REAL TIME DYNAMIC SIMULATION FOR CONTROL SYSTEM SOFTWARE VERIFICATION

by

R. Bettendorf
Metso Paper, Inc. (Valmet)
USA

ABSTRACT

It is a common practice in the paper industry to verify large portions of the control system software during machine commissioning. This is inefficient since the verification is done under production conditions when time is limited and delays are costly. The reason for this practice is that the machine, the drives, and other equipment are not integrated before installation and therefore no trial runs can be made with a web.

This paper describes a method for software verification using dynamic real time simulation. Primitive elements derived at the Web Handling Research Center are combined to form a system of simultaneous differential equations that model the behavior of the drives and the web. The system of equations is implemented in a programmable logic controller and solved in real time using a numerical integration algorithm. The outputs from the actual control software are used as inputs to the model and the simulated state variables are used as feedback for the software. This makes it appear to the software that all equipment has been integrated with the machine and a web is present.

Results are shown from a real time simulation that was successfully used on a winder rebuild to verify unwind brake tension control software. These results are compared with data from the actual machine. The differences and similarities are discussed with respect to the software verification process.

NOMENCLATURE

A	Cross-sectional area of the web
B_j	Viscous damping of a roll
E	Elastic modulus of the web
J_j	Inertia of a roll
$K_{E/P}$	Unwind voltage to pressure transducer gain
K_I	Integral gain of the drum drive speed regulator
K_P	Proportional gain of the drum drive speed regulator
K_{TB}	Unwind brake torque constant
K_{TM}	Windup motor torque constant
L	Web span length
N	Integer multiple of the timed interrupt sample period
n	Integral term of the drum drive speed regulator
p	Unwind brake pressure
R_j	Roll radius
Δt	Numerical integration time step
T_s	Sample period of the timed interrupt program file
t_j	Web tension
t_{1ref}	Web tension reference
V	Voltage
v_j	Surface velocity of a roll
x	Example state variable
v_{2ref}	Drum drive speed reference
$\tau_{E/P}$	Unwind voltage to pressure transducer time constant

Subscripts

k	An integer representing the present scan
j	1 = unwind, 2 = windup

INTRODUCTION

Software verification is the process of determining if a program will correctly meet it's specifications and not perform unintended functions. In the paper industry, large portions of this process are completed during machine commissioning. New web handling machinery is typically a multi-vendor effort between the machine builder and the electric drive supplier. Each vendor manufactures their equipment independently at their own facility, so only limited software interface testing is done between the two systems. When web handling machinery is rebuilt, one or more sections of the existing machine are replaced by new equipment. Whether or not this is a multi-vendor effort, some of the machinery is already in place at the mill site and is unavailable for testing. In both cases, there is limited opportunity to test the integration of the new software with other new or existing equipment before commissioning. Unfortunately, there is little time available during the machine commissioning for completing this task.

Programmable logic controllers (PLCs) with custom control software are used extensively in the paper industry to control web handling machinery. This paper describes a method to verify PLC control software by using real time dynamic simulation to model web handling machinery when it is unavailable for testing. The individual components used in the method are reviewed in the first part of this paper. The first component, numerical integration, is used to solve the dynamic model of the machinery. The next component, real time programming in the PLC, is used to scale the simulation

time to correspond to actual time. The final component is the modeling of the hardware interface between the PLC and the machinery. These components are combined to continuously solve, in real time, the differential equations that describe the states of the web handling machinery. The simulation is then coupled to the control software so that the verification process can be performed without the machinery.

An example is presented in the second part of this paper that explains how real time dynamic simulation was used to verify the software of a winder rebuild. The configuration of the actual winder was used to develop a simplified model. Then a simulation that used this simplified model was implemented in a PLC and used to verify whether the control software met its specifications. It is also shown that this method can simulate web handling machinery with sufficient accuracy to make the verification process meaningful.

THE SIMULATION METHOD

A web handling machine and its PLC based control system is shown in figure 1. The state of the machinery is returned as inputs through a hardware interface to the program in the PLC. The program is continuously cycling and processes the inputs to generate commands. The new commands are sent as outputs to the machinery through the interface.

Since the web handling machinery is unavailable, a mathematical model of it is formed consisting of a system of ordinary differential equations. This system of equations is then programmed into the PLC and solved using classical numerical integration methods. Figure 1 shows that the simulation of the physical system is done in the same PLC in which the control software is implemented. This reduces the amount of hardware required for testing and reduces the development time and costs of the simulation. The equations are solved with real time programming techniques so that simulation time is equal to actual time. The hardware interface is also modeled so that its effect on the dynamics of the system are accounted for.

To implement the simulation, the signal path between the control software and the hardware interface is temporarily broken as shown in figure 1. The model is then inserted into the signal path to generate the appropriate inputs for the control software based on the outputs.

Numerical Integration

Web handling machinery can be modeled by creating a system of ordinary differential equations based on the primitive elements derived at the Web Handling Research Center. This system of equations can then be solved with a numerical integration algorithm. These algorithms use only sums and products to find approximate solutions to a differential equation at a specified interval or time step Δt.

The fourth order constant step size Runge-Kutta algorithm [1] was chosen to simulate the physical system. This algorithm can accurately solve systems of differential equations with a moderate computational burden. It is also numerically stable as long as Δt is sufficiently short. To illustrate how the algorithm works, we begin with the first order differential equation shown in (1) that is a function of time t and the state variable x. If the differential equation had been second order or higher, it would need to be reduced to a system of first order differential equations.

$$\dot{x} \equiv \frac{dx}{dt} = f(t, x) \tag{1}$$

The derivative can be approximated by a finite change in the state variable over a discrete time step. Equation (1) can then be rewritten as:

$$\frac{\Delta x}{\Delta t} \approx f(t,x) \tag{2}$$

Both sides of (2) are multiplied by Δt to solve for the finite change Δx of the state variable. Converting to a difference equation results in the form shown in (3). The fourth order Runge-Kutta algorithm extends this by calculating four estimates of the change at different points of the interval Δt. The first estimate is taken at t_{k-1} in (3). The next two estimates are taken at the trial midpoint $t_{k-1} + 1/2 \cdot \Delta t$ in (4) and (5). The final estimate is taken at the trial endpoint $t_{k-1} + \Delta t$ in (6).

$$\Delta x1 = \Delta t \cdot f(t_{k-1}, x_{k-1}) \tag{3}$$

$$\Delta x2 = \Delta t \cdot f\left(t_{k-1} + \frac{1}{2}\Delta t, x_{k-1} + \frac{1}{2}\Delta x1\right) \tag{4}$$

$$\Delta x3 = \Delta t \cdot f\left(t_{k-1} + \frac{1}{2}\Delta t, x_{k-1} + \frac{1}{2}\Delta x2\right) \tag{5}$$

$$\Delta x4 = \Delta t \cdot f(t_{k-1} + \Delta t, x_{k-1} + \Delta x3) \tag{6}$$

A weighted average of the estimates is then taken. The weighting values are chosen with a mathematical procedure which ensures that the error is minimized. This average change is added to the value of the state variable from the previous scan to find the present value as shown in (7).

$$x_k = x_{k-1} + \Delta x = x_{k-1} + (\Delta x1 + 2 \cdot (\Delta x2 + \Delta x3) + \Delta x4) \cdot \frac{1}{6} \tag{7}$$

Timed Interrupts and Sample Rate Selection

Real time software executes with a deterministic time interval. Solving the equations of state in real time is necessary so that simulation time is scaled to correspond to actual time. A real time implementation also holds the sample rate constant. This prevents movement of the model's poles and zeros which would cause the dynamics to vary. Real time performance is achieved in a PLC by implementing the software in a timed interrupt. The main program execution or scan of the PLC is stopped at a preset time interval or sample period T_s. This allows the PLC to scan the program contained in the interrupt file. The main program execution resumes when the program in the timed interrupt file is finished. This procedure is shown graphically in figure 2a. An additional benefit of using a timed interrupt is that it allows the interrupt program to be executed more often than the main program is executed.

The sampling theorem requires that the sample frequency of a discrete approximation must be greater than twice the bandwidth of the continuous system. If this is not done, frequencies higher than twice the sample frequency will appear to be a lower frequency or an alias in the discrete approximation [2]. However, the sampling theorem only provides a lower bound on the sample frequency that prevents aliasing. It does not guarantee that the response of the discrete approximation will closely match the response

of the continuous system. A sample frequency at least 10 times the bandwidth of the continuous system is needed for a smooth discrete response which closely resembles the continuous response [3]. However, choosing a sample frequency that is too fast can drastically increase the main program scan time by interrupting it too often as shown in figure 2b. The PLC must also have enough computational power to execute the interrupt program in less time than the sample period. As the execution time of the interrupt program approaches the length of the sample period, the main program scan time will increase as shown in figure 2c.

The integration algorithm is implemented in the timed interrupt program file of the PLC. The time step Δt of the integration algorithm is set equal to the sample period T_s of the interrupt. Setting these two values to be equal scales the simulation time to match actual time.

Interface Modeling

The hardware interface between the PLC program and the web handling machine can consist of analog-to-digital (A/D) converters, digital-to-analog (D/A) converters, encoder inputs, or even a communications link to another computer system. The interface models that will be discussed in detail are for an A/D converter and a D/A converter. The encoder input would be modeled like an A/D converter and a communications link would be modeled as a pure time delay.

The A/D converter is modeled with a discrete time sampler. Simulating the physical system in a discrete device like a PLC means that the differential equations must be solved with numerical integration algorithms. To make the physical system appear continuous to the control software, the integration algorithm must be updated at a faster rate than the control software. This means that the sample rate of the timed interrupt must be set to what is required by the physical system model as shown in figure 3a. A discrete time sampler is then used as shown in figure 3b to take data points from the solution at an integer multiple N of the sample rate T_s. The down sampled data points will be used in the controller difference equations to calculate the controller output. The discrete time sampler suffers from aliasing just as the more familiar continuous time sampler does [2].

The stair step output of a D/A converter is shown in figure 4a. A zero order hold (ZOH) is used to model this characteristic of the D/A converter. The outputs of the control software are calculated every $N \cdot T_s$ seconds and held constant in the PLC's memory between calculations. The physical system model is being solved every T_s seconds, so it accesses these values between the control software calculations. This inherent ZOH models the operation of a D/A converter as shown in figure 4b.

ANALYSIS AND IMPLEMENTATION

An example is presented in this section that explains how a real time simulation was implemented for a winder that was rebuilt to increase it's throughput. The unwind parent roll uses a mechanical brake actuated by pneumatic airbags to maintain web tension. A portion of the rebuild involved replacing the pneumatic load cells and a pneumatic tension regulator with electronic load cells and a discrete tension regulator in the PLC. The pressure in the airbags is now controlled by a voltage to pressure transducer (E/P). The brake torque is approximately proportional to this pressure. The configuration of the actual winder is shown in figure 5.

A system of 17 first order differential equations would be required to model the winder as shown in figure 5. To reduce the computational burden of the simulation, the winder model was simplified. The web driven rolls between the unwind and windup were eliminated by assuming that they had negligible inertia. The driven guide roll was

eliminated by assuming that it's surface velocity exactly matched the web velocity. The lengths of the separate web spans were then summed to form one long span. The drums were combined to form one equivalent roll by assuming that there was no slippage between the wound roll and the drums. This results in the windup being modeled as a single inertia. The winder model that was formed with these simplifying assumptions is shown in figure 6.

The differential equations that describe the simplified winder model are given by (8) through (12) and are based on the primitive element models found in [4]. The pneumatic brake and E/P are modeled in (8) as a first order lag with the gain $K_{E/P}$ and the time constant $\tau_{E/P}$. The unwind parent roll is modeled in (9) as a simple inertia-damper system with the brake torque proportional to pressure. The web span is modeled in (10) as a nonlinear element whose tension-velocity relationship is based on mass flow. The windup is modeled as a single inertia-damper system in (12). A proportional-integral (PI) speed regulator was incorporated in the windup model to control the surface velocity of this roll. The proportional term is present in (12) while the integral term of the speed regulator is treated as a separate state in (11). The inputs into this model are the voltage signal V to the E/P and the speed reference v_{2ref} to the windup section. The voltage signal is an output from the actual control software that is being verified. The speed reference is created by an S-curve generator in the simulation software. The web tension t_1 passes through the discrete time sampler to be used as an input for the tension regulator in the control software.

$$\dot{p} = \frac{K_{E/P}}{\tau_{E/P}} V - \frac{1}{\tau_{E/P}} p \tag{8}$$

$$\dot{v}_1 = -\frac{B_1}{J_1} v_1 + \frac{R_1^2}{J_1} t_1 - \frac{R_1}{J_1} K_{TB} p \tag{9}$$

$$\dot{t}_1 = \frac{v_2}{L}(EA - t_1) - \frac{v_1}{L}(EA - t_0) \tag{10}$$

$$\dot{n} = K_I (v_{2\,ref} - v_2) \tag{11}$$

$$\dot{v}_2 = -\frac{B_2}{J_2} v_2 - \frac{R_2^2}{J_2} t_1 + \frac{R_2}{J_2} K_{TM} \left[K_p (v_{2\,ref} - v_2) + n \right] \tag{12}$$

Equations (8) through (12) were solved by the Runge-Kutta algorithm in the timed interrupt file. The initial conditions were manually entered into the memory of the PLC before placing it into run mode. The control software was also located in the timed interrupt file, but it was only executed every N interrupts. Figure 7 shows a block diagram of the actual system and a block diagram of how it was simulated in the PLC. Pressure, windup speed reference, windup speed, and tension are displayed on a human-machine interface (HMI) for use by the operator.

The shortest time constant in the system of differential equations was used as a guideline to choose the sample period for the timed interrupt. The unwind parent roll was nearly a pure integrator with a time constant measured in minutes. The time constant of the pneumatic system $\tau_{E/P}$ was 3 s. The speed regulator was tuned so that the windup had an overdamped response similar to a first order system with a 1 s time constant. The web

span is a nonlinear system with a time constant L/v_2 that varies with the velocity v_2. Near zero speed, L/v_2 is very long and the web span resembles a pure integrator. At the maximum winder speed, L/v_2 is 160 ms which is the shortest time constant in the model. This meant that the sample period needed to be shorter than 80 ms. The PLC required 5 ms to execute the simulation which meant that the sample period needed to be longer than 5 ms. To provide time for the main program to execute, the sample period of the timed interrupt was set to 10 ms. The resulting Δt was 16 times shorter than the time constant of the web span at maximum speed. This allowed the response of the discrete simulation to closely match the response of the continuous system. The sample rate of the control software was 50 ms which required that N = 5 for the discrete time sampler.

RESULTS

Two sets of results will be presented. The first set of results show how well the model represents the web tension. It is desirable to have a physical system model that is accurate so that the software verification process is meaningful. The second set of results discusses what the expectations were for the simulation. It explains what techniques were used to verify the software and the end result of the verification process during the machine commissioning.

Simulation Results

The simulation results presented here were done after machine commissioning. The gains determined during commissioning were used to compare the actual tension with the simulated tension. The results in figures 6 and 7 show that near zero speed the model was not accurate. This is because a motor model was used to model the brake instead of deriving a nonlinear brake model based on static and dynamic friction. Mechanical brake torque is generated by the friction force of the brake pads on the drum. Therefore, it can only oppose motion; it can not cause motion in the direction of the torque like a motor can. The actual machine also had an oscillatory response that the simulation did not exhibit. This difference could be due to the simplifying assumptions used in the model. The difference in response had no effect on the verification process, but the brake model that was used prevented extensive testing near zero speed.

The cross coupling that existed between the windup velocity and tension is shown in figures 8 and 9. The tension temporarily dropped below the setpoint when the windup velocity rounded out of the S-curve. This was caused by the relatively sudden loss of retarding torque when the unwind stopped accelerating. The tension regulator responded to the drop in tension by increasing the braking pressure which caused the tension to return to the setpoint value. It can be seen that the physical system model accurately represents the cross coupling and the response of the tension regulator.

As seen in figures 10 and 11, the model closely resembles the physical system at steady state. This indicates that the low frequency parameters in the model are relatively close to the actual values. However, it can be seen in figure 10 that the actual pressure was slowly decreasing while figure 11 shows that the simulated pressure was constant. This discrepancy occurred because the unwind radius was treated as a time varying parameter in the model. The unwind brake pressure must decrease to maintain a constant tension as the parent roll radius decreases. The change in radius was very slow compared to the system time constants. Therefore, it was possible to treat R_1 as a time varying parameter instead of as a separate state. Different values of R_1 (and J_1) were entered in the PLC's memory between simulation trials and held constant during the simulation. This allowed verification at different operating points without the extra computational time required to solve another state equation.

Verification Results

The verification results presented here describe how real time simulation was used to test the control software. The objectives of the software verification process were to:

- verify the HMI for proper functionality
- verify that the control software functions properly and that there are no undesirable numerical phenomena
- have the control software functionality at a level sufficient to allow the machine to run with paper immediately after commissioning
- reduce the amount of software modification that is usually necessary after commissioning

The verification techniques used to achieve these objectives included interface testing, prototyping, functional testing, and performance testing [5].

The functionality of the HMI was partially verified with interface testing. This test focused on the communications interface between the HMI software and the PLC software. All operator controls on the HMI were moved to their extreme positions while observing the corresponding values in the PLC. Conversely, values in the PLC were forced to their upper and lower limits while observing data displayed on the HMI. This procedure verified that the variables in the HMI were reading from and writing to the proper memory locations in the PLC. It also verified that the scaling in the HMI matched the scaling in the PLC. After this test was complete, all data was being properly passed between the HMI and the PLC.

The simulation was able to provide a prototyping capability. Since this was a custom machine rebuild, it was unknown what operator controls would be best suited for the application. Various combinations of operator control devices were programmed on the HMI and used to control the simulated machine. This was done until a suitable set of control devices were found. Functional testing was then performed for both the control software in the PLC and the operator controls on the HMI to prove that the operator requirements were met. This was accomplished with the simulation by accelerating up to speed, performing a normal stop, performing an emergency stop, and changing tension levels. It was established during the simulated test runs that the tension regulator algorithm regulated properly. This portion of the testing was also used to verify that there were no discontinuous changes in the voltage signal sent to the E/P when the tension regulator was switched between automatic and manual modes of operation. The functional testing proved that the control software functioned properly, the HMI software functioned properly, and there were no undesirable numerical phenomena.

Performance testing was conducted on the real time portion of the control software by monitoring the interrupt scan time in the PLC. This was done to ensure that the control software could execute quickly enough to allow real time operation. It was also important to verify that the software execution time did not approach the length of the sample period and degrade the performance of the main program.

The simulation was used to determine initial tension regulator gains. The proportional gain obtained during the simulation was .001 and the integral gain was 2. For comparison, the final proportional gain on the actual machine was .0045 and the final integral gain was 4. Although the initial gains were relatively small compared to the final gains, they resulted in a stable tension control system on the actual machine. This made it possible to run with paper immediately after commissioning. The only software modification that needed to be made after the commissioning was to add an unwind brake pressure step during deceleration. The purpose of the pressure step was to prevent the web tension from decreasing while the winder was stopping. Other work performed after commissioning was tuning the tension regulator and adjusting parameters in the software.

CONCLUSION

This paper described a method for verifying control software using real time dynamic simulation. Results were presented from an actual winder rebuild that used this method. The simulation results showed that this method can accurately model a web handling machine and provide for a meaningful verification process. The results of the verification included a discussion of the testing techniques used and the success experienced during machine commissioning.

There are many avenues to pursue in future work. Modeling the parent roll build down and wound roll build up as separate states will allow more accurate simulations and will allow additional control software capabilities to be verified. Another area for further investigation is the tradeoff between integration algorithm complexity and physical system model complexity. Finally, the use of real time dynamic simulation for operator or technician training can be explored.

REFERENCES

1. Wylie, C. Ray and Barrett, Louis C., Advanced Engineering Mathematics , 5th ed. , McGraw-Hill Book Co., New York, 1982, pp. 267-296.

2. Oppenheim, Allen V. and Willsky, Alan S., Signals and Systems , Prentice-Hall, Inc., New Jersey, 1983, pp. 514-551.

3. Franklin, Gene F., Powell, J. David, and Workman, Michael L., Digital Control of Dynamic Systems , 3rd ed. , Addison Wesley Longman, Inc., Menlo Park, CA, 1998, pp. 449-476.

4. Reid, Karl N., Tree, Alan, and Newton, John, "Seminar on The Analysis and Design of Web Transport Systems", Tab 1, Oklahoma State University March 1996.

5. Wallace, Dolores R., Ippolito, Laura M., and Cuthill, Barabara, "Reference Information for the Software Verification and Validation Process", NIST Special Publication 500-234, U.S. Department of Commerce, Gaithersburg, MD, 1996.

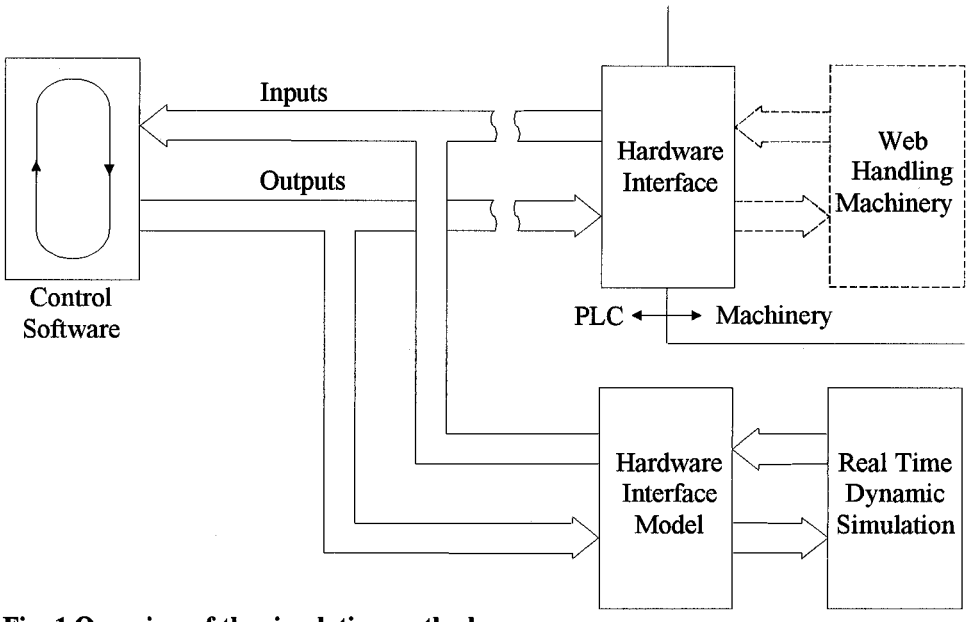
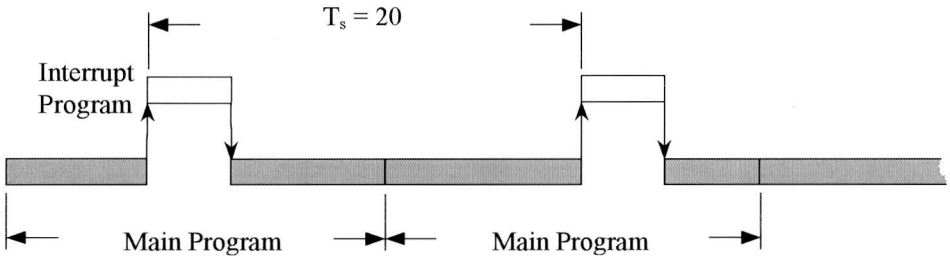
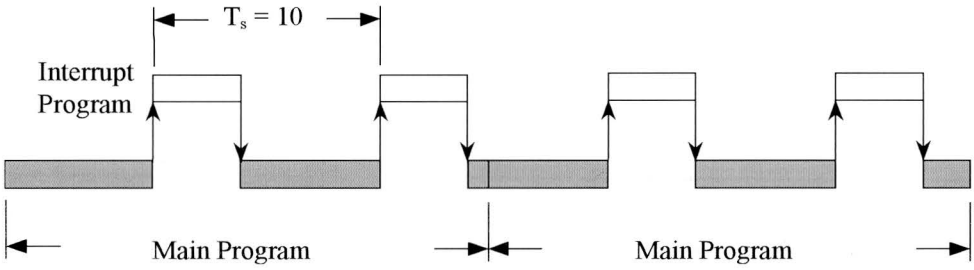


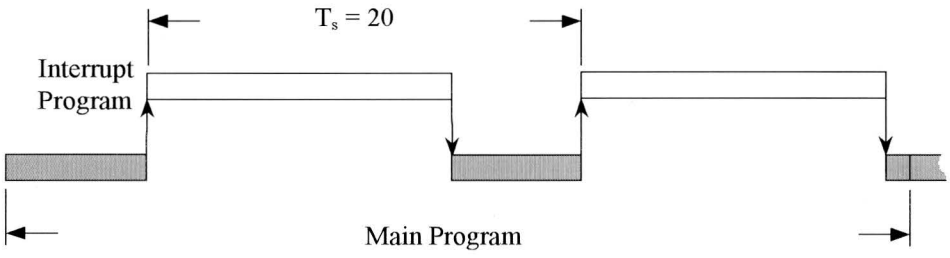
Fig. 1 Overview of the simulation method.



a) Timed interrupt with T_s = 20 ms.

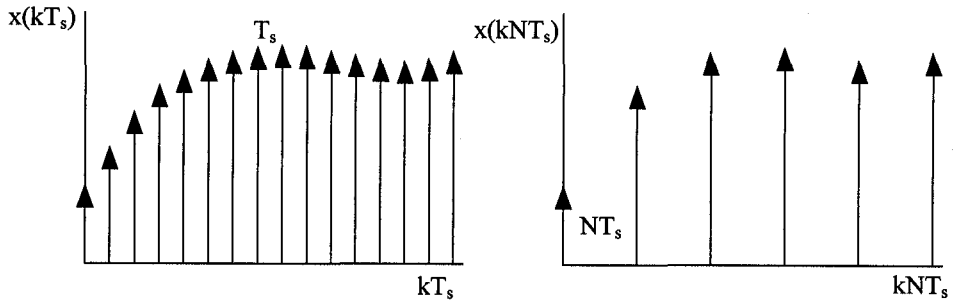


b) Timed interrupt with T_s = 10 ms.



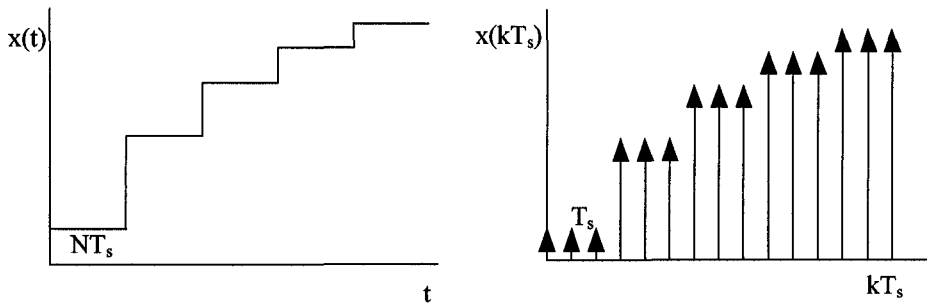
c) Timed interrupt with T_s = 20 ms and an execution time that approaches T_s.

Fig. 2 Effect of timed interrupts on main program scan times.



a) Physical system model sampling. **b) Discrete time sampling of $x(kT_s)$.**

Fig. 3 Physical system model sampling and discrete time sampling.



a) D/A converter output. **b) Simulated ZOH.**

Fig. 4 D/A converter output and simulated ZOH.

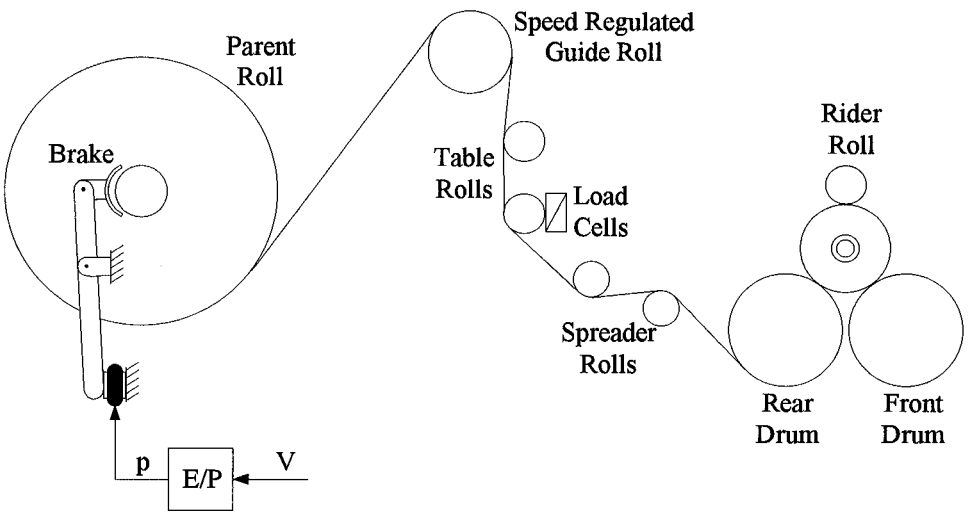


Fig. 5 Actual winder layout.

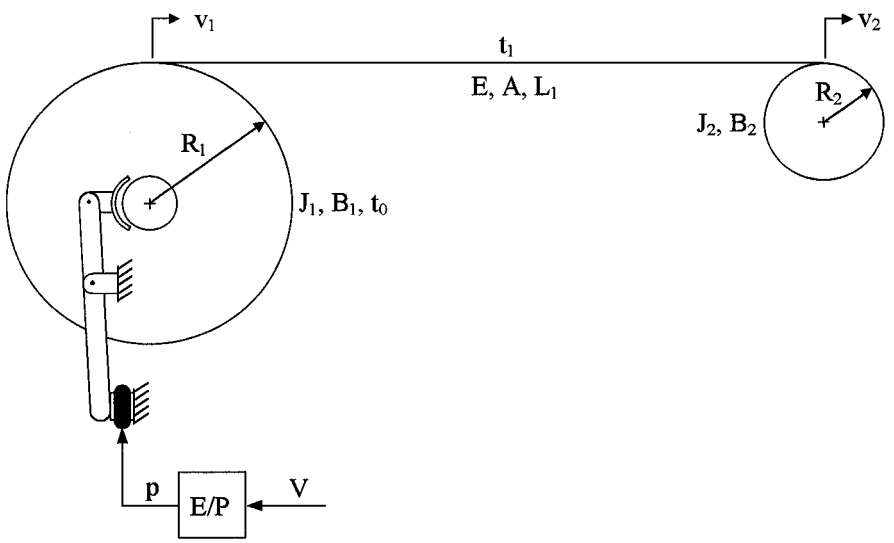
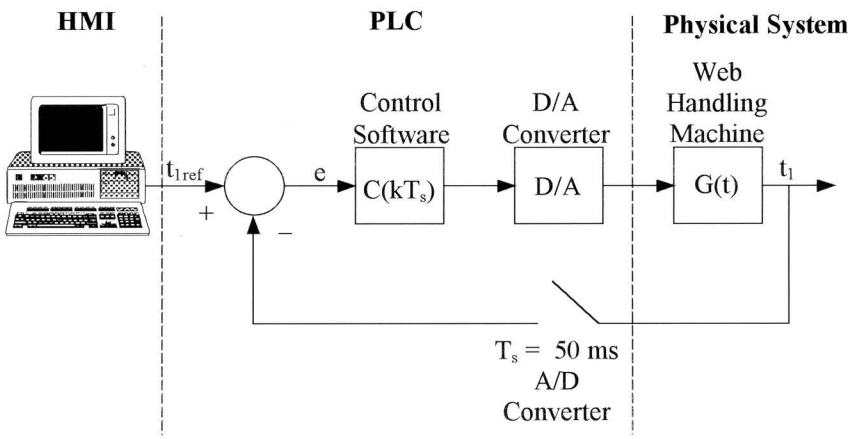
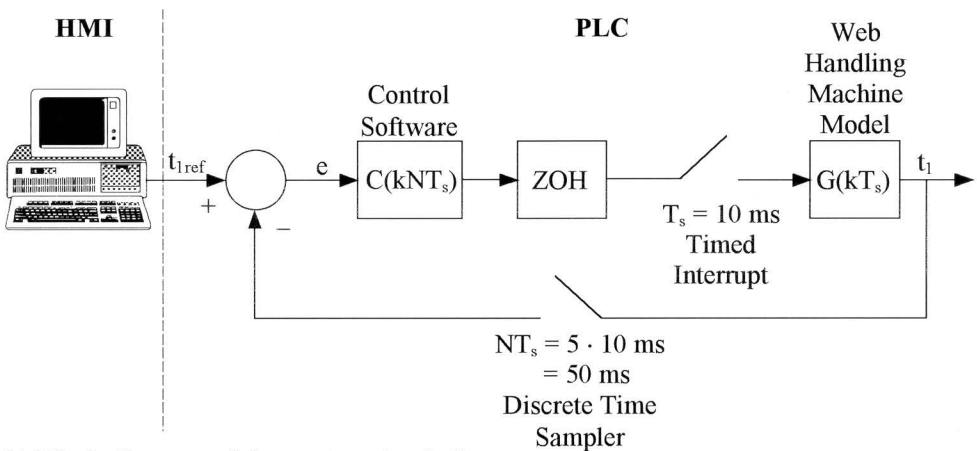


Fig. 6 Simplified layout of the winder.



a) Block diagram of the actual system.



b) Block diagram of the system simulation.

Fig. 7 Block diagram of the real time dynamic simulation.

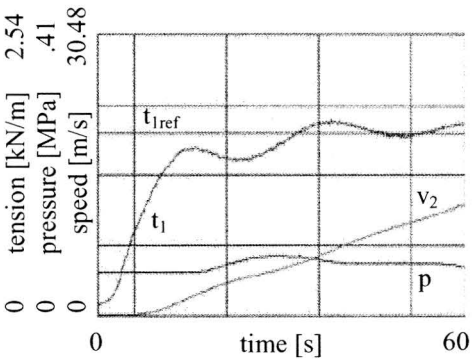


Fig. 8 Actual acceleration.

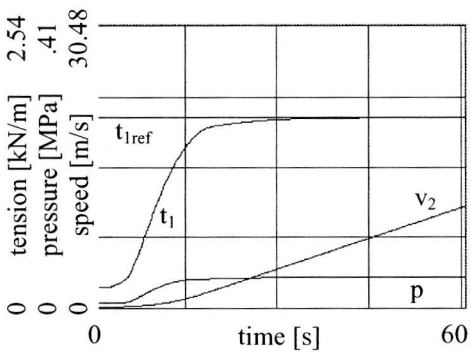


Fig. 9 Simulated acceleration.

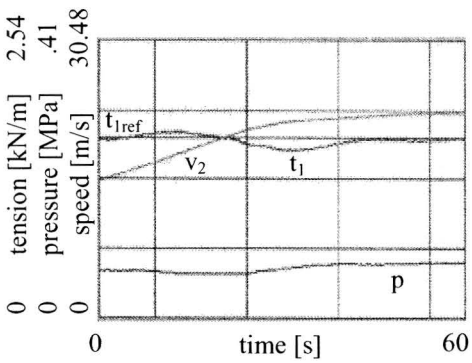


Fig. 10 Actual S-curve round out.

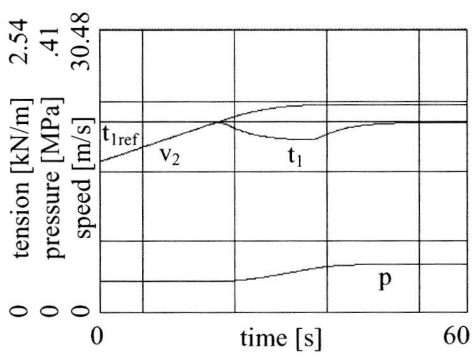


Fig. 11 Simulated S-curve round out.

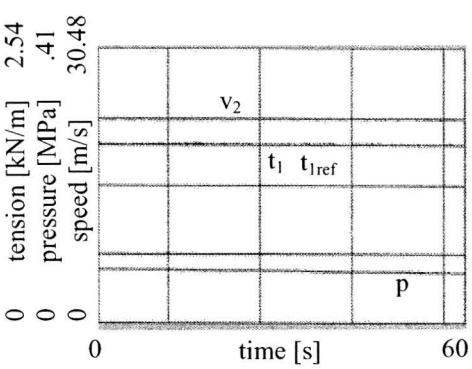


Fig. 12 Actual steady state.

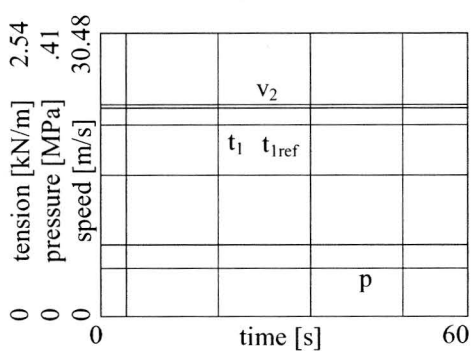


Fig. 13 Simulated steady state.

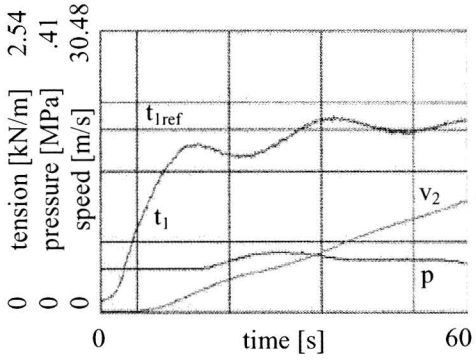


Fig. 8 Actual acceleration.

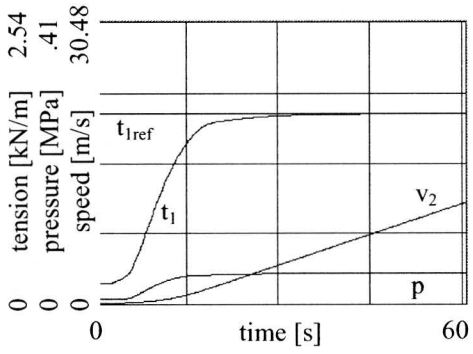


Fig. 9 Simulated acceleration.

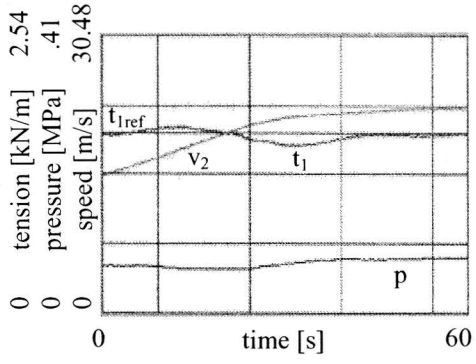


Fig. 10 Actual S-curve round out.

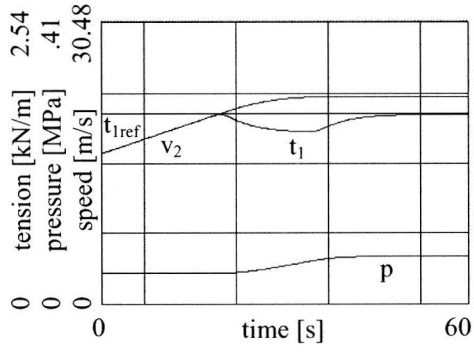


Fig. 11 Simulated S-curve round out.

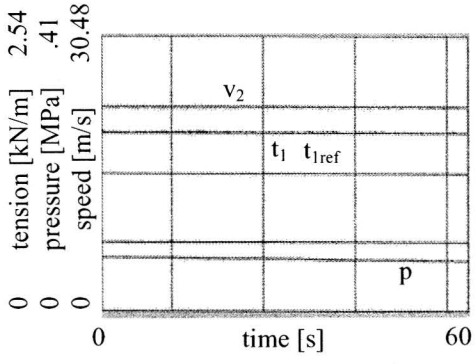


Fig. 12 Actual steady state.

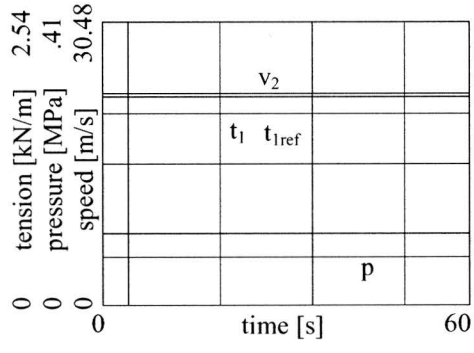


Fig. 13 Simulated steady state.

Real Time Dynamic Simulation for Control System Software Verification **R. Bettendorf** – Valmet, USA

Name & Affiliation	Question
P. Pagilla – OSU	Usually in control we want to measure the feedback signal at a higher sampling rate than the rate at which we send a control signal. In your verification, you were sampling at 50 milliseconds whereas your control input was at 10 milliseconds. Would you comment on that? Part of the reason we sample the feedback at a higher rate is so we can filter the measured signal and control based upon a filtered or averaged feedback signal.
Name & Affiliation	**Answer**
R. Bettendorf – Metso Paper	Normally, you desire the fastest sampling rate possible. I was simulating a continuous system in a discrete device. In order to make that continuous system look continuous to my controller software, I had to employ a much higher sample rate on my discrete model of a continuous system.