn_LbfFt / BR_DanMass_Slug) + BR_den[1] *
        BR_Kdn_LbfFt / BR_DanMass_Slug * BR_LS / zBR_webLength_Ft[1] ;
BR_PolyIn_den[5] := BR_den[2] * BR_LS / zBR_webLength_Ft[1] * BR_Kdn_LbfFt /
        BR_DanMass_Slug;

(* Call Subroutine to calculate the Routh Approximation and fill in the *)
(* BR_TenKPCalcAry array *)
JSR(DM01_GainCalcRADan);

(* Execute Crammer's Rule to solve set of 2 equations for 2 unknowns *)
BR_krTen := (BR_TenKPCalcAry_1[2] * BR_TenKPCalcAry_2[1] -
        BR_TenKPCalcAry_1[1] * BR_TenKPCalcAry_2[2]) / (BR_TenKPCalcAry_1[0]
        * BR_TenKPCalcAry_2[1] - BR_TenKPCalcAry_1[1] *
        BR_TenKPCalcAry_2[0]);
BR_KpTen := (BR_TenKPCalcAry_1[0] * BR_TenKPCalcAry_2[2] -
        BR_TenKPCalcAry_1[2] * BR_TenKPCalcAry_2[0]) /
        (BR_TenKPCalcAry_1[0] * BR_TenKPCalcAry_2[1] - BR_TenKPCalcAry_1[1]
        * BR_TenKPCalcAry_2[0]);

(* Calculate integral gain on the tension regulator *)
BR_KiTen := (BR_krTen * BR_NaturalFreqTen_RadSec**3 * BR_DampingRatioTen) /
        BR_RAMethodOutputNum[2];

(* Ensure positiveness of gains (the sign will be taken care of by whether *)
(* or not the trim speed is additive or subtractive from the reference *)
(* speed *)
DM01_TenRegKp_Spd := abs(BR_KpTen);
DM01_TenRegKi_Spd := abs(BR_KiTen);

(* Calculate dancer storage [seconds] *)
DM01_PMax_Sec := zDM01_PMax_In * 5 / zSy_LineSpdMax_FPM ;


(* Calculate dancer position regulator lead frequency *)
DM01_DanRegWld_Rad_Spd := zDM01_DanRegWcoTrgt_Rad / 5 ;

(* Calculate dancer position regulator feedback lead-lag frequency *)
DM01_DanRegFbWld_Rad := DM01_SpdRegWco_Rad;
DM01_DanRegFbWlg_Rad := DM01_DanRegFbWld_Rad * 5 ;
```

```
263
    (* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
265 (*                                                                 *)
    (* Calculate                                                      *)
267 (* (Tension Regulator Tuning, Speed Control)                      *)
    (* Replaced by Ben Reish 3-6-2020                                 *)
269 (* Revision 0.1                                                   *)
    (*                                                                *)
271 (* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)

273 (* Trap Damping ratio between 0.1 and 0.9 for accuracy of estimation *)
    IF (zBR_DampingRatioTen < 0.1 ) THEN
275         BR_DampingRatioTen := 0.1;
    ELSIF (zBR_DampingRatioTen > 0.9 ) THEN
277         BR_DampingRatioTen := 0.9;
    ELSE
279         BR_DampingRatioTen := zBR_DampingRatioTen;
    END_IF;
281
    (* Calculate the Speed loop natural frequency from the rise time and  *)
283 (* Damping ratio *)
    BR_NaturalFreqTen_RadSec := (zSy_Pi - ATAN(SQRT(1 - BR_DampingRatioTen**2) /
285     BR_DampingRatioTen)) / (zBR_RiseTimeTen_Sec * SQRT(1 -
        BR_DampingRatioTen**2));
287


289 (* Test the given Numerator last position for zero value, if so replace *)
    (* with 1 *)
291 IF (zBR_RAMethodOutputNum[2] = 0) THEN
        zBR_RAMethodOutputNum[2] := 1;
293 END_IF;

295 (* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
    (*                                                                *)
297 (* Calculate                                                      *)
    (* (Routh Approximation Routine)                                  *)
299 (* by Ben Reish 3-13-2020                                         *)
    (* Revision 0.1                                                   *)
301 (* Revision 0.2 4-2-2020                                          *)
    (* Revision 0.4 5-19-2020                                          *)
303 (*                                                                *)
    (* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
305
    (* Tags to be added to the system:
307 BR_RAMath_Array[10]
    BR_PolyIn_num[3]
309 BR_PolyIn_den[4]
    BR_an[]
311 BR_bn[]
```

```
BR_beta[2]
BR_alpha[2]
BR_num[3]
BR_den[4]
zBR_webLength_Ft[2] - [distance to load cell][distance to dancer]
BR_LS
BR_EA
BR_RAMethodOutputNum[3]
BR_RAMethodOutputDen[3]
*)
(* 60/2/pi =  9.5493 *)

(* Calculate transfer function from ref speed to speed output *)
(* Put coefficients in BR_num[3] and BR_den[3]  *)
BR_num[0] := 0;
BR_num[1] := DM01_SpdRegKp / BR_DM01_J_slugFt2;
BR_num[2] := DM01_SpdRegKi / BR_DM01_J_slugFt2;
BR_den[0] := 1;
BR_den[1] := DM01_SpdRegKp / BR_DM01_J_slugFt2;
BR_den[2] := DM01_SpdRegKi / BR_DM01_J_slugFt2;


(* Calculate transfer function to tension of span at load cell *)
(* Put coefficients in BR_PolyIn_num[3] and BR_PolyIn_den[4]  *)
BR_PolyIn_num[0] := 0;
BR_PolyIn_num[1] := DM01_RollDiam_In/24 / zBR_webLength_Ft[0] * BR_num[1] *
        BR_EA;
BR_PolyIn_num[2] := DM01_RollDiam_In/24 / zBR_webLength_Ft[0] * BR_num[2] *
        BR_EA;
BR_PolyIn_den[0] := 1;
BR_PolyIn_den[1] := BR_den[1] + BR_LS / zBR_webLength_Ft[0];
BR_PolyIn_den[2] := BR_den[1] * BR_LS / zBR_webLength_Ft[0] + BR_den[2];
BR_PolyIn_den[3] := BR_den[2] * BR_LS / zBR_webLength_Ft[0];

(* Call Subroutine to calculate the Routh Approximation and fill in the *)
(* BR_TenKPCalcAry array *)
JSR(GainCalcRALC)

(* Execute Crammer's Rule to solve set of 2 equations for 2 unknowns *)
BR_krTen := (BR_TenKPCalcAry_1[2] * BR_TenKPCalcAry_2[1] -
        BR_TenKPCalcAry_1[1] * BR_TenKPCalcAry_2[2]) / (BR_TenKPCalcAry_1[0]
        * BR_TenKPCalcAry_2[1] - BR_TenKPCalcAry_1[1] *
        BR_TenKPCalcAry_2[0]);
BR_KpTen := (BR_TenKPCalcAry_1[0] * BR_TenKPCalcAry_2[2] -
        BR_TenKPCalcAry_1[2] * BR_TenKPCalcAry_2[0]) / (BR_TenKPCalcAry_1[0]
        * BR_TenKPCalcAry_2[1] - BR_TenKPCalcAry_1[1] *
        BR_TenKPCalcAry_2[0]);

(* Calculate integral gain on the tension regulator *)
```

```
BR_KiTen := (BR_krTen * BR_NaturalFreqTen_RadSec**3 * BR_DampingRatioTen) /
        BR_RAMethodOutputNum[2];

(* Ensure positiveness of gains (the sign will be taken care of by whether *)
(* or not the trim speed is additive or subtractive from the reference     *)
(* speed*)
DM01_TenRegKp_Spd := abs(BR_KpTen);
DM01_TenRegKi_Spd := abs(BR_KiTen);

(* Scale Kpt and Kit by build up ratio squared and multiply the adjustment *)
(* factor to Ki instead of Kp *)
DM01_TenRegKp_Spd := DM01_TenRegKp_Spd * (DM01_RollDiam_In /
        zDM01_DiamMinEC_In)**2 * zBR_TenLoopSample_ratio;
DM01_TenRegKi_Spd := DM01_TenRegKi_Spd * (DM01_RollDiam_In /
        zDM01_DiamMinEC_In)**2 * zBR_TenLoopSample_ratio *
        zBR_Final_Value_adj;

(* Calculate the tension regulator lead frequency from gains *)
DM01_TenRegWld_Rad_Spd := DM01_TenRegKi_Spd / DM01_TenRegKp_Spd;
```

## K.2 Routh Approximation Method Implementation in RSLogix

### K.2.1 Load Cell Feedback

This code is used to setup the simultaneous equations to be solved for the case where a load cell feedback is used.

```
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
(* Routh Approximation method for Load Cell                       *)
(* Replaced by Ben Reish 4-2-2020                                 *)
(* Revision 0.1                                                   *)
(*                                                                *)
(*                                                                *)
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)


(* Calculate required parts for Routh Approximation (Very         *)
(* streamlined, only works for this situation, do not use in general. *)
BR_alpha[0] := BR_PolyIn_den[3] / BR_PolyIn_den[2];
BR_alpha[1] := BR_PolyIn_den[2] / (BR_PolyIn_den[1] - BR_alpha[0] *
        BR_PolyIn_den[0]);

BR_beta[0] := BR_PolyIn_num[2] / BR_PolyIn_den[2];
BR_beta[1] := BR_PolyIn_num[1] / (BR_PolyIn_den[1] - BR_alpha[0] *
        BR_PolyIn_den[0]);

BR_RAMethodOutputNum[0] := 0;
```

```
BR_RAMethodOutputNum[1] := BR_beta[1];
BR_RAMethodOutputNum[2] := BR_alpha[1] * BR_beta[0];
BR_RAMethodOutputDen[0] := 1;
BR_RAMethodOutputDen[1] := BR_alpha[1];
BR_RAMethodOutputDen[2] := BR_alpha[1] * BR_alpha[0];

(* Fill in array of values for computing kr and Kpt using Crammer's Rule *)
BR_TenKPCalcAry_1[0] := BR_NaturalFreqTen_RadSec * BR_DampingRatioTen;
BR_TenKPCalcAry_1[1] := -BR_RAMethodOutputNum[1];
BR_TenKPCalcAry_1[2] := (-2 * BR_DampingRatioTen * BR_NaturalFreqTen_RadSec +
        BR_RAMethodOutputDen[1]);

(* Fill in second array of values for computing kr and Kpt *)
BR_TenKPCalcAry_2[0] := (2 * BR_DampingRatioTen**2 *
        BR_NaturalFreqTen_RadSec**3 - BR_RAMethodOutputNum[1] /
        BR_RAMethodOutputNum[2] * BR_DampingRatioTen *
        BR_NaturalFreqTen_RadSec**3);
BR_TenKPCalcAry_2[1] := -BR_RAMethodOutputNum[2];
BR_TenKPCalcAry_2[2] := -BR_NaturalFreqTen_RadSec + BR_RAMethodOutputDen[2];
```

### K.2.2 Dancer Feedback

This RA Method is specifically for initializing the simultaneous equations to be solved for the case where a motor uses dancer feedback.

```
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
(* Routh Approximation method for Dancer                        *)
(* Replaced by Ben Reish 4-2-2020                               *)
(* Revision 0.1                                                 *)
(*                                                              *)
(*                                                              *)
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)

(* Add tags *)
(* zBR_Unw_dan_Kn := 0 ; *)
(* zBR_webLength_Ft[0] - add info for rewind in [2] and [3] *)
(* BR_PolyIn_den[6] - add length to array for Dancer *)
(* BR_NaturalFreqDan_RadSec *)
(* BR_DampingRatioDan *)


(* Calculate required parts for Routh Approximation (Very       *)
(* streamlined, only works for this situation, do not use in general. *)
BR_alpha[0] := BR_PolyIn_den[5] / BR_PolyIn_den[4];
BR_alpha[1] := BR_PolyIn_den[4] / (BR_PolyIn_den[3] - BR_alpha[0] *
        BR_PolyIn_den[2]);
```

```
BR_beta[0] := BR_PolyIn_num[2] / BR_PolyIn_den[4];
BR_beta[1] := BR_PolyIn_num[1] / (BR_PolyIn_den[3] - BR_alpha[0] *
        BR_PolyIn_den[2]);

BR_RAMethodOutputNum[0] := 0;
BR_RAMethodOutputNum[1] := BR_beta[1];
BR_RAMethodOutputNum[2] := BR_alpha[1] * BR_beta[0];
BR_RAMethodOutputDen[0] := 1;
BR_RAMethodOutputDen[1] := BR_alpha[1];
BR_RAMethodOutputDen[2] := BR_alpha[1] * BR_alpha[0];

(* Fill in array of values for computing kr and Kpt using Crammer's Rule *)
BR_TenKPCalcAry_1[0] := BR_NaturalFreqTen_RadSec * BR_DampingRatioTen;
BR_TenKPCalcAry_1[1] := -BR_RAMethodOutputNum[1];
BR_TenKPCalcAry_1[2] := (-2 * BR_DampingRatioTen * BR_NaturalFreqTen_RadSec +
        BR_RAMethodOutputDen[1]);

(* Fill in second array of values for computing kr and Kpt *)
BR_TenKPCalcAry_2[0] := (2 * BR_DampingRatioTen**2 *
        BR_NaturalFreqTen_RadSec**3 - BR_RAMethodOutputNum[1] /
        BR_RAMethodOutputNum[2] * BR_DampingRatioTen *
        BR_NaturalFreqTen_RadSec**3);
BR_TenKPCalcAry_2[1] := -BR_RAMethodOutputNum[2];
BR_TenKPCalcAry_2[2] := -BR_NaturalFreqTen_RadSec + BR_RAMethodOutputDen[2];
```

# GLOSSARY

| Notation | Description |
|---|---|
| $\Delta$Speed | The difference of the the reference speed and the data sample. 76–78, 80, 87–92, 94, 114, 119 |
| 1-Per-Rev | (Once-per-revolution) An event that happens one time during one revolution of an object; for example the event of an object rotating through one revolution can be a 1-per-rev because of small eccentricities in the object. 80, 149–151, 153–156 |
| Accumulator | A multi-roller mechanism of $2N + 1$ rollers where $N + 1$ rollers are stationary and $N$ rollers move together (the festoon) to store web for use in a zero-speed-splice operation. 7, 9, 19, 20, 24 |
| Basis Weight | The mass of $3000\ ft^2$ of a material. Or, more generally, a weight per quantity surface area. 141 |
| Dancer | A roller attached to a movable support usually configured to travel linearly, a translational dancer, or in an arc, a pendulum dancer, in order to attenuate tension variations. 1–3, 5, 8, 18, 32 |
| DG | Ducotey-Good. 7, 86, 87, 102, 111, 112, 114, 117–119, 122 |

| Notation | Description |
|---|---|
| Downstream | An ordinal term describing something being in the same direction as web travel from the unwinding roll to the rewinding roll. 2, 3, 17, 18, 57, 74, 100, 101 |
| Driven Roller | A driven roller is a roller that is powered by a motor.. 1, 2, 4 |
| EWL | Euclid Web Line. 7–11, 13, 14, 25–28, 32, 36, 39, 40, 47–49, 55, 60, 62, 63, 66, 69, 70, 72, 82, 95, 96, 99, 100, 104, 106, 110, 111, 114, 115, 117, 118, 122, 134, 142, 147, 148, 150, 156, 183, 184, 189, 201, 202, 208, 295, 308, 327 |
| Fast Fourier Transform | Fast Fourier Transform analysis fits a set of sinusoids to an input data set, see Appendix I. 19, 48, 52, 79, 80, 82, 87, 154, 155, 184 |
| HMI | (Human-Machine Interface) The control screen where the user inputs the desired values run speeds, tensions, motors for operation, and winding tapers and where the control system reports the line's current condition. 8, 13, 25, 179, 180 |
| HSC | High-Speed Counter. 183, 184 |
| HSLT | High-Speed, Low-Tension. 9–11, 13, 21, 25 |
| HSWL | High-Speed Web Line. 5, 12–14, 19, 25–28, 36, 45, 55, 60, 66–79, 82, 84, 85, 87–92, 94–96, 98, 134, 176, 177, 179–181, 201, 202, 235, 242, 295, 384, 402 |

| Notation | Description |
| --- | --- |
| Idle Roller | A free-spinning support also known as a roller or idler with which the web comes into contact that is allowed to spin about its long axis. 1, 17, 19, 23, 33 |
| Industrial S-Curve | A smooth curve between the starting value and a slope defined by the acceleration rate which then transitions from the slope to the ending value through a smooth curve governed by a constant jerk rate. More information is given in Appendix I.2. 36, 71, 77, 85, 87, 201, 202 |
| Loss Torque | The combination of all parasitic losses for a roller. 20 |
| Modeling | The act of assembling a set of, usually time-dependent, differential equations (models) to represent a physical system as close as can be had to the real system. 5, 6 |
| Nip | A roller that applies pressure to a web to force it into contact with a driven roller to guarantee no slip at that point. 2, 36, 198 |
| Nipped Roller | A driven roller with a nip roller sometimes called a bridle. 2, 8, 12, 17, 85 |
| Parent Roll | A large mass of web wound uniformly on a core which supplies a process with the raw material it needs. 1, 2, 6, 42, 67, 74, 79, 84–87, 143, 179, 227 |
| PDE | Partial Differential Equation. 19 |
| PI | Proportional plus Integral. 3–5, 24–27, 53, 60–62, 73, 98 |
| PID | Proportional, Integral, Differential. 14, 15, 18, 142 |

| Notation | Description |
| --- | --- |
| PLI | (per linear inch) A unit of tension where the force on a web is divided by that web's cross-machine direction length (or width since length in usually referring to machine direction measurements) in inches, e.g. a 2 foot wide web with 6 lbf tension would have 0.25 pli tension. 8, 12, 79, 176, 179 |
| Polyethylene Terephthalate | (PET) smooth, white plastic film. 76, 80, 384 |
| RA | Routh Approximation. 59, 64, 66, 68, 71, 72, 74–82, 89, 93–98, 124, 177, 272, 327, 402, *see also* Routh Approximation Method |
| Rewinding Roll | The roll of parent material, spindle, gear train, belts, and motor used to wind material onto a roll at the end of a process. 2, 20, 23 |
| Roll | A quantity of material wound on a hollow core that is the source (or sink) of a web transport system. These are part of an unwind or rewind. 20 |
| Routh Approximation Method | a method of reducing polynomials from higher order (or degree) to lower order (or degree) by a process that uses the Routh Array. The method maintains the stability of the starting polynomial. The method was created by M. Hutton in 1975. 7, 54, 60, 62, 71, 73, 98, 144, 402 |
| S-Wrap Dancer | A type of dancer that incorporates a pair of rollers in a fixture which is allowed to rotate about a parallel axis and causes the incoming and outgoing spans to increase or decrease in length based on the torque balance about the axis of rotation of the fixture. 39, 45 |

| Notation | Description |
| --- | --- |
| SFDR | Sliding Friction Driven Roller. 102, 103, 106, 108–110, 118, 119, 122, 174, 224 |
| Simulation | Solving the time-dependent set of mathematical equations from a given initial condition to obtain the response of the modeled system. 5 |
| Slack | A condition of a web wherein the tension in the span has dropped to zero and the span may begin to collect additional mass or alternatively, the condition of a span of material becoming longer than the distance between the two rollers that support it. 11, 23, 161 |
| Slip | The event when a web moves relative to a roller instead of being stationary with respect to the roller as the web travels over the extent of the roller with which it comes into contact. The web and roller surfaces are in contact during the relative motion. 2, 7, 8, 11, 99 |
| Span | The web material between two rollers which has a natural, unstretched length and mass, but is usually under tension so it has been stretched. 17, 19, 23 |
| Start-up | An event in a web line operational sequence where the motors transition from not rotating to rotating at a rate required to cause the web to travel at the desired line speed. 25, 26, 68, 71, 76, 78, 79, 87–91, 186 |
| State | a list of variables that have differential equations. 42, 84, 85, 96 |

| Notation | Description |
| --- | --- |
| Unwinding Roll | The roll of parent material, spindle, gear train, belts, and motor used to unwind material off the parent roll and into the process. 20 |
| Upstream | An ordinal term describing something being in the reverse direction of web travel from the rewind roll toward the unwinding roll. 2, 3, 17, 57, 100, 101 |
| Validate | Comparing a solution of a model to real data not used to create the model. 23 |
| Web | A flexible strip of material that can be quite long in length compared to its width and thickness, and can be many materials, e.g. cloth, metal, glass, or plastic. 1–4, 19 |
| Web Line | The combination of hardware, software, and material that unwinds, transports, processes, and finishes a product from a web. The hardware and software usually make up a feedback system that maintains the web linear speed and web tension throughout the processing. 1, 4, 7, 14, 62, 68, 71, 87, 95 |
| Web Transport System - V2.1 | A computer program created by John P. Newton and the WHRC to calculate web tension and speed dynamics. 36, 198 |
| WHRC | Web Handling Research Center. 7, 8, 10–13, 99, 111, 134, 148 |
| WTS | Web Transport System. 87, 198, 210, *see also* Web Transport System - V2.1 |

VITA

Benjamin David Reish

Candidate for the Degree of

Doctor of Philosophy

Dissertation: MODELING WEB HANDLING SYSTEMS AND A METHOD FOR DETERMINING FEEDBACK CONTROLLER GAINS

Major Field: Mechanical and Aerospace Engineering

Biographical:

Education:

Completed the requirements for the Doctor of Philosophy in Mechanical and Aerospace Engineering at Oklahoma State University, Stillwater, Oklahoma in July, 2020.

Completed the requirements for the Master of Science in Mechanical and Aerospace Engineering at Oklahoma State University, Stillwater, Oklahoma in 2015.

Completed the requirements for the Bachelor of Science in Mechanical Engineering at Oklahoma Christian University, Edmond, Oklahoma in 2004.

Experience:

Cognizant Engineer F101/F118 Engines Section, Tinker AFB, OK: Jet engine performance and engine accessories maintenance from 2005 to 2012.

Adjunct Professor Engineering and Science Division, Rose State College, Midwest City, OK: Statics, SP & SU 2016.

Adjunct Professor College of Engineering, Math, and Computer Science, Oklahoma Christian University, Edmond, OK: Systems II,III, 2014-15. Introduction to Solid Modeling, SP & FA 2018, SP 2019. Systems II,III, 2019-20.

Research Assistant Web Handling Research Center, Oklahoma State University, Stillwater, OK: SP 2016 to 2019.

Professional Memberships:

American Society of Mechanical Engineers (ASME)