

TAKE HOME LABS AND THE FURUTA PENDULUM

By

SEAN HENDRIX

Bachelor of Science in Electrical Engineering

Oklahoma State University

Stillwater, Oklahoma, United States of America

2012

Submitted to the Faculty of the  
Graduate College of  
Oklahoma State University  
in partial fulfillment of  
the requirements for  
the Degree of  
MASTER OF SCIENCE  
December, 2015

TAKE HOME LABS AND THE FURUTA PENDULUM

Thesis Approved:

Dr. Martin Hagan

---

Thesis Advisor

Dr. Carl Latino

---

Committee Member

Dr. George Scheets

---

Committee Member

## ACKNOWLEDGMENTS

Firstly, I would like to thank Dr. Hagan for his endless support, patience, and knowledge provided to me during the writing process of this thesis. Without his guidance, I would not have come this far with the research and thesis. I would like to also thank Carion Pelton for his collaboration with me on the Take Home Labs concept. We began with nothing but an idea, and now it has come to fruition. Many thanks to Amir Jafari for his support of Carion and I while researching his Phd.

Additionally, I would like to thank my parents Floyd and Mary Lou Hendrix, my brother Paul Hendrix, and my grandparents Betty and Floyd Hendrix for the endless love and support they have poured out to me while writing this thesis. They have always been there for me, through thick and thin, and I love them all very much.

Lastly, praise be to God for his support and all of my prayers he has answered. I truly would have been in a rough place without my faith in the Lord. I dedicate this thesis to my grandfather Floyd Hendrix. May your soul and spirit be at peace among the angels in heaven, and with our Heavenly Father. <sup>1</sup>

---

<sup>1</sup>Acknowledgements reflect the views of the author and are not endorsed by committee members or Oklahoma State University.

Name: Sean Hendrix

Date of Degree: December, 2015

Title of Study: TAKE HOME LABS AND THE FURUTA PENDULUM

Major Field: Electrical Engineering

Students garner a better understanding of concepts in control courses using hands-on labs; however, labs hosted at universities are often expensive and limited to specific times. Take Home Labs consist of a series of inexpensive labs, costing less than the price of a textbook, without the need for a teaching assistant (TA). Take Home Labs can be performed at home, and at a time convenient for students, eliminating the need for university laboratory space. The Take Home Labs website <<http://thl.okstate.edu>> was created to host all of the information needed in completing experiments. This thesis first describes the website and main hardware and software common to the experiments currently on the website. Next, 5 experiments are described. The main experiment featured in this thesis is the Furuta Pendulum, also known as the Rotary Inverted Pendulum. Finally, the thesis is concluded with the results found and future work on the Take Home Labs.



## TABLE OF CONTENTS

Chapter	Page
<b>1 INTRODUCTION</b>	<b>1</b>
<b>2 OVERALL FRAMEWORK</b>	<b>7</b>
2.1 Website . . . . .	9
2.1.1 Experiments . . . . .	9
2.1.2 Courses . . . . .	11
2.1.3 Participate . . . . .	12
2.2 Selection of Hardware Used In the Labs . . . . .	14
2.2.1 Previously Considered Common Components . . . . .	14
2.2.2 Final Common Components . . . . .	16
2.2.3 3D Printer (Solidoodle SD4) . . . . .	20
2.3 Selection of Software . . . . .	20
2.3.1 MATLAB/Simulink . . . . .	20
2.3.2 Repetier Host . . . . .	21
2.4 3D Printed Motor Load (Pennies) . . . . .	21
2.4.1 Description . . . . .	21
2.4.2 Insert . . . . .	22
2.5 Furuta Pendulum (Pole Positioning State Feedback) . . . . .	23
2.5.1 Model . . . . .	24
2.5.2 Development of Hardware . . . . .	26
2.5.3 Final Hardware Design . . . . .	28
2.6 Summary . . . . .	30

<b>3</b>	<b>Simple DC Motor</b>	<b>31</b>
3.1	Objective . . . . .	31
3.2	Setup . . . . .	31
3.2.1	Required . . . . .	32
3.2.2	Hardware Setup . . . . .	33
3.2.3	Software Setup . . . . .	39
3.3	Experimental Procedures . . . . .	52
3.3.1	Exercise 1: Comparing Duty Cycles . . . . .	52
3.3.2	Exercise 2: Slider Gain Analysis . . . . .	52
3.3.3	Exercise 3: Normal Mode . . . . .	53
3.4	Conclusion/Student Feedback . . . . .	54
3.4.1	Conclusion . . . . .	54
3.4.2	Student Feedback . . . . .	54
<b>4</b>	<b>Introduction to 3D Printing</b>	<b>55</b>
4.1	Objective . . . . .	55
4.2	Setup . . . . .	55
4.2.1	Required . . . . .	55
4.2.2	Software Setup . . . . .	57
4.2.3	Hardware Setup . . . . .	57
4.3	Experimental Procedures . . . . .	64
4.3.1	Exercise 1: Adjusting Print Surface . . . . .	64
4.3.2	Exercise 2: Printing Motor Load . . . . .	69
4.3.3	Exercise 3: Printing Insert . . . . .	71
4.4	Conclusion/Student Feedback . . . . .	74
<b>5</b>	<b>Open Loop Step Response</b>	<b>75</b>
5.1	Objective . . . . .	75

5.2	Setup . . . . .	75
5.2.1	Required Materials . . . . .	75
5.2.2	Hardware Setup . . . . .	76
5.3	Experimental Procedures . . . . .	93
5.3.1	Exercise 1: Deriving Motor Transfer Function . . . . .	93
5.3.2	Exercise 2: Theoretical Open Loop Step Response . . . . .	95
5.3.3	Exercise 3: Open Loop Step Response Variables . . . . .	96
5.3.4	Exercise 4: Experimental Open Loop Step Response . . . . .	98
5.3.5	Exercise 4: Find Transfer Function from Experimental Data . . . . .	104
5.3.6	Exercise 6: Compare Simulation and Experimental Results . . . . .	106
5.4	Conclusion/Student Feedback . . . . .	114
<b>6</b>	<b>Closed Loop Step Response</b>	<b>115</b>
6.1	Objective . . . . .	115
6.2	Setup . . . . .	115
6.2.1	Required Materials . . . . .	115
6.2.2	Hardware Setup . . . . .	116
6.2.3	Software Setup . . . . .	116
6.3	Experimental Procedures . . . . .	116
6.3.1	Exercise 1: Deriving Closed Loop Motor Transfer Function . . . . .	116
6.3.2	Exercise 2: Simulated Closed Loop Step Response (First Feedback Gain Set) . . . . .	123
6.3.3	Exercise 3: Experimental Closed Loop Step Response (First Feedback Gain Set) . . . . .	129
6.3.4	Exercise 4: Simulated Closed Loop Step Response (Second Feedback Gain Set) . . . . .	136
6.3.5	Exercise 5: Experimental Closed Loop Step Response (Second Feedback Gain Set) . . . . .	136

6.4	Conclusion/Student Feedback . . . . .	140
<b>7</b>	<b>Pole Positioning State Feedback</b>	<b>141</b>
7.1	Objective . . . . .	141
7.2	Setup . . . . .	141
7.2.1	Required Materials . . . . .	141
7.2.2	Software Setup . . . . .	143
7.2.3	Hardware Setup . . . . .	144
7.3	Experimental Procedures . . . . .	161
7.3.1	Exercise 1: Open Loop System Verification . . . . .	162
7.3.2	Exercise 2: System Linearization . . . . .	165
7.3.3	Exercise 3: Linear State Feedback Using Pole Positioning . . . . .	167
7.3.4	Exercise 4: Nonlinear State Feedback Using Pole Positioning . . . . .	168
7.3.5	Exercise 5: Experimental State Feedback Using Pole Positioning . . . . .	174
7.3.6	Exercise 6: Pole Positioning (Experimental vs. Simulations) . . . . .	178
7.4	Conclusion . . . . .	180
<b>8</b>	<b>CONCLUSIONS</b>	<b>181</b>
8.1	Summary . . . . .	181
8.2	Contributions . . . . .	181
8.3	Future Work . . . . .	182
	<b>BIBLIOGRAPHY</b>	<b>184</b>

## LIST OF TABLES

Table		Page
5.1	Transfer Function Values for G blocks . . . . .	94
5.2	Code for Plotting Experimental Results . . . . .	105
5.3	Code for Plotting Simulation Results (Added to <b>OL_Plot.m</b> ) . . . . .	111
6.1	Results Found From Simulating Different Gains and Load Configurations	120
6.2	Code for Plotting the Closed Loop Step Response Simulated Results .	128
6.3	Code for Sign to Direction Function) . . . . .	130
6.4	Code for Plotting the Closed Loop Step Response Experimental Results	135

## LIST OF FIGURES

Figure		Page
2.1	Take Home Labs Homepage . . . . .	9
2.2	Take Home Labs Experiments Page . . . . .	9
2.3	Take Home Labs Courses Page . . . . .	11
2.4	Take Home Labs Participate Page . . . . .	12
2.5	First Considered Motor Shield (SN754410) . . . . .	15
2.6	First Considered Power Supply (S-50-24) . . . . .	16
2.7	Final Motor Shield (DFRobot L298P) . . . . .	16
2.8	Final Power Supply (Universal AC Adapter) . . . . .	17
2.9	Final Microcontroller (Arduino Mega 2560) . . . . .	18
2.10	Final Motor/Encoder (Mitsumi M25N-2R-14) . . . . .	18
2.11	Final Pendulum Encoder (CUI AMT103) . . . . .	19
2.12	Solidoodle 3D Printer (SD4) . . . . .	20
2.13	3D Printed Motor Load . . . . .	21
2.14	3D Printed Motor Load Insert . . . . .	22
2.15	Furuta Pendulum Model (Circled x Denotes Center of Mass) . . . . .	24
2.16	Furuta Pendulum Model (First Revision) . . . . .	26
2.17	Furuta Pendulum Model (Second Revision) . . . . .	27
2.18	Furuta Pendulum Model (First Revision) . . . . .	28
3.1	Hardware Required for Laboratory . . . . .	32
3.2	Correct Motor Shield Configuration . . . . .	33
3.3	Circuit Diagram for Simple DC Motor Hardware . . . . .	34

3.4	Correct Motor Shield/ Arduino Connection . . . . .	35
3.5	Mitsumi M25N Motor Wiring Diagram . . . . .	36
3.6	Motor Shield Motor Ports . . . . .	36
3.7	Motor Connected to Motor Shield . . . . .	37
3.8	Female Barrel Jack Power Ports . . . . .	37
3.9	DFRobotics Motor Shield Power Ports . . . . .	37
3.10	Correct Power Connection . . . . .	38
3.11	USB-B Connected . . . . .	38
3.12	supportPackageInstaller in Command Window . . . . .	39
3.13	Install from Internet . . . . .	40
3.14	Arduino Support Checklist . . . . .	40
3.15	Log in to MathWorks Account . . . . .	41
3.16	Type in MathWorks Username and Password . . . . .	41
3.17	License Agreement . . . . .	42
3.18	Third-party Software Licenses . . . . .	42
3.19	Confirm Installation . . . . .	43
3.20	Install/Update Complete . . . . .	43
3.21	Final Simulink Model Used for Simple DC Motor . . . . .	48
3.22	Click Deploy to Hardware . . . . .	48
3.23	Set Target Hardware to Arduino Mega 2560 . . . . .	49
3.24	Set COM Port to Automatic . . . . .	50
3.25	In Solver Pane, Change time to Inf and step size to 0.03 . . . . .	50
3.26	Model Prepared to Run on Arduino Mega . . . . .	51
4.1	Hardware Required for <i>Introduction to 3D Printing</i> Experiment . . . . .	55
4.2	Solidoodle SD4 3D Printer Required for <i>Introduction to 3D Printing</i> Experiment . . . . .	56
4.3	Tape Placed Length-Wise On Printer Surface . . . . .	58

4.4	Tape Placed Length-Wise On Printer Surface . . . . .	59
4.5	Cover Printer Surface With Tape . . . . .	59
4.6	Feed PLA Plastic Into Printer . . . . .	60
4.7	Put Plastic In Nozzle Assembly . . . . .	61
4.8	Click Extrude . . . . .	61
4.9	Extruding Plastic Through the Nozzle . . . . .	62
4.10	Remove Excess Plastic From Nozzle . . . . .	62
4.11	Click Extrude . . . . .	63
4.12	Retracting Plastic From the Nozzle . . . . .	63
4.13	3D View of BedLevel.stl Part . . . . .	64
4.14	Click Configure . . . . .	65
4.15	Print Settings With .3mm Resolution . . . . .	65
4.16	Printer Settings . . . . .	66
4.17	Filament Settings . . . . .	67
4.18	Wingnuts Under Printing Surface . . . . .	68
4.19	Desired vs. Thin Plastic On Plate . . . . .	69
4.20	Motor Load in 3D Viewer . . . . .	70
4.21	Gear Insert in 3D Viewer . . . . .	72
4.22	Correct Object Placement Values . . . . .	72
4.23	Gear Insert in 3D Viewer (Corrected) . . . . .	73
5.1	All Hardware . . . . .	77
5.2	Unplug USB B to A cable from the Arduino . . . . .	77
5.3	Remove Motor Shield from the Arduino . . . . .	78
5.4	Fasten Screws with Phillips Head Screw Driver . . . . .	78
5.5	Attach the 3D Printed Motor Holder to the Arduino . . . . .	79
5.6	Motor Placed Inside Motor Holder With Clamp Pieces Attached . . . . .	79
5.7	Plug Motor Shield Back Into Arduino . . . . .	80



5.8	Completed Case Setup . . . . .	80
5.9	Circuit Diagram for Simple DC Motor Hardware . . . . .	81
5.10	Connecting Power and Ground of Encoder to Arduino (Motor Connector Side) . . . . .	82
5.11	Connecting Power and Ground of Encoder to Arduino (Motor Shield Side) . . . . .	82
5.12	Connecting Encoder A and B ports to pins 18 and 19 on the Arduino(Motor Encoder Side) . . . . .	83
5.13	Connecting Encoder A and B ports to pins 18 and 19 on the Arduino (Motor Shield Side) . . . . .	83
5.14	3D Printed Parts From <i>Introduction to 3D Printing</i> Experiment . . . . .	84
5.15	Pennies Inserted Into Mass (Suspended with Scotch Tape) . . . . .	84
5.16	Opening Notches of Gear Insert With Knife . . . . .	85
5.17	Gear Pushed Into Gear Insert . . . . .	85
5.18	Sanding Gear Insert . . . . .	86
5.19	Gear Insert Inside 3D Printed Mass . . . . .	86
5.20	Gear Pushed into Mass/Gear Insert Combination . . . . .	87
5.21	Completed Load Setup . . . . .	87
5.22	Sticky Tack Rolled Balls . . . . .	88
5.23	Cylindrical-shaped Sticky Tack . . . . .	88
5.24	Sticky Tack On The Bottom of the Case and Motor Encoder . . . . .	89
5.25	Correct Sticky Tack Placement on Motor Encoder . . . . .	89
5.26	Wrap Wires Under Leg of Case . . . . .	90
5.27	Press Each Corner of the Case and the Motor Encoder to Flat Surface . . . . .	91
5.28	Plug USB B Connector Into Arduino . . . . .	92
5.29	Completed Hardware Setup . . . . .	92
5.30	Circuit Diagram for DC Motor . . . . .	93

5.31	Empty Block Diagram for Motor . . . . .	94
5.32	New Script Button Location on Main Matlab Page . . . . .	96
5.33	Save Button Location on m-file Editor Page . . . . .	97
5.34	Sampling Time Variable “Ts” Added to m-file . . . . .	97
5.35	Final Matlab Variables . . . . .	98
5.36	Modified Simulink Model from Simple DC Motor . . . . .	99
5.37	Encoder Output Position to Velocity . . . . .	100
5.38	<b>Serial Send single Port 0</b> and <b>Plot Data 'single'</b> Block and Text Locations, Respectively . . . . .	101
5.39	Final Simulink Diagram to Load on Arduino . . . . .	102
5.40	Function Block Parameters . . . . .	108
5.41	Final Simulink Diagram to be Simulated and Compared With the Ex- perimental Results . . . . .	109
5.42	Add These Two Lines to Code . . . . .	109
5.43	Three Open Loop Step Responses With Full Pennies in Load (Using <b>Km = 18.5 <math>\frac{rad}{Vs}</math></b> and <b>tau = 16.2 s</b> For Simulation) . . . . .	112
5.44	Three 10 Second Open Loop Step Responses With No Pennies in Load (Using <b>Km = 17 <math>\frac{rad}{Vs}</math></b> and <b>tau = 2.7 s</b> For Simulation) . . . . .	113
6.1	Empty Block Diagram for Closed Loop Motor . . . . .	116
6.2	Full Pennies Over-Damped and Under-Damped Simulation Results .	121
6.3	Empty Load Over-Damped and Under-Damped Simulation Results .	121
6.4	Matlab File for Closed Loop Experiment . . . . .	122
6.5	Correct Setup for Second Sum Block . . . . .	125
6.6	Final Simulink Simulation Model Used in the Closed Loop Step Re- sponse Experiment . . . . .	126
6.7	Final Closed Loop Step Response Simulink Model for Arduino . . . . .	132

6.8	Full Pennies Over-Damped and Under-Damped Simulation Vs. Experimental Results . . . . .	139
6.9	Empty Load Over-Damped and Under-Damped Simulation Vs. Experimental Results . . . . .	140
7.1	Hardware Required for Pole Positioning State Feedback Experiment	
	<b>NOTE:</b> All 3D printed materials were printed using .3mm resolution.	142
7.2	Hardware for Encoder Setup . . . . .	144
7.3	Pressing Bearings Into 3D Printed Objects . . . . .	144
7.4	Mount Black Backing of the CUI Encoder Kit to the Encoder Mount	145
7.5	Stick the M4 x 0.70 x 40 screw through the back of the bearing holder	145
7.6	Correct Encoder Configuration . . . . .	146
7.7	Stick the M4 x 0.70 x 40 screw through the Encoder . . . . .	146
7.8	Place the Black Encoder Insert Into the Encoder . . . . .	147
7.9	Press the Gray Insert Into the Black Insert and Ensure Gap is Reasonable	148
7.10	Slide the Encoder Mount Assembly Down (Do Not Clip) . . . . .	148
7.11	Clip the Encoder Black Backing to the Metal Encoder Face . . . . .	149
7.12	Mount the Encoder Mount to the Bearing Holder . . . . .	149
7.13	Final Encoder Assembly . . . . .	150
7.14	Beam Hardware . . . . .	150
7.15	Mount Encoder Assembly to Beam . . . . .	151
7.16	Hardware Used to Attach the Pendulum to the Encoder Screw . . . . .	151
7.17	Place the Aluminum Rod Into the Elbow . . . . .	152
7.18	Hardware Used to Attach the Pendulum to the Encoder Screw . . . . .	152
7.19	Slide the Elbow Onto the Encoder Screw . . . . .	153
7.20	Screw to Clamp the Elbow Bracket to the Encoder Screw . . . . .	154
7.21	Roll the Sticky Tack into a Ball . . . . .	154
7.22	Slide Sticky Tack Ball Onto End of Aluminum Rod . . . . .	155

7.23	Final Pendulum/Beam/Encoder Assembly . . . . .	155
7.24	Hardware for Attaching the Pendulum/Beam/Encoder Assembly to the Motor Shaft . . . . .	156
7.25	Screw Self-Drilling Screw Into the Back of the Beam . . . . .	157
7.26	Beam Attached to Motor Shaft With a Small Gap Between the Top of the Motor and the Beam . . . . .	157
7.27	Hardware for Connecting the Encoder to the Arduino . . . . .	158
7.28	Pin-out and Power and Ground Wire Connections to Encoder . . . . .	159
7.29	Power and Ground Wire Connections to Arduino . . . . .	159
7.30	2 Female Wires Connected to Encoder A and Encoder B Pins . . . . .	160
7.31	Encoder A and Encoder B Wires Connected to Pins 20 and 21 On the Arduino . . . . .	160
7.32	<b>Optional:</b> Scotch Tape Holding Wires to Beam . . . . .	161
7.33	Furuta Pendulum Model (Circled x Denotes Center of Mass) . . . . .	162
7.34	Open Loop Pendulum Response (Nonlinear) . . . . .	166
7.35	Open Loop Beam Response (Nonlinear) . . . . .	166
7.36	Open Loop Pendulum Response (Linear (red) and Nonlinear (blue) Comparison) . . . . .	167
7.37	Open Loop Beam Response (Linear (red) and Nonlinear (blue) Com- parison) . . . . .	167
7.38	Closed Loop Pendulum Response (Linear (red) and Nonlinear (blue) Comparison with a Pendulum Initial Condition of $\frac{\pi}{40}$ ) . . . . .	170
7.39	Closed Loop Beam Response (Linear (red) and Nonlinear (blue) Com- parison with a Pendulum Initial Condition of $\frac{\pi}{40}$ ) . . . . .	171
7.40	Closed Loop Input Voltage Response (Linear (red) and Nonlinear (blue) Comparison with a Pendulum Initial Condition of $\frac{\pi}{40}$ ) . . . . .	171

7.41	Closed Loop Pendulum Response (Linear (red) and Nonlinear (blue) Comparison with a Pendulum Initial Condition of $\frac{\pi}{10}$ ) . . . . .	172
7.42	Closed Loop Beam Response (Linear (red) and Nonlinear (blue) Comparison with a Pendulum Initial Condition of $\frac{\pi}{10}$ ) . . . . .	172
7.43	Closed Loop Input Voltage Response (Linear (red) and Nonlinear (blue) Comparison with a Pendulum Initial Condition of $\frac{\pi}{10}$ ) . . . . .	173
7.44	Closed Loop Pendulum Response (Linear (red) and Nonlinear (blue) Comparison with a Pendulum Initial Condition of $\frac{\pi}{4}$ ) . . . . .	173
7.45	Closed Loop Beam Response (Linear (red) and Nonlinear (blue) Comparison with a Pendulum Initial Condition of $\frac{\pi}{4}$ ) . . . . .	174
7.46	Closed Loop Input Voltage Response (Linear (red) and Nonlinear (blue) Comparison with a Pendulum Initial Condition of $\frac{\pi}{4}$ ) . . . . .	174
7.47	Experimental Results (Pendulum Angle and Angular Velocity) . . . . .	179
7.48	Experimental Results (Base Angle and Angular Velocity) . . . . .	180
7.49	Experimental Results (Input Voltage) . . . . .	180

## CHAPTER 1

### INTRODUCTION

Students often find difficulty in understanding the relationship between control theory and real systems [1]. While simulations often help to bridge the gap between theory and application, simulations are inherently somewhat abstract. To help students better understand this relationship, hands-on experiments are used in many dynamics and control courses, and successfully relate theory with practice for students. However, laboratories offered within these courses at many universities are often limited to a small number of experiments. Additionally, students are often pressed for time in completing the experiments due to limited laboratory resources accommodating large class sizes.

The work provided in this thesis will describe experiments that can be performed at home, and at a time convenient for the students, eliminating the need for laboratory space. In addition, the experiments will not require supervision from teaching assistants (TAs), traditional in most university laboratory settings. This is achieved by providing a website hosting all of the information needed in completing experiments.

The Take Home Labs website [2] was created to host materials for various experiments described in this thesis, and includes links to all experiments, courses, and components. The website also provides experimental templates for those who wish to contribute their own experiments to the website. The range of experiments in this thesis vary between introductory, undergraduate, and graduate levels. Given that others may contribute their own experiments to the website, experiments will also be

able to cover multiple disciplines (controls, signal processing, vibrations, electronics, etc.). In order to host a large number of experiments, it is important that the associated costs are low.

The experiments described in this thesis are inexpensive, costing less than the price of a text book. This is achieved by utilizing resources provided by universities, including MATLAB and Simulink and 3D printing. Also, the use of inexpensive components (including the hardware for each experiment) promotes faster prototyping in experiments with the aforementioned cheaper overhead costs. Rapid prototyping is accomplished by utilizing 3D printers and easy to find off-the-shelf components. The components provided on [2] are from many common websites for the student's convenience. However, students may also find less expensive components from other websites. While take-home laboratories are different than the traditional laboratory paradigm, they are certainly not a completely new concept.

## **Previous Work**

Many types of interesting mechanical systems have been explored for laboratory experiments in automatic control, including but not limited to: Ball on Beam, Magnetically Levitated Sphere, Cart and Pole, Wedge Balancer, Two-Link Pendulum and the Inverted T [3]. For instance, the approaches described in [4] and [3] offer laboratories that work in conjunction with introductory-type courses in control system design. Each experiment is exhibited on a series of workstations in a laboratory hooked to a college network. They cover a multitude of topics, including:

- Step Response and System Poles
- Frequency Response
- State Variable Feedback Effects of Time and Frequency Responses
- Stability and Nyquist

- Root Locus Design
- Computer Simulation

The introductory experiments covered in the laboratory utilize the common theme of a DC motor setup with an encoder. The motor requires minimum maintenance, reducing the amount of hardware upkeep by students, allowing them to focus more on the concepts from each laboratory. Additionally, each paper discusses topics used in most introductory texts, which allows the laboratory follow the curriculums used at many universities. The laboratories are designed to supplement the associated lecture courses. While the laboratories are successful in subjecting students to different types of real control problems, students can only complete experiments during a specified lab time with the computers exclusively on that network.

Another approach to the control lab used in control theory courses is mentioned in [5]. In this survey, there is heavy emphasis placed on using real-time software packages, including: the MATLAB/Simulink Real-Time Toolbox, Real-Time Workshop, and Real-Time Interface. The idea presented in this lab is an open laboratory for students, where students would implement designs in a lab setting. [5] also describes a PC-I/O interface to connect the physical models with the real-time software packages on various models of control systems. However, the survey is general in mentioning that virtually any I/O interface should work with real-time software, which is profound in opening the doors of take-home experiments to many types of micro-controllers.

[1] expounds on the idea of using micro-controllers in conjunction with take-home labs by offering their own version of "take-home kits". The paper mentions that "students all have computers that are suitable for take-home experiments." Additionally, [1] mentions that "laboratory access is often limited", which creates a need to expose students to laboratories for a longer period of time. [1] boasts an increase in exposure to laboratories with the use of a low-cost kit including a board (designed by the



authors in [1]) to handle the PC-I/O interface. The main examples presented in the paper are a DC motor tachometer and temperature measurement system utilizing cheap, off-the-shelf components (including a PIC micro-controller, Windows based user interface program, and cheap sensors). However, with the dependance on the author's own board, development and maintenance is still required and students are limited to that specific board.

To eliminate the dependency on university space, student schedules, and specific hardware interfaces, well documented websites ([6] and [7]) containing modular take-home laboratories are available, so that students may perform experiments in their own homes. [6] offers multiple types of experiments, and also offers documentation for MATLAB and Simulink and National Instrument's Labview in their experiments. The caveat to the experiments offered from [6] is that they are expensive, resulting in high costs for student in addition to their typical university expenses, including tuition, textbooks, and room and board. [7] describes a less expensive prototyping platform that utilizes only MATLAB and Simulink (connected to an Arduino using the Arduino/Simulink interface) with access to resources that help the laboratories run smoothly [7]. However, only one type of platform (a miniature Segway) is explored, and the laboratory documentation offers limited theoretical topics. Neither [6] nor [7] allow outside contributions to their websites, limiting the control problems to only those produced by each site.

## **Innovations**

A growing trend at universities is student access to certain free resources, including MATLAB and Simulink and 3D printing. The work provided in this thesis will utilize these free resources in each experiment. This is different from other control system laboratory approaches since most the laboratories in other works require large overhead costs. Additionally, 3D printing is not considered in any previous works explored

in this thesis.

Another feature of the take-home laboratories described in this thesis is that students will not be limited to control system experiments. A wide range of experiments will be provided in addition to the possible contributed experiments that others can add to the Take Home Labs website. The Take Home Labs website [2] is different from approaches [4], [3], and [5] in that the inexpensive take-home laboratories can be accessed on the internet from anywhere. [2] also varies from [1] since both Raspberry Pi and Arduino (well-documented, and supported circuit boards) may be considered as PC-I/O interfaces, instead of a specific interface board that can only be purchased from the creators of the [1] architecture.

The Take Home Labs website is successful in taking the best ideas from previous the websites [7] and [6], and creating the "best of both worlds." [7] is different from [2] in that it only describes a miniature Segway experiment, where [2] promotes multiple experimental platforms. While [7] is less expensive than [6], the laboratory documentation offers limited theoretical concepts along with the experiments. [6] differs from [2] in that each experiment on the website are expensive. Additionally, each experiment is limited to only the platforms available on the website, and cannot be altered from the pre-determined physical dimensions. Another difference with [6] is that students are limited to exercises offering only control systems. [2] varies from the other websites [7] and [6] in that anyone can contribute experiments. Also, all of the experiments use hardware created from 3D printing, and the experiments are not limited to only control systems. The chapters in this thesis will serve as instructor guides encapsulating the handouts given to students performing the experiments. The Take Home Labs concept was developed jointly in this thesis and in reference [8]. Five experiments are described in this thesis, and five complementary experiments are described in [8]. Together, these documents represent the current state of the Take Home Labs concept.

## Thesis Organization

This thesis will begin with the system overview in Chapter 2, which describes the Take Home Labs website, the software and hardware used in the experiments, the 3D printed motor load (used in Chapters 5 and 6), and the Furuta Pendulum model (used in Chapter 7). Chapter 3, the *Simple DC Motor* experiment, describes an introductory experiment, serving as a “Hello World” lab, common to many computer science courses. Students will learn the basic hardware and software components by interfacing MATLAB/Simulink with an Arduino connected to a DC motor. Chapter 4, the *Introduction to 3D Printing* experiment, describes an additional introductory experiment, which introduces students to the 3D printing software, used to create 3D printed hardware components in more advanced experiments. While the experiment describes only one particular 3D printer, it can easily be modified to accommodate other printers. Chapter 5, the *Open Loop Step Response* experiment, describes an experiment used in an undergraduate “Dynamics or Control” course. Students will find a first-order DC motor model for the physical system using the open loop step response, given motor parameters from the datasheet. Chapter 6, the *Closed Loop Step Response* experiment, describes an experiment that adds feedback to the Open Loop Step Response experiment. Students will find the open loop poles and then, given gains, observe the effects of a proportional-derivative feedback controller. Students will both simulate the model using Simulink and observe the effects from the physical system. Chapter 7, the *Pole Positioning State Feedback* experiment, describes an experiment that uses pole positioning on the Furuta pendulum (also known as the Rotary Inverted pendulum) system. Students are tasked with creating their own Simulink model based on given equations of motion and then told to iteratively find gains that control the pendulum system. Subsequently, the gains will be applied using full state feedback on the physical system. Chapter 8 will provide a conclusion for this thesis and insight into future work.

## CHAPTER 2

### OVERALL FRAMEWORK

The opportunity for students to take home inexpensive labs without the need for a university laboratory or teaching assistants is the main concept presented in this thesis. Additionally, the Take Home Labs website was created in order host a number of experiments that students may easily access and complete at home. Take home labs reduce the need for university resources, the impact on student schedules, and the cost to students. This is achieved by providing simple hardware designs, and the use of well-known software provided by universities (such as Matlab and Simulink). This chapter will provide the system overview of the Take Home Labs website, and the experiments presented in the body of this thesis, including all of the components used to create them. After that, the 3D printed motor load and the Furuta pendulum system (also known as the Rotary Inverted Pendulum), which form the basis for several experiments, will be explained. The overall structure of the take home labs consists of the following:

- **Website**

The Take Home Labs website ([thl.okstate.edu](http://thl.okstate.edu)) is the main source of information for experiments. Student can easily access the website from anywhere an internet connection is available. The website consists of a simple interface, with links to all experiments, handouts, and lists of materials. 3D printed .stl files are also available on the website.

- **Experiment Handouts**

The experiment handouts are the main source of information given to students performing experiments. The handouts can be found on the website and will list all necessary materials (hardware, software, and prerequisite experiments) needed to run the experiments correctly. Not included in the handouts are the instructor sections, which provide the results for each experiment for the instructors.

- **Hardware**

Hardware used in each experiment is placed together to form a physical structure, and varies depending on the experiment. Each experiment lists all necessary hardware. Some hardware may be purchased from external websites. Other hardware may be created from a 3D printer. All hardware is specified in the experiment handout, and the website lists links to the appropriate sources of hardware.

- **Software** Software used in the labs generally varies for each experiment. Each handout lists the software necessary to run the experiments. Additionally, a software setup portion in the handout lists the steps needed to set up the software.

## 2.1 Website

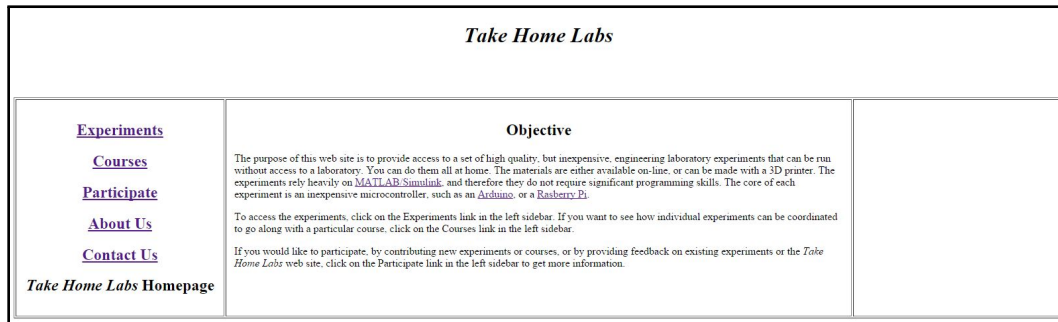


Figure 2.1: Take Home Labs Homepage

The Take Home Labs website provides two options to search for experiments. On the Take Home Labs Homepage, the links **Experiments** and **Courses** list out all of the links for experiments based on certain criteria. For example, experiments listed under **Experiments** will be identified in a more general category. The **Courses** link will list all of the course names, where each course name link will list links to every experiment that pertains to that course.

### 2.1.1 Experiments

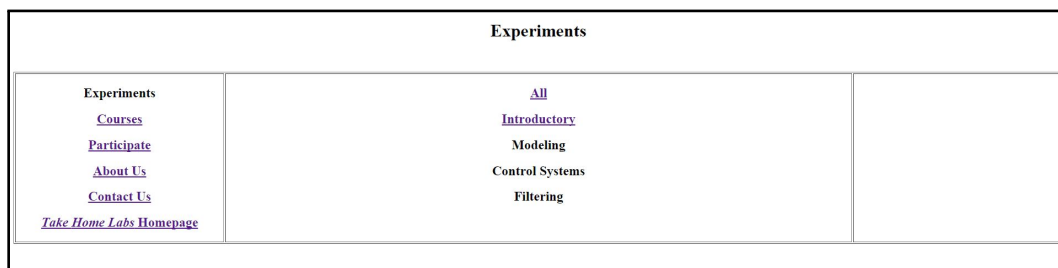


Figure 2.2: Take Home Labs Experiments Page

Under the **Experiments** link, experiments are listed as the following categories: All, Introductory, Modeling, Control Systems, and Filtering. (It is expected that more categories will be added.) Clicking on **All** lists all of the available experiments. The other categories are explained in the descriptions below.

## **Introductory**

Introductory experiments provide initial exposure of students to the hardware and software used in later experiments. These labs require no prerequisite experiments, as they are the most basic level labs. Introductory experiments serve as “Hello World” experiments, common to many computer science courses.

## **Modeling**

Modeling experiments encapsulate the concept of modeling systems. For example, System Identification would be considered in this category, since it is one concept used in finding a system model. These types of experiments will vary, and serve as intermediate level experiments used in modeling systems for other experiments.

## **Control Systems**

Control system experiments typically explore feedback control systems. Control System experiments include, but are not limited to, the following: PID control, Lead-Lag Compensators, Optimal control, Adaptive Control, and Digital Control. These experiments will be presented in more advanced courses in graduate school settings.

## **Filtering**

Filtering experiments consist of labs that focus on digital filtering and signal processing. Labs under this category will function more as signals and systems type experiments, rather than control systems. (Although, some control system experiments may contain concepts from these experiments) Experiments listed under this category represent the breadth of subjects which the take-home labs can cover, since take home labs are not limited to only control systems.

## 2.1.2 Courses

Courses		
<a href="#">Experiments</a>		
<a href="#">Courses</a>		
<a href="#">Participate</a>		
<a href="#">About Us</a>		
<a href="#">Contact Us</a>		
<a href="#">Take Home Labs Homepage</a>		
	Adaptive Control	
	Automatic Control Systems	
	Digital Control Systems	
	Digital Signal Processing	
	Estimation Theory	
	Neural Networks	
	Optimal Control	
	Signals and Systems	
	<a href="#">System Dynamics</a>	
	System Identification	
	Vibrations	

Figure 2.3: Take Home Labs Courses Page

Clicking on the **Courses** link on the Take Home Labs homepage will separate experiments into categories that pertain to different types of courses, including, but not limited to: Adaptive Control, Automatic Control Systems, Digital Control Systems, Digital Signal Processing, Estimation Theory, Neural Networks, Optimal Control, Signals and Systems, System Dynamics, and System Identification. However, this section will focus on system dynamics and optimal control categories since they are explained in more detail later in this thesis. Additionally, courses will be added to this list, since additional experiments will be contributed to the website.

### System Dynamics

System dynamics courses consist of the following examples: Simple DC Motor, Sampling and Data Acquisition, Open Loop Step Response, Closed Loop Step Response, Open Loop Frequency Response, Closed Loop Frequency Response, and PD Position Control Design. The experiments listed in this category will describe dynamics and control courses available at most universities.



## Optimal Control

Optimal Control experiments will serve as more advanced experiments for students who are in graduate school optimal control courses. The topics for these experiments would include pole positioning state feedback control, optimal state feedback control, and optimal output feedback control. These experiments would typically use more complex physical systems, like types of inverted pendulum.

### 2.1.3 Participate

Participate		
<p><a href="#">Experiments</a></p> <p><a href="#">Courses</a></p> <p><a href="#">Participate</a></p> <p><a href="#">About Us</a></p> <p><a href="#">Contact Us</a></p> <p><a href="#">Take Home Labs Homepage</a></p>	<p>We welcome everyone to participate in the <i>Take Home Labs</i> website. You can do that by submitting new experiments, submitting corrections to existing experiments or by providing general comments on the site.</p> <p>The guiding principles of the <i>Take Home Labs</i> website are the following: 1) The experiments can be done at home with a minimum of equipment. They do not require scopes, signal generators, spectrum analyzers, or other devices that would not normally be found in a dorm room. 2) The materials-parts are inexpensive and are conveniently available online, or can be made with a 3-D printer (printer files provided). The materials required to run all of the experiments for a given course cost no more than the price of a textbook. 3) Since the experiments are done at home, without the assistance of a TA, the authors make every effort to produce experiment handouts that are clear and complete. Prerequisite experiments are indicated in the lab handout and always include at least one of the introductory experiments. Background material required of the student is limited to theoretical topics relevant to the experiment (e.g., what a transfer function is), and not hardware/software technicalities (e.g., how an Arduino is interfaced to Simulink). 4) The heart of most experiments is an inexpensive microcontroller, such as the <i>Arduino</i> or the <i>Espressino</i>. 5) Many experiments take advantage of Simulink blocks for programming the microcontroller, to minimize the amount of programming required by the student. For experiments in which programming is the key topic, this is not the case.</p> <p>If you would like to submit a new experiment for the site, please email the following four items to the email address shown in the Contact Us link in the left sidebar. (If your files make up more than 10 MB, contact us before sending the files.)</p> <ul style="list-style-type: none"><li>• LaTeX and PDF versions of the experiment handout (along with any figure files, etc.) zipped into one file. (A LaTeX template for the experiment handout can be downloaded through the link in the right sidebar. Also, look at other experiment handouts for example formats.)</li><li>• List of hardware parts needed to run the experiment, with links to online sites to order the parts. (See the Hardware Parts links on some of the experiment pages to see examples of this information.) The experiment handout should not contain any active links. The links will be located on the Hardware Parts web page.</li><li>• If necessary, any code, including Simulink files, that are needed to run the experiment (zipped into one file).</li><li>• If necessary, any 3-D printer files for parts required to run the experiment (zipped into one file).</li></ul> <p>The contributor of each experiment will be recognized on the experiment web page and within the experiment handout.</p> <p>To submit files, corrections or comments, use the email address in the Contact Us link in the left sidebar.</p> <p style="text-align: center;"><b>Thanks in advance for your help.</b></p>	<p><a href="#">LaTeX Handout Template</a></p>

Figure 2.4: Take Home Labs Participate Page

On the **Participate** page, others may contribute experiments to the website. In order to maintain the structure of the take home labs, certain guidelines are explained on the Participate page. Contributors to the website will be recognized on their contributed experiment's page and within the experiment handout.

## Contributions

Contributions may come in the form of submission of new experiments, feedback on current experiments, or general comments about the website. These types of contributions will be invaluable to the take home labs, given that any sort of feedback provides perspectives on the experiments that were not originally explored or considered. This allows the labs to reach more people, and, in turn, benefit others. Other

guidelines presented are the following:

- Experiments must be able to be done at home with minimum equipment.
- Scopes, signal generators, spectrum analyzers, or any other devices will not be required.
- Materials/parts are inexpensive and readily available online, or can be created with a 3-D printer (with 3D printer files provided). They must also be cheaper than the price of a text book.
- Clarity of the experiments is important, since no TA guidance will be available. Prerequisite experiments must be clearly indicated, and must include at least one of the introductory experiments. Background material for the experiment must be limited to topics relevant to the experiment, and not hardware/software related.
- Inexpensive microcontrollers, such as Arduino or Raspberry Pi, will be the heart of most experiments.
- The use of Simulink Blocks to program the microcontroller is encouraged in order to reduce the amount of programming by the student. The exception to this would be experiments that are focused on programming topics.

### **L<sup>A</sup>T<sub>E</sub>X Handout Template**

Handouts for each contributed experiment are required. The following guidelines are laid out on the website:

- Handouts must be submitted in LaTeX form, and placed into a zip folder along with all additional files (e.g. figures/plots). Files less than 10MB are preferred. A LaTeX template for experiment handouts is provided on the Participate page.

- A list of hardware/parts needed to run the experiments (including links to order the parts) will be required with each submission. However, the experiment handout must exclude any active links, leaving only the list of parts listed.
- If necessary, code (including Simulink files) needed to run the experiment must be zipped into one file and submitted along with each handout.
- If necessary, 3D printer files for parts included in the experiment must be zipped into one file and submitted along with each handout.

## **2.2 Selection of Hardware Used In the Labs**

The selection of common hardware used in creating experiment designs was a lengthy process that included many iterations. While the components used in the final design now achieve their intended functions, any foresight of their applicability in the final design was not initially obvious. This section will first explain previous components that were considered along the way. Next, the design considerations for the common components that were used in the final design will be explained.

### **2.2.1 Previously Considered Common Components**

Many components were considered along the way in the design of the final take home labs. Among these, the motor shield and power supply were the components that changed the most. Most of the experiments involve the use of a DC motor. The motor shield supplies the voltage to the motor and the power supply powers the motor shield. The subsections below will explain the component and design considerations that went into removing it from the final design.

## Motor Shield: SN754410



Figure 2.5: First Considered Motor Shield (SN754410)

The SN754410 motor shield is made by Texas Instruments, a well known American electronics company that creates many different types of IC chips. Aside from being small and inexpensive (costing only a couple of dollars), this particular motor shield has the following characteristics:

- Voltage range of 4.5V to 36V
- 2A maximum output current
- Works with DC Motors

This chip was considered because the voltage and current ranges overlapped the ranges of the motor being used at the time (see the section labeled **Mitsumi Motor Encoder** below for more information on this motor). However, since this chip requires additional hardware (including wires and a breadboard) in order for it to run with the Arduino correctly, it was desirable to find another alternative that required no wires and could plug directly into the Arduino microcontroller. Eventually this component was scraped when another option, with the same specs, was found and could plug into the Arduino.

## Power Supply: (S-50-24)

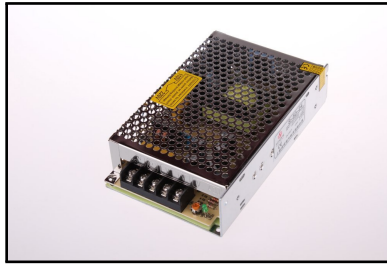


Figure 2.6: First Considered Power Supply (S-50-24)

The S-50-24 power supply was originally considered because of the following qualities:

- Inexpensive
- Voltage output of 24V
- Max current output of 2A

While the power supply performed well and was within the voltage and current requirements of the motor, it was more desirable to find a power supply with more voltage options. Additionally, it was desirable to find a power supply of a laptop AC power adapter form factor. Eventually this component was scrapped, and an AC power adapter with the laptop form factor and large voltage range was found.

## 2.2.2 Final Common Components

### Motor Shield: DFRobot

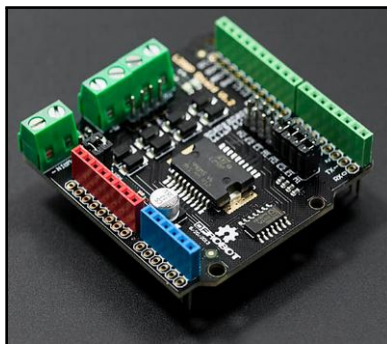


Figure 2.7: Final Motor Shield (DFRobot L298P)

The DFRobot motor shield (see Figure 2.7) was the final choice for the motor shield used in most experiments that require an interface with the Arduino. Since it can easily be plugged into the Arduino, and function without the use of additional wires, the motor shield was a good fit for the experiments. Additionally, the voltage and current ratings were identical to the motor shield described in the **Motor Shield: SN754410** section, which caused no drastic changes in function.

### Power Supply



Figure 2.8: Final Power Supply (Universal AC Adapter)

The Universal AC Adapter fit all of the necessary requirements for power to the motor shield. Additionally it varies from the following voltages: 15V, 16V, 18.5V, 19.5V, 20V, 22V, and 24V. With a power rating of 70W a large range of currents are also available and reach well within the 2A limit (with the highest current being 4.67A and the lowest current being 2.92). However, the power that goes to the motor is ultimately limited to the 2A rating on the motor shield, since all of the maximum currents draws are larger than 2A.

## Microcontroller

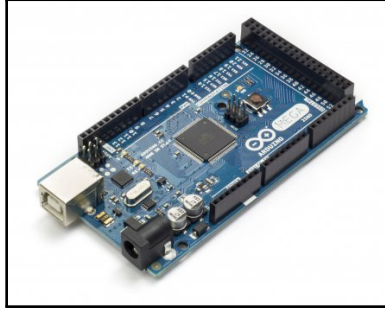


Figure 2.9: Final Microcontroller (Arduino Mega 2560)

The Arduino Mega 2560 (see Figure 2.9) was chosen among a number of Arduino boards that are compatible with MATLAB/Simulink. This Arduino was chosen over other Arduino models due to the larger number of pins that are available. These additional pins, especially during the prototyping phase, were invaluable for including additional peripherals (such as encoders and motor shields). Additionally, the Arduino comes with 256KB of onboard flash memory, which is larger than any other Arduino boards compatible with MATLAB/Simulink. Extra memory was favorable since the sizes of programs can vary and take up more space with added complexity of experiments.

## Motor/Encoder

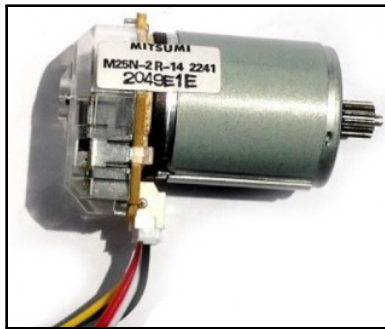


Figure 2.10: Final Motor/Encoder (Mitsumi M25N-2R-14)

The challenge of finding a motor/encoder combo was difficult, since most motor/encoders are either expensive, or have low encoder resolutions. The Mitsumi M25N-2R-14 (see

Figure 2.10) was found with the following characteristics:

- Maximum voltage of 34V
- No load motor speed of 11,000 rpm
- No load current of 100mA or less
- Encoder resolution of 1336 counts/revolution
- Inexpensive (found as low as \$7)

Given the high motor speed, range of voltages, high encoder resolution, and low cost, this motor has proven to be very valuable for the take home labs. The only disadvantage to this motor has been availability. The motor is used in a printer and is manufactured in China, rendering it potentially out of stock in the future. However, as far as function, it works well with all of the experiments that require a motor/encoder.

## Encoder

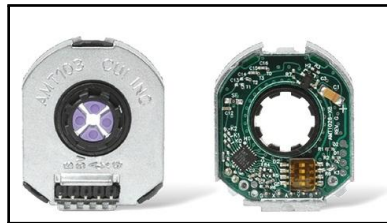


Figure 2.11: Final Pendulum Encoder (CUI AMT103)

The CUI AMT103 encoder functions as a rotary encoder that may be used for experiments that require additional encoders, without a motor attached. With a maximum resolution of 4096 counts/revolution, this encoder is very accurate and works well with many applications. Additionally, it is inexpensive compared with other encoders (costing of around \$25), which keeps the total experiment cost low. In this thesis, the Furuta pendulum problem utilized this encoder for measuring the pendulum angle.



### 2.2.3 3D Printer (Solidoodle SD4)

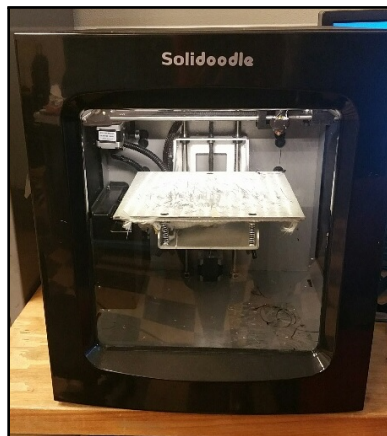


Figure 2.12: Solidoodle 3D Printer (SD4)

While the 3D printer itself was not a component used in experiments, the Solidoodle SD4 (see Figure 2.12) was used to create many different components. Since 3D printing material is cheap (in this thesis PLA plastic was used) this printer was valuable in creating additional components, while keep experiment costs low. In this thesis, a number of components were created for the experiments described in sections 2.4 and 2.5 below.

## 2.3 Selection of Software

While the software selections for the experiments required less considerations than the hardware, the software is considered the “heart” of each experiment. Without software, the hardware is useless in allowing the experiments to function. The subsections below describe the software that was used in all of the experiments.

### 2.3.1 MATLAB/Simulink

MATLAB and Simulink were used the most in allowing the experiments to function. MATLAB is generally used in creating m-files that set parameters into the workspace, a variable environment that shows all variable with their stored values after run time. Simulink files consist of blocks that interconnect with one another creating a “visual

coding” platform. Simulink is useful in limiting the amount of coding that students actually perform during an experiment. Requiring less code allows students to focus more on the underlying concepts without the burden of syntax.

### 2.3.2 Repetier Host

Repetier Host is an open-source software that is used with Solidoodle 3D printers. Since the software is open source, there are no costs associated with this software. Additionally, the software is quite user-friendly and requires a small learning curve, unlike many other software packages.

## 2.4 3D Printed Motor Load (Pennies)

This section describes the 3D Printed Motor Load that is used in the following experiments: *Open Loop Step Response*, *Closed Loop Step Response*, *Open Loop Frequency Response*, and *Closed Loop Step Response*.

### 2.4.1 Description

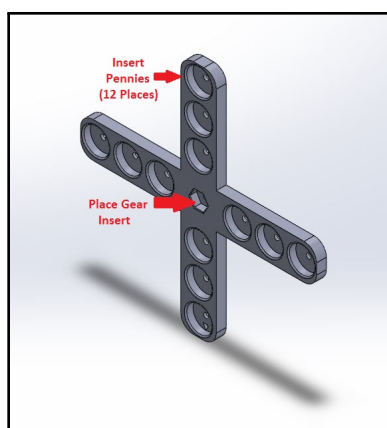
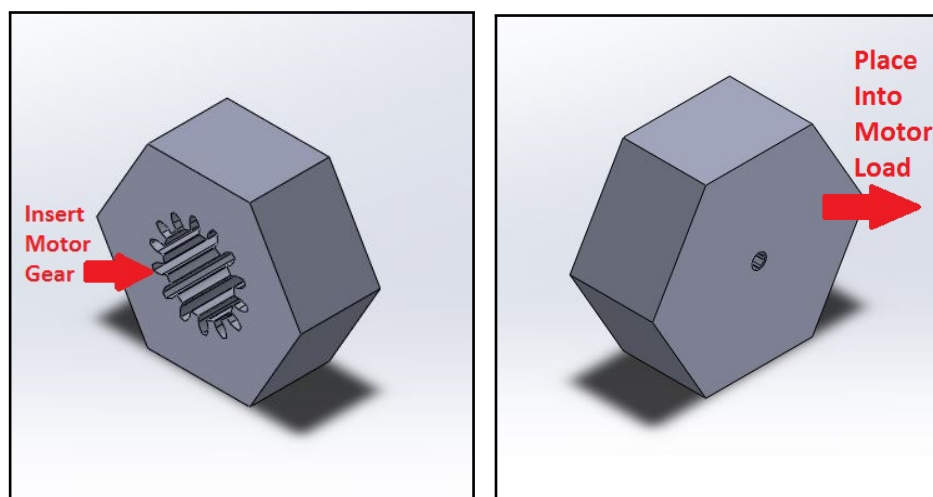


Figure 2.13: 3D Printed Motor Load

The 3D Printed Motor Load was created as a load to be used in dynamics and system courses that require a load attached to a motor. Additionally, there was a need for a load to be created that could be different for each student in the course. Since

the motor load needs to be inexpensive, and all students have access to pennies, the experiments are kept inexpensive and add enough mass to the overall load to change the open and closed loop step and frequency responses significantly. The 3D printed motor load achieves this function with 12 penny sized slots along the arms that can fit up to 4 pennies per slot. Assuming that the center of the mass will need to be centered around the pivot of the motor, and class sizes are generally around 30 students, the mass can easily be configured to create different inertias for each student in the course. See Figure 2.13 for an example load.

### 2.4.2 Insert



(a) 3D Printed Motor Load Insert (Front View) (b) 3D Printed Motor Load Insert (Rear View)

Figure 2.14: 3D Printed Motor Load Insert

In the center of the 3D printed motor load is a 3D printed motor insert. The insert was designed to easily fit onto the gear of the Mitsumi motor/encoder gear, as well as fit into the 3D printed motor load. The insert acts as a medium between the motor and the motor load. Additionally, if the insert becomes too worn down, printing another one is cheap and takes approximately 10 minutes of print time at

high resolution. See Figure 2.14 for an example insert.

## **2.5 Furuta Pendulum (Pole Positioning State Feedback)**

This section describes the Furuta Pendulum (also known as the rotational inverted pendulum) that is controlled using pole positioning state feedback. The Furuta pendulum consists of a motor connected to a beam that spins parallel with the ground. Attached to the end of the beam is a pendulum that can hang freely. The objective of the Furuta pendulum is to balance the pendulum straight up using the motor connected to the beam. The following subsections will explain the equations of motion and the states chosen for the linear state-space model, the development for the physical system on hardware, and the final physical system. The Furuta pendulum can be used for many types of control experiments. A later chapter in this thesis will describe an experiment using state feedback on the Furuta pendulum.

## 2.5.1 Model

### Equations of Motion

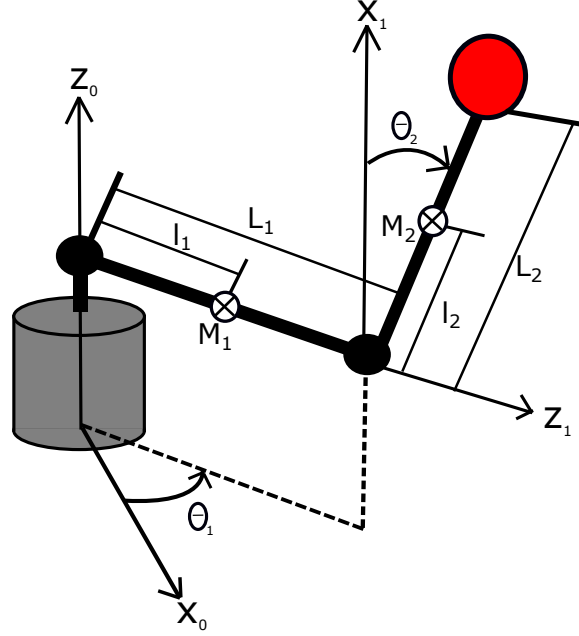


Figure 2.15: Furuta Pendulum Model (Circled x Denotes Center of Mass)

The following equations of motion describe the dynamics of the Furuta Pendulum model:

$$\begin{aligned}\ddot{\theta}_1 &= \frac{C_2 L_1 \cos(\theta_2) \dot{\theta}_2}{l_2 B} + \frac{M_2 L_1 l_2 \sin(\theta_2) \dot{\theta}_2^2}{B} + \frac{\tau}{B} - \frac{M_2 g L_1 \cos(\theta_2) \sin(\theta_2)}{B} \\ &\quad - \frac{C_1 \dot{\theta}_1}{B} - \frac{M_2 l_2^2 \cos(\theta_2) \sin(\theta_2) \dot{\theta}_1 \dot{\theta}_2}{B} \\ \ddot{\theta}_2 &= \frac{-L_1 \cos(\theta_2) \ddot{\theta}_1}{l_2} - \frac{C_2 \dot{\theta}_2}{M_2 l_2^2} + \sin(\theta_2) \cos(\theta_2) \dot{\theta}_1^2 + \frac{g \sin(\theta_2)}{l_2} \\ B &= -M_2 L_1^2 \cos^2(\theta_2) + M_1 l_1^2 + M_2 L_1^2 + M_2 l_2^2 \sin^2(\theta_2) \\ \tau &= [u - N K_m \dot{\theta}_1] \frac{K_t}{R_a}\end{aligned}$$

Where

$\theta_1$  : Angular Position of Base Arm

$\theta_2$  : Angular Position of Pendulum

$M_1$	: Mass of Base Arm
$M_2$	: Mass of Pendulum
$L_1$	: Length of Base Arm
$L_2$	: Length of Pendulum
$l_1$	: Length from Pivot to Center of Mass of Base Arm
$l_2$	: Length from Pivot to Center of Mass of Pendulum
$C_1$	: Viscous Friction Coefficient of Motor Pivot
$C_2$	: Viscous Friction Coefficient of Pendulum Pivot
$g$	: Gravitational Constant
$\tau$	: Torque
$u$	: Motor Voltage
$R_a$	: Armature Resistance of Motor
$K_t$	: Torque Constant of Motor
$K_m$	: Back EMF Constant of Motor
$N$	: Gear Ratio of Motor

## States

The States for this model are:

$$x_1 = \theta_1$$

$$x_2 = \dot{\theta}_1$$

$$x_3 = \theta_2$$

$$x_4 = \dot{\theta}_2$$

## 2.5.2 Development of Hardware

Developing the Furuta Pendulum system was not an obvious task from the start. It required many different iterations of design considerations. The subsections below describe the hardware revisions that were considered before developing a final design.

### First Revision

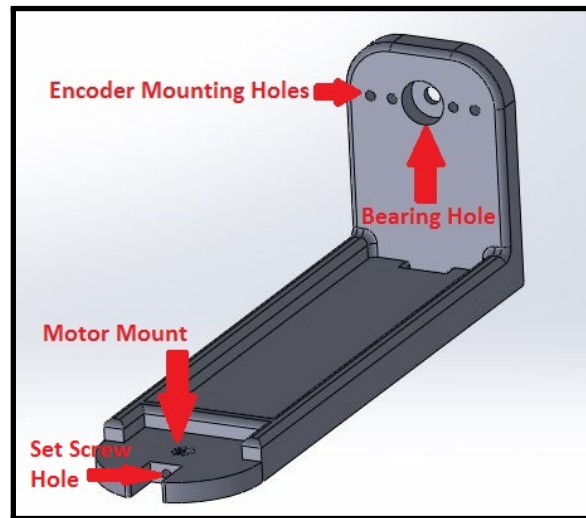


Figure 2.16: Furuta Pendulum Model (First Revision)

The first design revision consisted of one piece that was attached to the motor. At one end, the beam attached to the motor, and the other end housed a bearing and functioned as the encoder mount, where the encoder measured the angle of the inverted pendulum. The bearing was used to reduce the friction of the pendulum at the pivot. A set screw was used instead of a 3D printed insert to mount the motor, due to the weight of the pendulum. Set screws hold the beam to the motor and keep the assembly together stronger than a 3D printed insert, since the friction of attaching and removing the insert would eventually wear it down. The encoder mount was used to hold the encoder so that it was fixed to the end of the beam. This design was eventually changed because the 3D printer had difficulty in printing the beam as one part. Since the end where the encoder mounted extruded upwards and was a

thinner piece, the 3D printer took a long time to print and the encoder mount had questionable integrity at the end of the print (it broke off in some cases).

## Second Revision

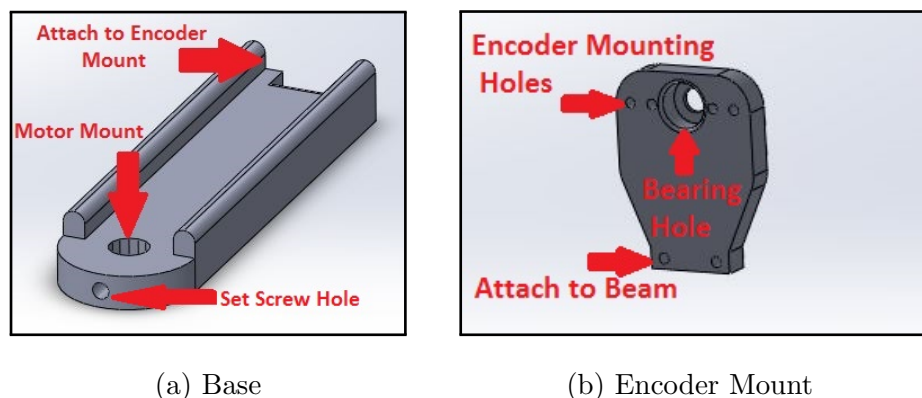


Figure 2.17: Furuta Pendulum Model (Second Revision)

The second revision of the Furuta Pendulum split the hardware from the first revision into two pieces, one for the beam that attaches to the motor, and the other to mount the encoder to. Both pieces connected to one another with screws to create one large piece. The biggest differences between the second revision and the first revision is that the second model was smaller to reduce the weight on the motor pivot, and the model consisted of two separate pieces instead of one large piece. The model was split into two pieces because the 3D printer had difficulty printing the large piece in a timely manner. The two-piece methodology was found to make 3D printing much faster, and was carried into the final design.



### 2.5.3 Final Hardware Design

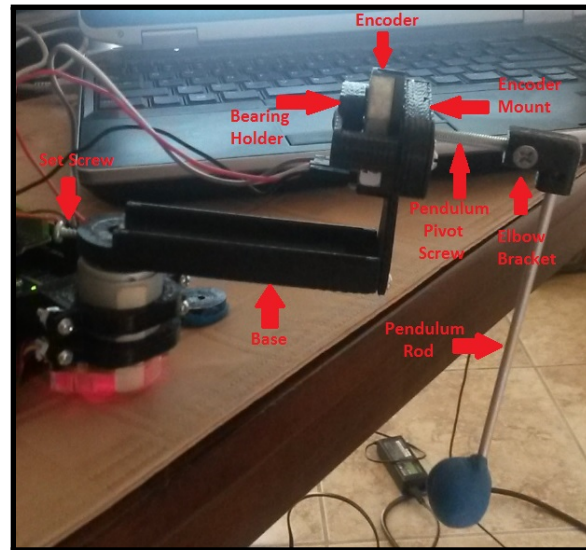


Figure 2.18: Furuta Pendulum Model (First Revision)

After the design process listed in the section above, the final revision of the Furuta Pendulum 3D printed hardware was found. The Elbow Bracket and Pendulum Rod hardware did not change throughout the design process for the Furuta Pendulum.

#### Base

The final base used in the final hardware was the same as the one created in the second revision. This design decision was made based on the small size of the second revision, and the time it took to print the part (which is roughly 20 minutes).

#### Encoder Holder

The final encoder became a smaller version of the second revision, since there was unneeded 3D printing material in the second revision. Material was removed mostly from the middle portion of the encoder mount and kept where the bearing and encoder were mounted.

## **Bearing Holder**

An addition to the design was the bearing holder, which adds an additional bearing to where the pendulum is attached to the encoder. This additional bearing was added to further remove friction from the pendulum pivot. Without the additional bearing, the pendulum screw, which attaches the beam/encoder mount assembly to the pendulum, would press upwards on the encoder due to the weight of the pendulum. When this occurred, the encoder would seize up (causing the pendulum to stay straight up, and not move) due to the slots on the encoder rubbing against the internal walls of the encoder. The additional bearing and bearing holder allows the pendulum screw to stay straight, without adding unwanted friction to the system.

## **Elbow Bracket**

The elbow bracket was never changed from the original design. This part functions as the link between the pendulum and pendulum screw, and allows the twisting motion of the pendulum pivot screw to translate to the rotation of the pendulum rod, by placing the rod tangent to the pendulum pivot. Without the elbow bracket, the pendulum would not be able to be mounted to the pendulum pivot screw.

## **Pendulum Rod**

The pendulum rod, another hardware component that did not change with design revisions, is the portion of the system that functions as the free-hanging pendulum. While the mass of the pendulum is not very large, mass is easily added in the form of sticky tack to the tip of the pendulum, allowing it to overcome the friction of the pendulum pivot more easily. The function of the whole system is to keep the pendulum rod standing straight up.

## 2.6 Summary

Chapters 3 - 8 will describe how the hardware and software mentioned in this chapter will be used in the experiments. Chapter 3 will describe an introductory experiment that sets up the MATLAB/Simulink software described in section 2.3.1 to be used with the DC motor, Arduino, and motor shield described in 2.2.2. Chapter 4 describes an introductory experiment using the 3D printer described in section 2.3.2 and the Repetier Host software in section 2.2.3. Chapters and 5 and 6 will use the hardware described in section 2.4. Chapter 7 will use the hardware described for the Furuta pendulum in section 2.5. The following chapters will be extensions to the experiment handouts, in that they will function as the instructor guides to the experiments by adding the results from each exercise to the handout.

## CHAPTER 3

### Simple DC Motor

#### 3.1 Objective

This chapter describes a typical introductory lab experiment. This lab serves as the “Hello World” lab, common to most computer science courses. The objective is to introduce students to the basic hardware and software components that will be used in later experiments. Since students will be performing these experiments at home, without assistance from TAs, it is important that any introductory experiment, like the one described here, be very clear, and that it not require any previous experience with the hardware and software. The introductory experiments should help the students build their confidence.

#### 3.2 Setup

This section of the report describes all of the required materials needed to make the lab work correctly. This is achieved by splitting the setups into two subsections, hardware and software. The hardware subsection provides all necessary, physical materials the student needs along with how they are physically connected. The software subsection provides information and steps to download all of the software needed to run the lab with the hardware.

### 3.2.1 Required

This section lists all of the software and hardware materials that are needed to perform this experiment. It also lists any prior experiments that should be performed before running this experiment.

#### Hardware

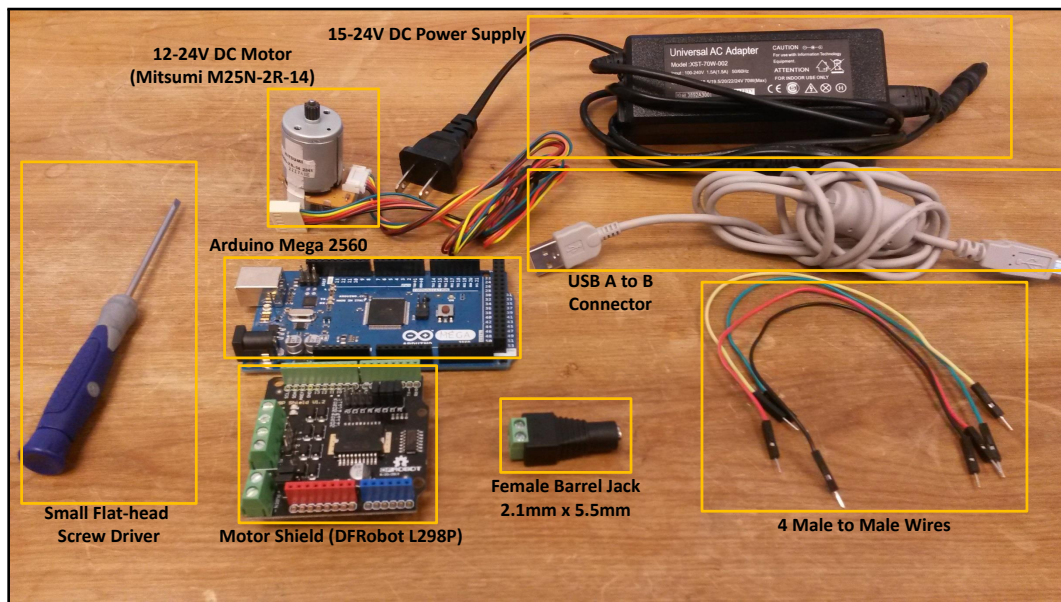


Figure 3.1: Hardware Required for Laboratory

- DC Motor (Mitsumi M25N-2R-14)
- Microcontroller (Arduino Mega 2560)
- Motor Shield (DFRobot L298P)
- 15-24V Battery or Power Supply (Universal AC Adapter with 2.1mm x 5.5mm Male Connector)
- USB B to A Converter Cable (USB 2.0 A-Male to B-Male Cable)
- 4 Wires (20cm Male To Male Jumper Wire)
- Female Barrel Jack (2.1mm x 5.5mm Female CCTV Power Jack Adapter)

## Software

- Matlab/Simulink 2014b
- Windows 7

## Previous Experiments

- This is an introductory experiment and does not require that any other experiments be performed first

### 3.2.2 Hardware Setup

This section describes the hardware design of the DC motor lab. The DC motor used in this lab will be utilized in later labs, and this lab will develop a familiarity with how the motor is interfaced to Simulink. A circuit diagram for the DC motor will be included as well as a photo of the circuit connection.

### Configuring Motor Shield

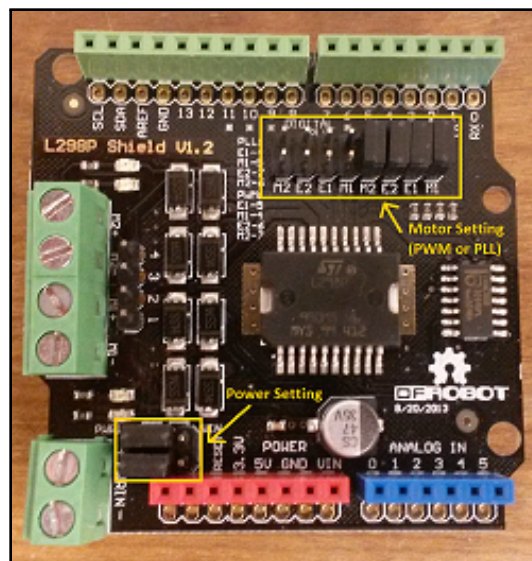


Figure 3.2: Correct Motor Shield Configuration

1. Jumper the pins of the DFRobot motor shield as a PWM input with external power (power outside of the Arduino Mega). Figure 3.2 shows the correct setup of the motor shield with jumpers on the right four sets of pins (labeled as M2, E2, E1, and M1 from left to right). These jumpers ensure that the motor shield is configured to accept PWM signals to drive the DC motor.
2. Additionally, the motor shield has 6 pins in the bottom left-hand corner of the board, which are also shown in Figure 3.2. Jumper the pins to accept an external power supply in order to power the DC Motor. This requires that the left four pins out of the six pins are jumpered, with the top-left two pins jumpered together and the bottom-left two pins jumpered together.

### Connecting Hardware

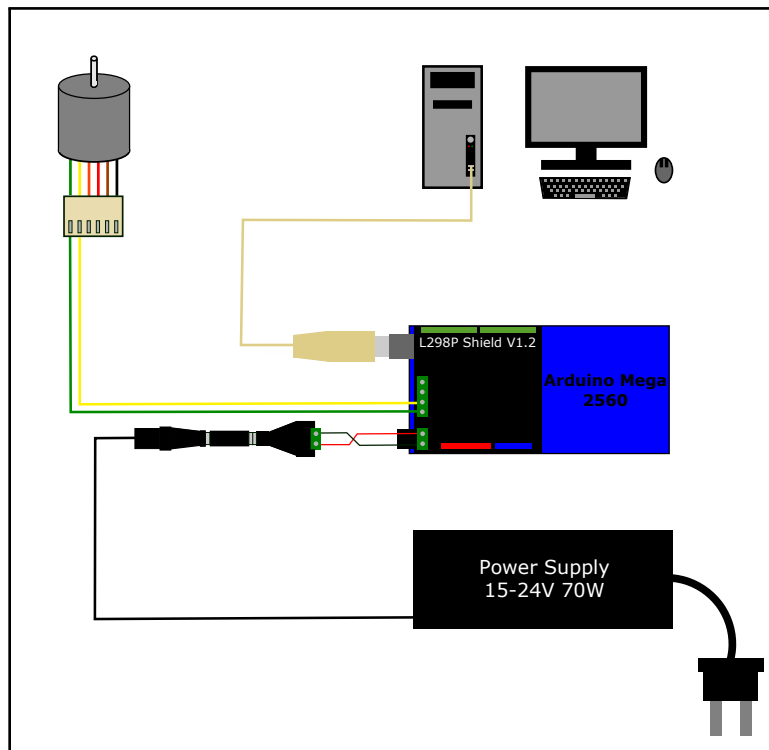


Figure 3.3: Circuit Diagram for Simple DC Motor Hardware

1. Check to see that all hardware shown in Figure 3.1 is available.

2. Take the motor shield and plug into the Arduino board as seen in Figure 3.4. Make sure that the pin labeled 5 on the motor shield is plugged into the A5 pin on the Arduino. Additionally, make sure that the Rx pin lines up with the RX 0 pin on the Arduino. This will ensure that the motor shield is connected properly to the Arduino.

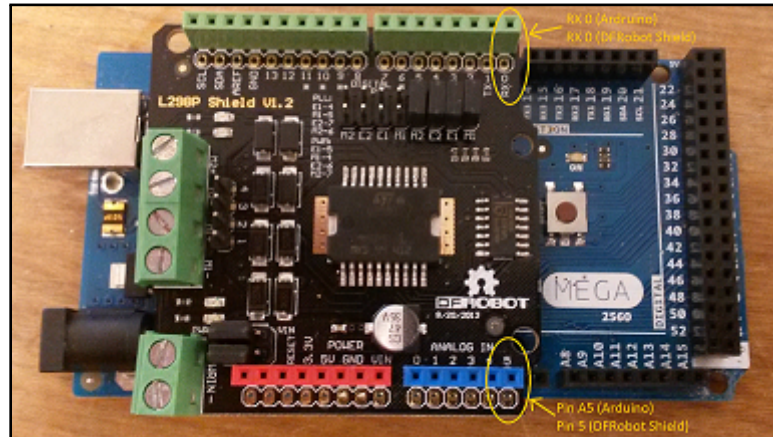


Figure 3.4: Correct Motor Shield/ Arduino Connection

3. Next, connect the “M1 +” and “M1 -” of the motor shield (see Figure 3.6) to the corresponding “M+” and “M-” ports of the Mitsumi Motor (see Figure 3.5) using two male to male wires. Refer to Figure 3.7 for the final correct DC motor connection with the motor shield.



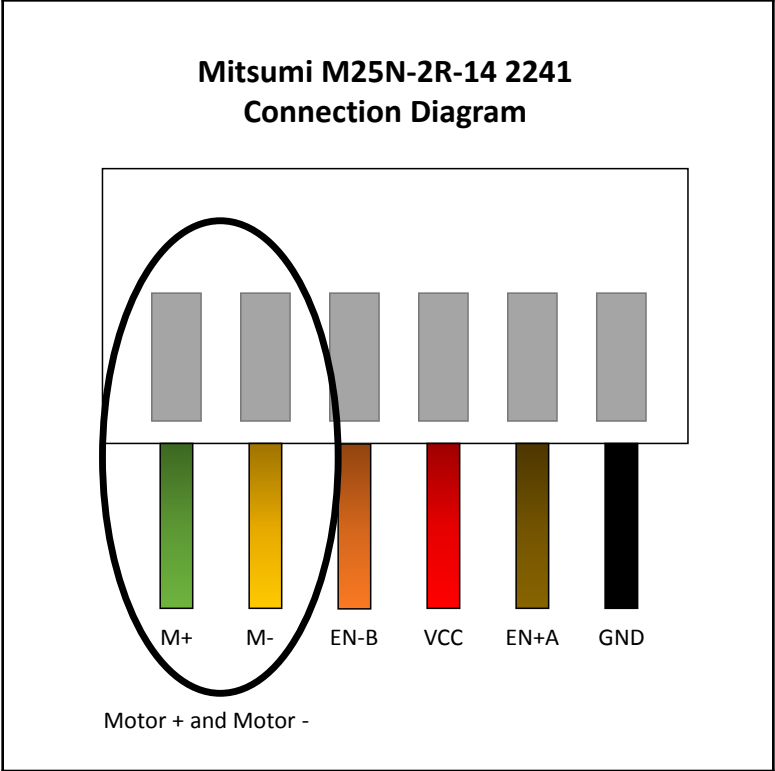


Figure 3.5: Mitsumi M25N Motor Wiring Diagram

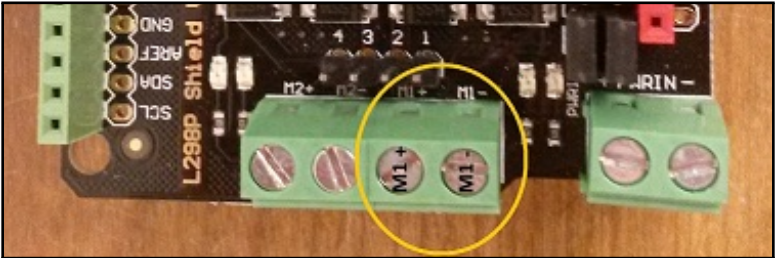


Figure 3.6: Motor Shield Motor Ports

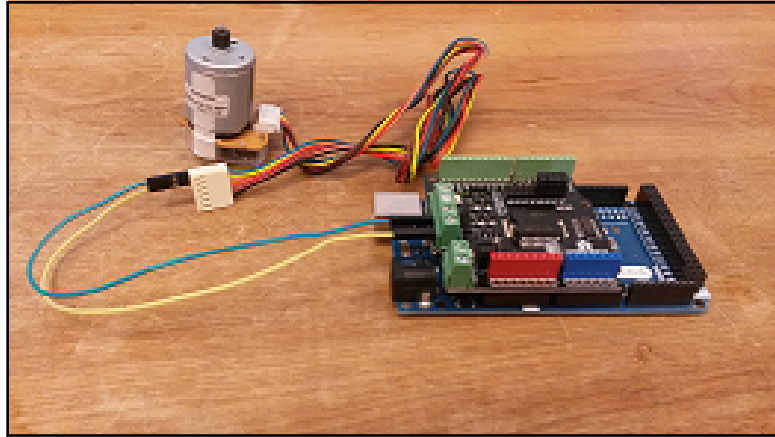


Figure 3.7: Motor Connected to Motor Shield

4. Next, connect the power ports (PWR+ and PWR-) of the motor shield (see Figure 3.9) with the corresponding power ports (PWR+ and PWR-) of the female barrel jack connector (see Figure 3.8). Each connector port has screw terminals, so clamp them down on the wires by using a small flat-head screw driver. The final connection should look similar to Figure 3.10.

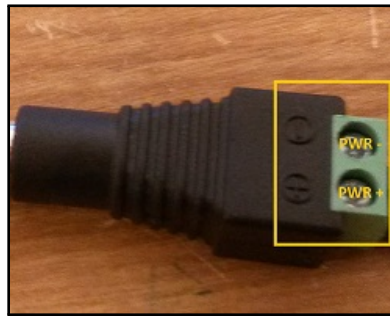


Figure 3.8: Female Barrel Jack Power Ports



Figure 3.9: DFRobotics Motor Shield Power Ports

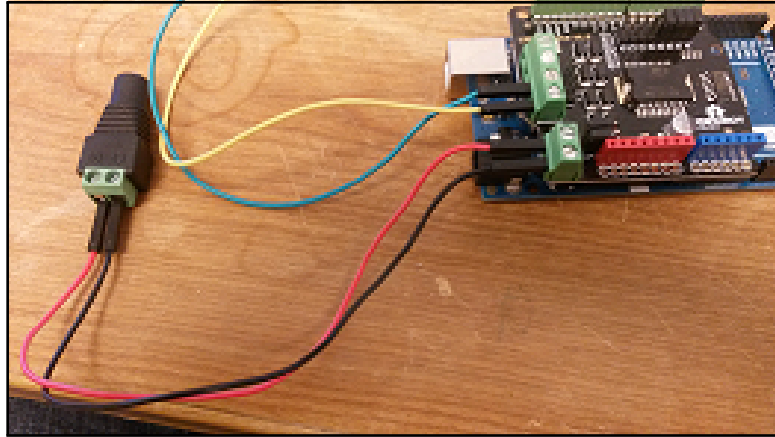


Figure 3.10: Correct Power Connection

5. Now, connect the USB-B connector into the USB-B port on the Arduino board. (The USB-B connector should be the more square-like connector of the USB B to A convertor cable, see Figure 3.11). Don't connect the USB-A connector yet.

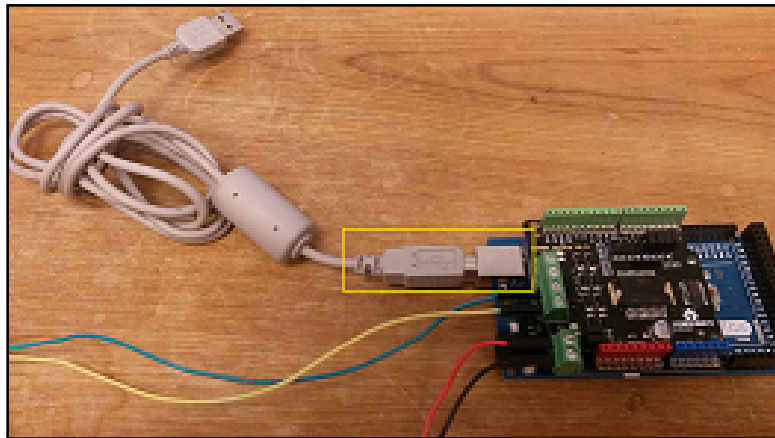


Figure 3.11: USB-B Connected

6. At this point, the motor shield should be mounted on the Arduino, the motor should be connected to the motor shield, and the female barrel jack should be connected to the motor shield. No power will be applied to the circuits until simulations have been created and are ready to run on the Arduino.

### 3.2.3 Software Setup

This section will describe how to set up the Simulink diagram used to run the simple DC motor. Matlab 2014b with Simulink should be installed before beginning this section.

#### Installing Arduino Simulink

7. Open Matlab 2014b and type “supportPackageInstaller” into the command window. Note that for any Matlab versions before 2014, the command “target intaller” could be used in the same way.

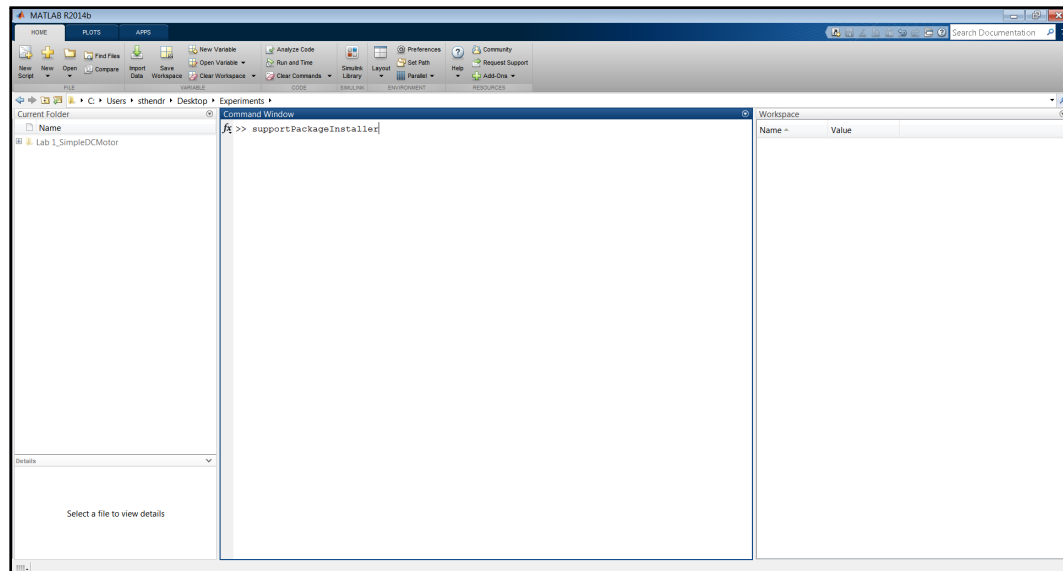


Figure 3.12: supportPackageInstaller in Command Window

8. The page labeled “Support Package Installer” should now be visible. Click the radio button labeled “Install from Internet” and click the “Next” button at the bottom right of the page.

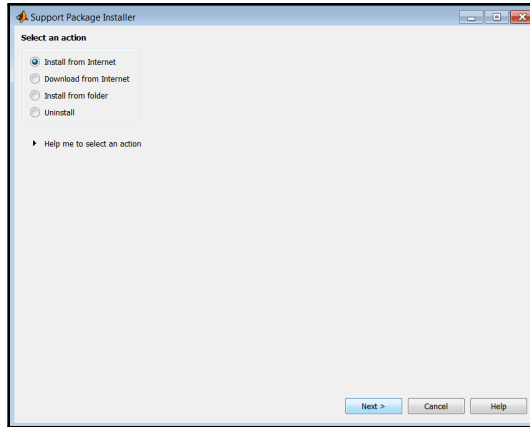


Figure 3.13: Install from Internet

9. Under the “Support for:” list, click on the tab labeled “Arduino.” While on the same page, look under the “Support packages:.” Check the boxes labeled as follows:

- Acquire inputs and send outputs on Arduino Uno, Due, and more
- Run Models on Arduino Uno, Mega 2560, Leonard, and More Boards
- Run models on Arduino Due (different IDE download)

After selecting those listed above, click Next (at the bottom of the page, refer to Figure 3.14).

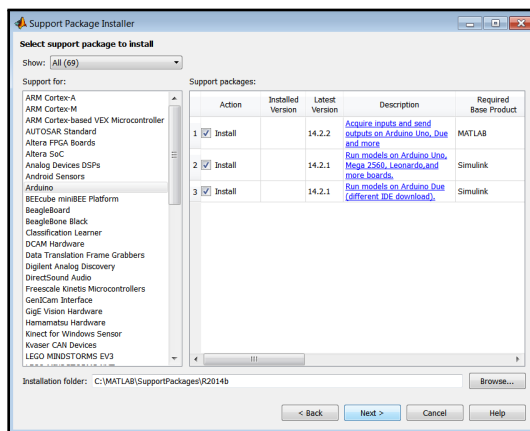


Figure 3.14: Arduino Support Checklist

10. The prompt “Log in to MathWorks Account” should now be visible. Click “Log In” at the bottom of the page.

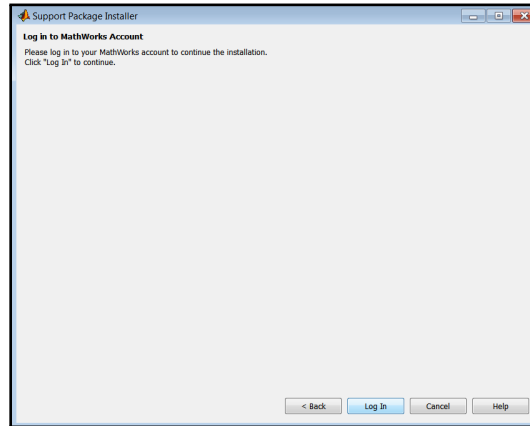


Figure 3.15: Log in to MathWorks Account

11. Now the prompt for a username and password should be visible. This requires the user to type in their respective email address and password associated with their MathWorks account. After this information is entered, click “Log In” under the “Password:” prompt.

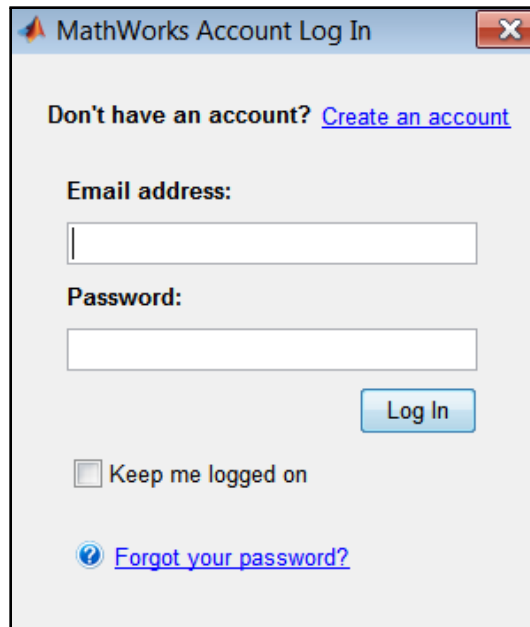


Figure 3.16: Type in MathWorks Username and Password

12. On the page “MATHWORKS AUXILIARY SOFTWARE LICENSE AGREEMENT” read over the license agreement by scrolling, ensure the box labeled “I accept” is checked, and then click “Next” at the bottom of the page.

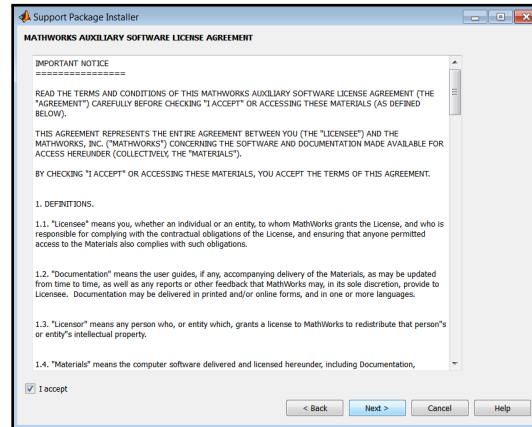


Figure 3.17: License Agreement

13. The page with the heading “Third-party software licenses” should now be visible. This page shows all of the libraries, which were chosen in step 3, to install. Check over the list to ensure the correct software is being downloaded then click “Next.”

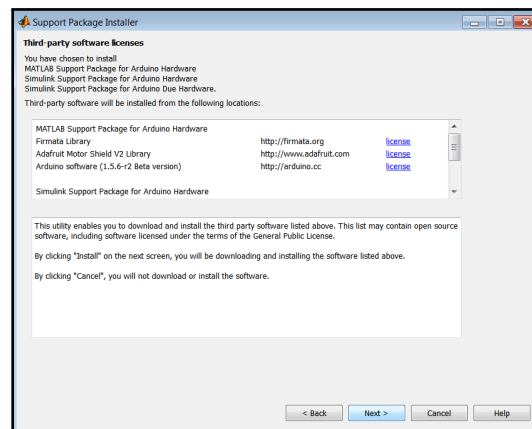


Figure 3.18: Third-party Software Licenses

14. The “Confirm Installation” page should now be visible followed by a list of the packages chosen. After verifying all 3 packages are listed, click on “Install” at

the bottom of the page.

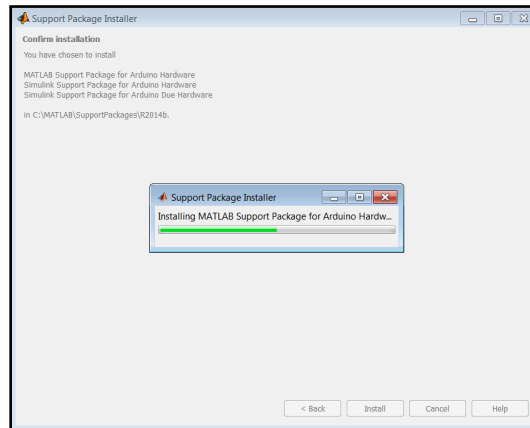


Figure 3.19: Confirm Installation

15. Now the page with the heading "Install/update complete" should be visible and the button option labeled "Finish" should be available at the bottom right of the page. Click Finish. This concludes the installation of the Arduino packages. For support, help, and examples of using the Simulink Arduino blocks, refer to the Simulink Support Package info window that pops up after the installation.

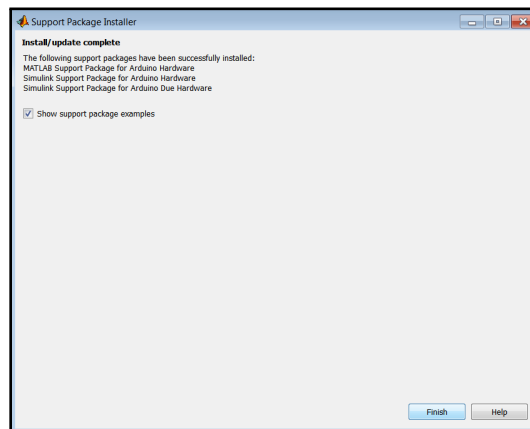


Figure 3.20: Install/Update Complete



## Installing Arduino Mega 2560 Drivers

16. Connect the USB-A connector coming from the Arduino into the desired computer workstation.
17. Once connected, the computer will begin to search for the appropriate drivers for the Arduino Mega 2560 board online. This is an acceptable means of acquiring the appropriate drivers for the board (if the drivers are downloaded correctly).
18. Next, navigate to the Device Manager to see if the drivers were downloaded correctly. There are many ways to do this, but a simple, quick way is examined in this setup. Click the “Start” button in the bottom, right-hand corner of the Windows environment.
19. In the Start Search box, type “mmc devmgmt.msc” without the quotations.
20. Click enter. Now a prompt labeled “User Account Control” will open requiring administrative access to the Device Manager. Click yes. The Device Manager should now be visible.
21. Next, Click the arrow to the left of “Ports (COM&LPT)” in order to expand the list of COM ports. Included in the list should be “Arduino Mega 2560.” If this is not included in the list, unplug the Arduino Mega 2560 board from the USB port and continue to the next step. If it is on the list, you can continue to the next section.
22. Plug the USB from the board back into the computer. Keep track of the device name that appears in the Device Manager (under “Ports (COM&LPT)”) when the Arduino is plugged back in. If the name is not “Arduino Mega 2560” and appears to be “Unidentified Device” it is likely that the driver software did not download correctly. To remedy this, continue to step 8. If the Arduino board is

correctly shown in the Device Manager, then the driver setup is complete and you can continue to the next section.

23. Since the drivers were not downloaded correctly, right click on the “Unidentified Device” option under “Ports (COM&LPT)” and select the option “Update Driver and Software.”
24. Now a page labeled “Update Driver Software” should appear. Select the option “Browse my computer for driver software.”
25. The title “Browse for driver software on your computer” should be visible. Under the title “Search for driver software in this location” enter the location as “C:\MATLAB\SupportPackages\R2014b\arduino-1.0.5\drivers.”
26. A message will appear claiming that Windows cannot verify the publisher of the driver. Disregarding this notice, click the option “Install this driver software anyway.”
27. Windows should now install the required drivers to run the Arduino Mega 2560. To verify this is done correctly, a message will appear stating that the driver software has been installed.
28. In the unlikely case that the two methods above did not work, the Arduino website provides drivers for the Arduino Mega. Simply navigate to the website <http://www.arduino.cc> and click on the “Downloads” tab. Go through the steps outlined in the website in order to get the latest IDE for Arduino. In doing so, the drivers will be downloaded during the process.

### **Setting Up Simulink File**

29. With MATLAB R2014b open, create a new Simulink Model by navigating to the top left hand corner and click the “New” drop down menu.

30. Select the option “Simulink Model” An empty Simulink model window should now be visible.
31. Now find the Library Browser block on the menu bar (below the bar containing file, edit, etc.). Note that the block contains four squares of colors - red, blue and white. Click on the Library Browser block. The Library Browser should now be visible.
32. At this point, the Arduino Simulink package should be visible under one of the options on the left half side of the browser (the portion with a scroll bar). If the option “Simulink Support Package for Arduino” is not available in the menu, please refer to the steps under “Setting Up Arduino Simulink” above in order to have access to the Simulink/Arduino blocks. Otherwise, click on the drop down menu under “Simulink Support Package for Arduino.” New options should now be visible on the right hand side of the browser.
33. On the right side of the browser, double-click on the block labeled “Common.” Various Arduino blocks should now be visible. Click, hold, and drag the block labeled “PWM” and drop it on to the blank Simulink model window. PWM or “Pulse Width Modulation” is a series of voltage pulses used to drive many DC motors via digital output. The PWM signla is defined by the frequency of the pulses and the percent of time that the pulse is high (the duty cycle). The Simulink PWM block uses 490 Hz, so only a duty cycle input is required.
34. Double-click the PWM block and input 5 as the Pin number. Pin 5 on the motor shield is used to specify the voltage being applied to the DC motor. Pin 5 accepts a value from 0 to 255, which varies the duty cycle from 0 to 100 percent in the PWM wave.
35. While in the Simulink Library Browser, click, hold, and drag the block labeled

- “Digital Output” and drop it on to the Simulink model window.
36. Double-click on the Digital Output block and input 4 as the Pin number. Pin 4 on the motor shield is used to define the DC motor direction. In this case, a 1 on pin 4 specifies the clockwise direction and a 0 specifies the counter clockwise direction.
  37. There should now be two blocks (Digital Output and PWM) in the Simulink model window. In order to find the rest of the blocks used in this experiment, open or bring the Simulink Library Browser into focus once again. Click on the option “Commonly Used Blocks” on the left half portion of the window. Grab two “Constant” blocks and one “Gain” block and drop them into the blank Simulink model window.
  38. Connect the output of one Constant block into the input of the Gain block and connect the output of the Gain block into the input of the PWM block. Click on the name “Constant” and change it to “Voltage.” Click on the name “Gain” and change it to “Voltage to PWM.”
  39. Double-click on the Voltage to PWM gain and enter a value of  $255/15$ . This value converts the input from a voltage to a duty cycle percentage. A value of 0 into the PWM block represents 0 percent duty cycle and a value of 255 into the PWM block represents a 100 percent duty cycle.
  40. Additionally, connect the output of the other Constant block into the input of the Digital Output block. Click on the name “Constant” and change it to “Direction.” The result of the connections should look similar to Figure 3.21.
  41. Before continuing to the Configuration Parameter setup, simply plug in the USB-A connector from the Arduino board into the desired computer running this project.

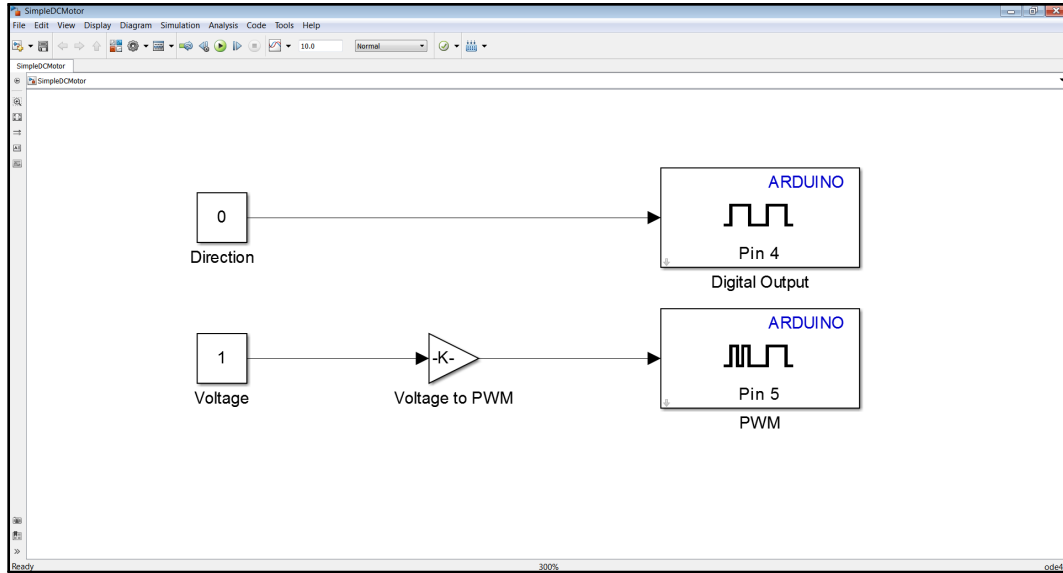


Figure 3.21: Final Simulink Model Used for Simple DC Motor

### Configuration Parameters

42. Click on the icon shaped like a serial port in the top right hand corner of the Simulink model window, labeled “Deploy to Hardware.” This button is used to build the code in Simulink and “deploy” or load it (in Arduino C code) onto the corresponding Arduino board.

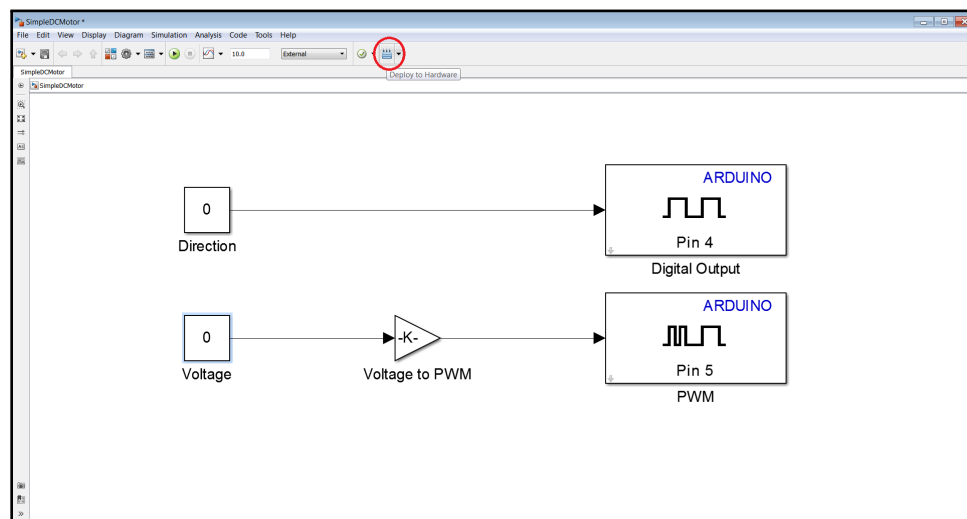


Figure 3.22: Click Deploy to Hardware

43. On the page labeled “Configuration Parameters: file name/Run on Hardware Configuration (Active)” click on the drop down menu next to “Target Hardware:.” Choose the option “Arduino Mega 2560.”

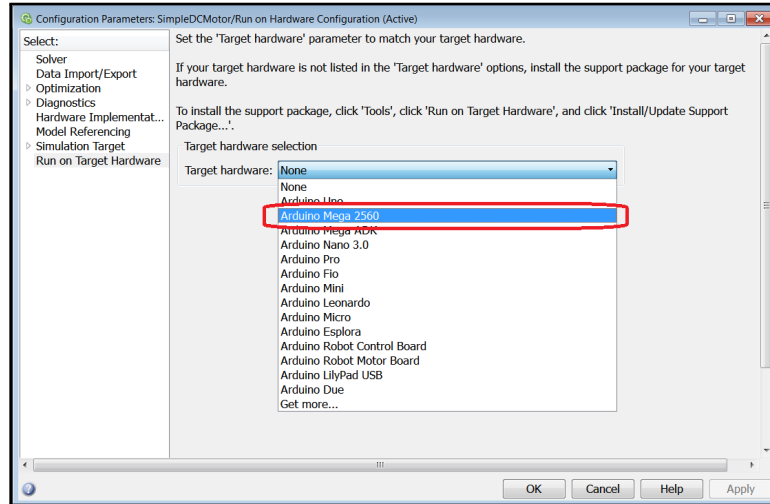


Figure 3.23: Set Target Hardware to Arduino Mega 2560

44. Now, many more options should appear under the Target Hardware menu. While many of these options are useful in regards to communication, leave them as pictured in Figure 3.24. Ensure that the “Set host COM port:” is chosen to be “Automatically.” This will allow Simulink to automatically detect the Arduino COM port. (If Simulink is unable to find the Arduino COM port, navigate to the device manager and find out the corresponding COM port for the Arduino Mega 2560. Also set the “Set host COM port:” to Manually and enter the correct COM port).

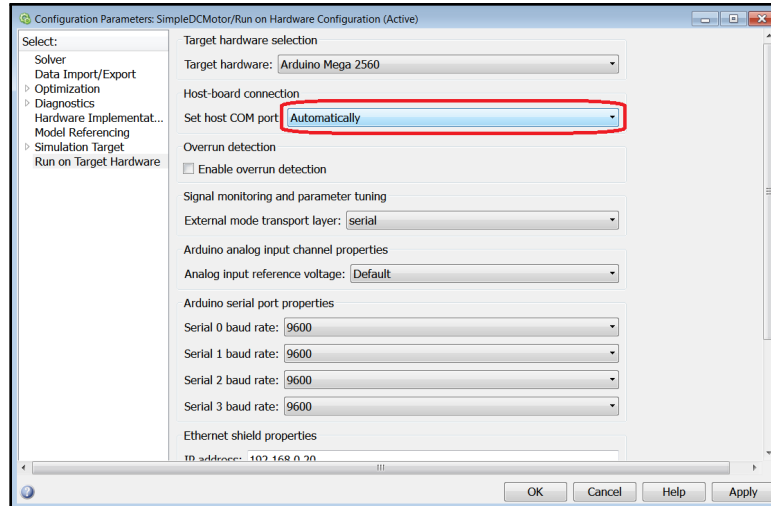


Figure 3.24: Set COM Port to Automatic

45. On the left pane, click on the option “Solver.” While on the Solver page, change the “Fixed-step size (Fundamental sample time):” to 0.03 seconds and the “Stop time:” to inf (meaning infinite). (See Figure 3.25 After making these changes, now click “OK” at the bottom of the page.

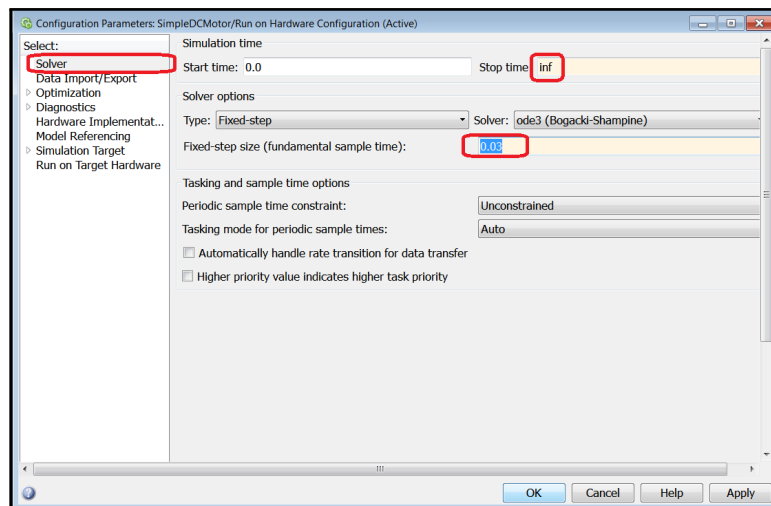



Figure 3.25: In Solver Pane, Change time to Inf and step size to 0.03

46. At this point, the model is now ready to run on the Arduino. Create a folder named “Experiments” on your computer desktop. Save this file by clicking the “Save” button  at the top. Save the file into the “Experiments” folder as

“SimpleDCMotor.slx” and click okay.

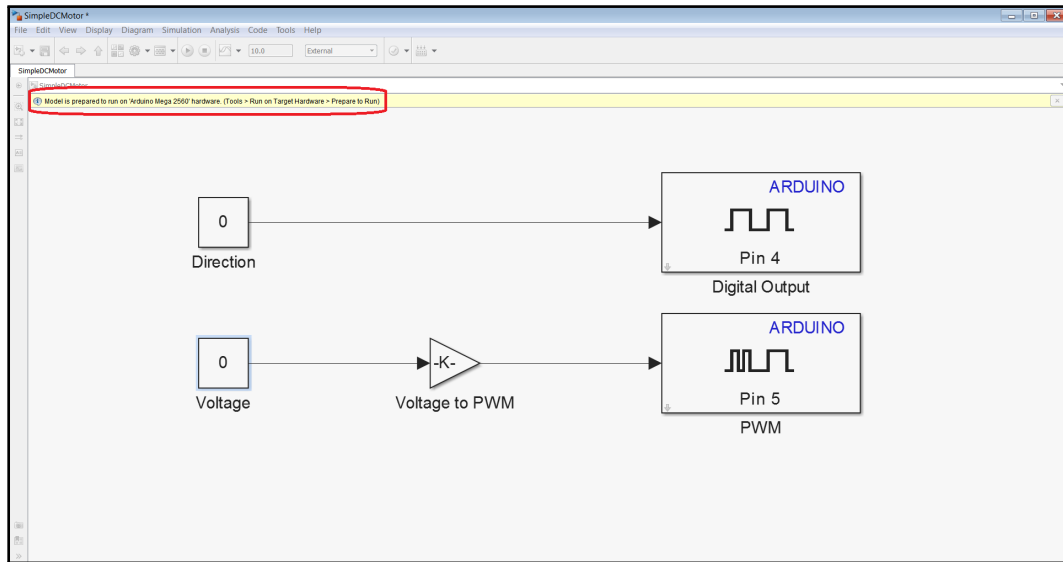


Figure 3.26: Model Prepared to Run on Arduino Mega



### 3.3 Experimental Procedures

Now that the hardware and software have been set up, the exercises can be performed. The exercises in this experiment will use two different “modes”. In the external mode, data can be conveniently passed from the software on the Arduino to Simulink running on the PC. In normal mode, it is more difficult to transfer data back and forth, but the software on the Arduino can run faster. To learn more about these two modes, perform Experiment: *Sampling and Data Acquisition*. The first exercise will be to change the PWM values sent to the motor and visually analyze the resulting responses in external mode. The next experiment will be to add a slider gain in between the Voltage block and the Voltage to PWM gain block, which will enable the speed of the motor to be changed fluidly. Finally, the model will be loaded onto the Arduino via Normal mode. Normal mode allows faster sampling times, but at the expense of less convenient access to data. This will be required more often in later labs.

#### 3.3.1 Exercise 1: Comparing Duty Cycles

47. In the Simulink model window, double-click on the “Voltage” input block, and change the value to 5. Then double-click on the “Direction” block, change the value to 1, and click the green arrow at the top of the Simulink model window to begin the simulation. Vary the input voltage with the following values: 6, 8, 12, and 15. Record the results by describing how the motor speed changes.
48. While keeping the value of the Voltage block as 15, change the value of the Direction block from 1 to 0. Record how the motor response changes.

#### 3.3.2 Exercise 2: Slider Gain Analysis

49. Using the same Simulink model, add a slider gain (from the Math Operations Section of the Simulink Library Browser) between the Voltage block and Voltage

to PWM block. Double-click the slider gain block and change the min and max values in the block to 0 and 15, respectively. Also, add a “Scope” block (from the Sources section of the Simulink Library Browser) to the output of the slider gain. Now click on the green run arrow. While the model is running, slide the gain from left to right, record the value from the scope for each increase in the slider gain, and note the change in speed.

50. With the same model, change the direction from 0 to 1 (or vice-versa). Now slide the gain and record the resulting motor response. Did the motor start speeding up and slowing down in the opposite direction?

### **3.3.3 Exercise 3: Normal Mode**

51. Remove the Slider-Gain block from the Simulink model used in Exercise 2. The Simulink model should now look like the one used in Exercise 1.
52. Double-click on the Voltage block and change the value to 8. Additionally, double-click on the Direction Block and change the value to 0. Now, at the top of the Simulink model, where the drop down menu shows “External,” click the drop down menu and select “Normal.” After this, click on the “Deploy to Hardware” button, which is shaped like a serial port, in order to build the model onto the Arduino. After the project compiles, record how the motor responds. Does it run at a fixed speed?
53. Change the value in the Direction block to 1. Click the Deploy to Hardware button. Describe the resulting response. Does the motor go the opposite direction?

## **3.4 Conclusion/Student Feedback**

### **3.4.1 Conclusion**

This chapter provided a basic introductory experiment on the use of the Arduino to control a DC motor. Simulink blocks were used to provide real time access to the input of the motor using the “external mode,” implementation. An introduction was also provided to the “normal mode” operation, in which software is downloaded to the Arduino and runs outside the Simulink environment. This experiment forms the foundation for many future experiments, as the concepts presented here are used in almost all other labs.

### **3.4.2 Student Feedback**

## CHAPTER 4

### Introduction to 3D Printing

#### 4.1 Objective

This chapter describes a typical introductory lab experiment. The objective is to introduce students to 3D printing by printing parts that will be used in later experiments. The introductory experiments should help the students build their confidence. This experiment is described for a specific printer, but could be used with other printers with a few modifications.

#### 4.2 Setup

##### 4.2.1 Required

##### Hardware

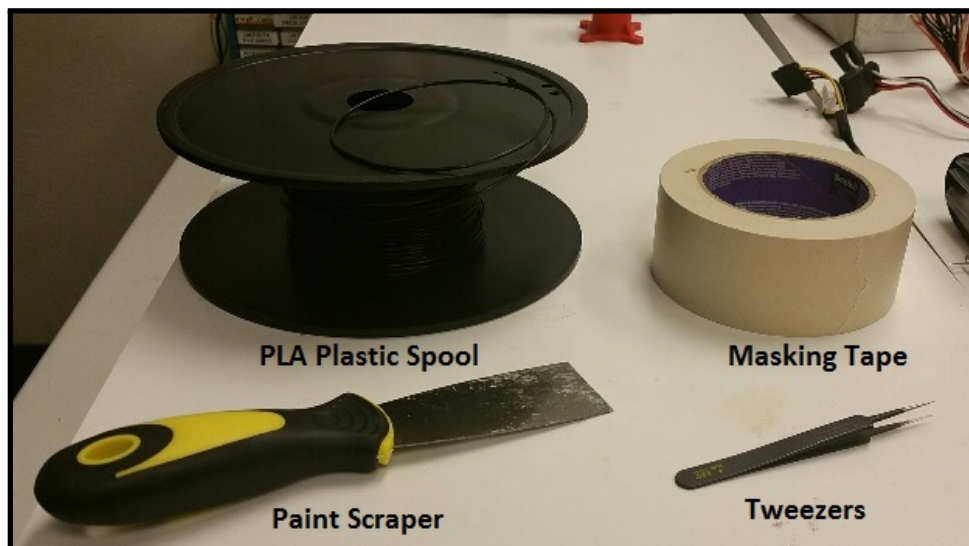


Figure 4.1: Hardware Required for *Introduction to 3D Printing* Experiment



Figure 4.2: Solidoodle SD4 3D Printer Required for *Introduction to 3D Printing* Experiment

- Solidoodle SD4 3D Printer
- PLA Plastic Spool
- Masking Tape (or painters tape)
- Paint Scraper
- Tweezers

#### Software

- Repetier Host V0.85b
- Windows 7

## Previous Experiments

- This is an introductory experiment and does not require that any other experiments be performed first

### 4.2.2 Software Setup

#### Downloading Repetier Host (If the printer software is not already available on your print server)

1. Download the file **setupRepetierHostSolidoodle\_0.85.1.exe** from the Take Home Labs website (<http://thl.okstate.edu>).
2. Double-click on the file and follow the setup instructions.

**NOTE:** Follow the standard installation procedures. This would require clicking next for each prompt in the setup window until it says to click “Finish”.

3. Double-click on the Repetier Host desktop icon . If the Repetier Host page appears with no error, then the setup is complete. If you have errors, refer to the <http://www.repetier.com/technicalsupport/> page for Repetier Host support.

### 4.2.3 Hardware Setup

#### Power and USB for Solidoodle Printer

4. Obtain a power supply and USB B-to-A Converter for the Solidoodle SD4 3D Printer.

**NOTE:** If the power supply and USB cable is already plugged into your printer, you may skip this section.

5. Plug the power supply (barrel jack) into the Solidoodle 3D printer. Connect the plug side of the power connector into a wall outlet. If the light inside the Solidoodle printer does not come on, check to see that the power cable is connected correctly to the power socket.
6. Plug the USB-B connector into the back of the 3D printer (see Figure 4.3), and the USB-A connector into your computer's serial port.

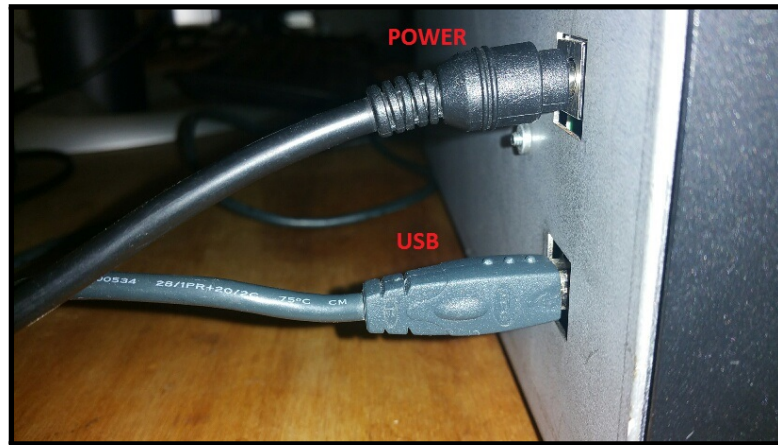


Figure 4.3: Tape Placed Length-Wise On Printer Surface

7. You have completed the power and USB connections for the Solidoodle 3D printer.

### Placing Masking Tape On Printing Surface

8. Obtain masking tape or painters tape.
9. Take a piece (slightly longer than the length of the bed, in the direction coming from you, and going to the back of the printer) and rip it from the roll.
10. Place the piece of tape on the bed in the length-wise direction (see Figure 4.4).

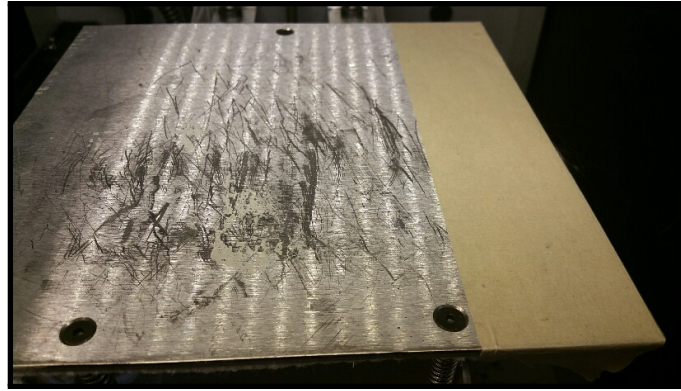


Figure 4.4: Tape Placed Length-Wise On Printer Surface

11. Cover the entirety of the printer surface with tape, leaving a gap between each strand, as shown in Figure 4.5.




(a) Leave Gap Between Tape Strands (b) Cover the Whole Printer Surface With Tape

Figure 4.5: Cover Printer Surface With Tape


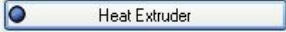
12. The masking tape setup for the 3D printer is now complete.

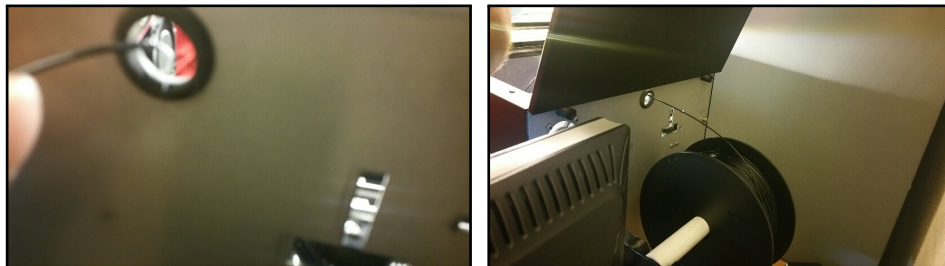
### Placing Plastic Into Extruder (with no existing plastic in the nozzle)

13. Retrieve the PLA plastic spool.

14. Open **Repetier-HostV0.85b** by double-clicking on the icon .



15. Once the Repetier Host program appears, click on  in the upper left-hand corner of the screen. This will connect the Repetier Host program with the Solidoodle 3D printer.
16. Click on the tab labeled **Manual Control**. Once the GUI has appeared, scroll down until a bar labeled **Extruder 1** appears. Type “200” in the box to the right of the bar and press “Enter”. Click on the **Heat Extruder**  icon above.
17. Allow the Extruder to heat to 200 degrees Celsius. Once the text to the right of the **Extruder 1** button shows 200 degrees Celsius, proceed to the next step.
18. Feed the PLA plastic spool through the back opening of the Solidoodle printer.



(a) Feed PLA Plastic Into Printer (Close-up View)      (b) Feed PLA Plastic Into Printer (Side View)

Figure 4.6: Feed PLA Plastic Into Printer

19. Inside the 3D printer, pull the plastic through the hole further until it reaches the nozzle. Push the plastic through the top of the nozzle assembly until it touches the two disks. Make sure the plastic is on top of the groove between the two disks. Do not push it any further.

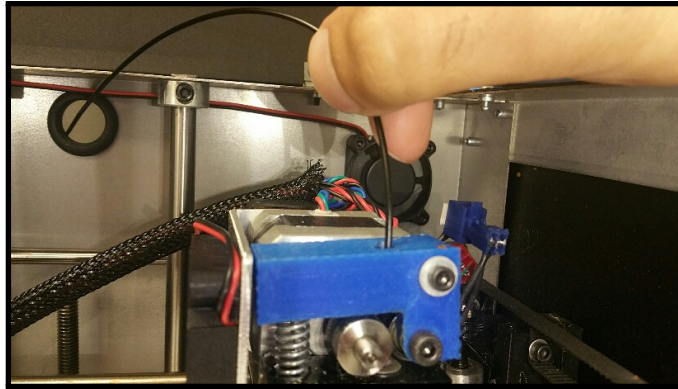


Figure 4.7: Put Plastic In Nozzle Assembly

20. In the Repetier Host program, set the Speed [mm/min] as 100, the **Extrude** [mm] as 10, and the **Rertract** [mm] as 10. Click on the extrude icon (while pressing the plastic slightly into the nozzle), which is the down-arrow farthest to the right, shown in Figure 4.8.

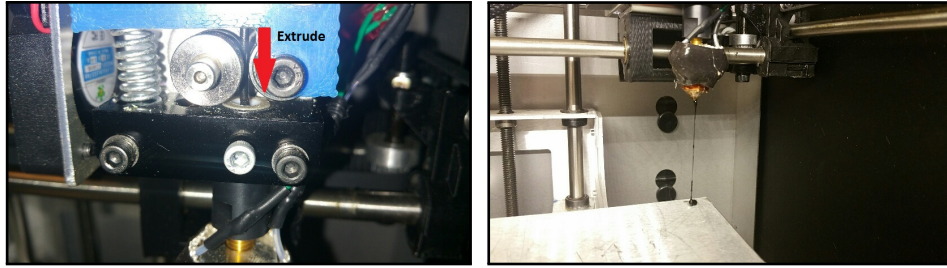


Figure 4.8: Click Extrude

21. Continue clicking on the extrude button, waiting between each extrusion, until you begin to see the color of your plastic appear from the nozzle.

**NOTE:** Click the extrude button a couple of more times after you begin to see plastic to ensure you have a clear stream of plastic coming through. Make sure that the plastic is between the two disks of the nozzle assembly the whole time you are extruding.

22. Remove the excess plastic from the nozzle and the printer area using tweezers.



(a) Extrude Direction

(b) Plastic Appearing From Nozzle

Figure 4.9: Extruding Plastic Through the Nozzle

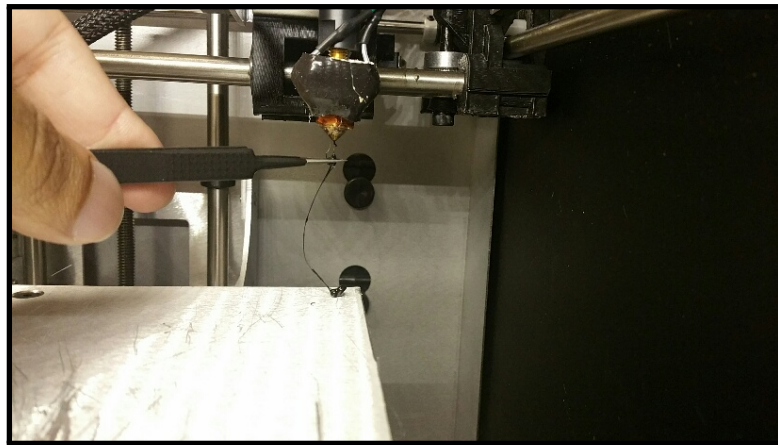


Figure 4.10: Remove Excess Plastic From Nozzle

23. You have completed the extruder setup for the Solidoodle 3D Printer. Skip to the **Experimental Procedures** section if you are satisfied with the plastic material currently in the nozzle. If you wish to retract out the material in the nozzle (especially old plastic,) continue to the next section.

#### Placing Plastic Into Extruder (with unwanted plastic in the nozzle)

24. Repeat Steps 14-17.

**NOTE:** If you are already connected to Repetier Host, start from the next step.

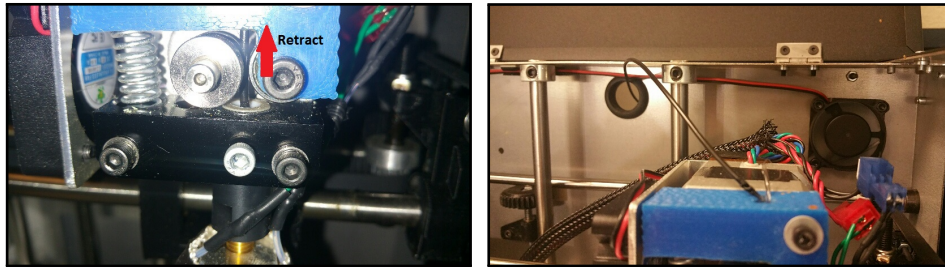
25. Hold on to the plastic coming out of the nozzle. While gently pulling on the

plastic in the nozzle, click on the **Retract** button shown in Figure 4.11.



Figure 4.11: Click Extrude

26. Continue pulling and clicking retract until the plastic comes out of the nozzle.



(a) Retract Direction

(b) Plastic Pulled From Nozzle

Figure 4.12: Retracting Plastic From the Nozzle

27. Once you have pulled the plastic from the nozzle, repeat steps 18 - 23.


**NOTE:** With old plastic in the nozzle (especially when the color that you just retracted is different from what you wish to extrude into the nozzle) it will take a while to extrude all of the old plastic out. Repeat step 21 until the color of the plastic is the color of the plastic you are extruding through the nozzle.

28. You have completed the hardware setup for this experiment.

## 4.3 Experimental Procedures

### 4.3.1 Exercise 1: Adjusting Print Surface

#### Printing Bed Level Part

29. Download the **BedLevel.stl** file from the website.
30. Open Repetier Host and click **Load**  in the upper left hand corner.
31. Navigate to the folder where **BedLevel.stl** is saved, click on it, and then click “Open” at the bottom right-hand corner of the window. The part should now be in the 3D Viewer (see Figure 4.13).

**NOTE:** The extruder should already be heated up to 200 degrees Celsius. If this is not the case, repeat step 16.

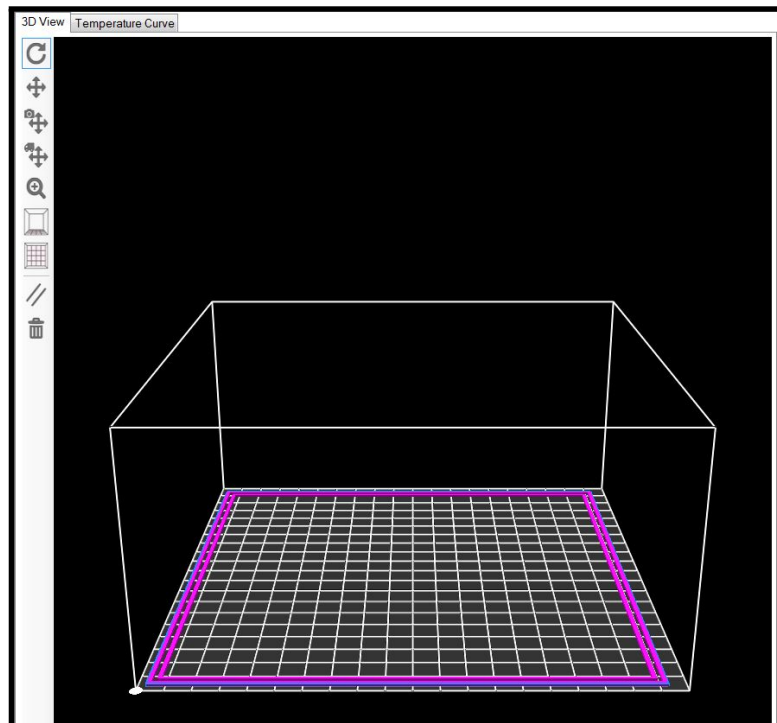


Figure 4.13: 3D View of BedLevel.stl Part

32. In the Repetier Host program, click on the tab labeled “Slicer” on the right

hand side of the screen. Click on the **Configure** button on the top right hand corner of the Slicer tab window (see Figure 4.14).

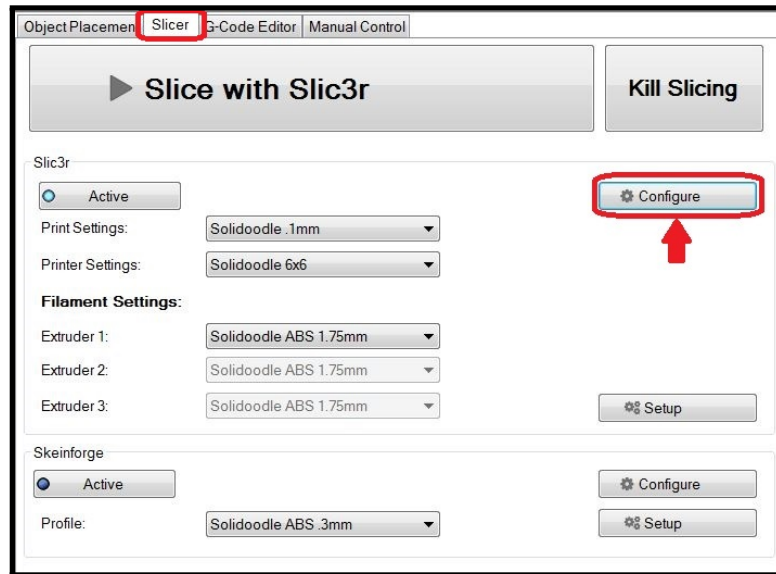


Figure 4.14: Click Configure

33. Once the Slicer Configure page appears, click on the drop-down menu under the **Print Settings** tab and select **Solidoodle .3mm**. This will configure the extruder to create strands of plastic that are .3mm thick.

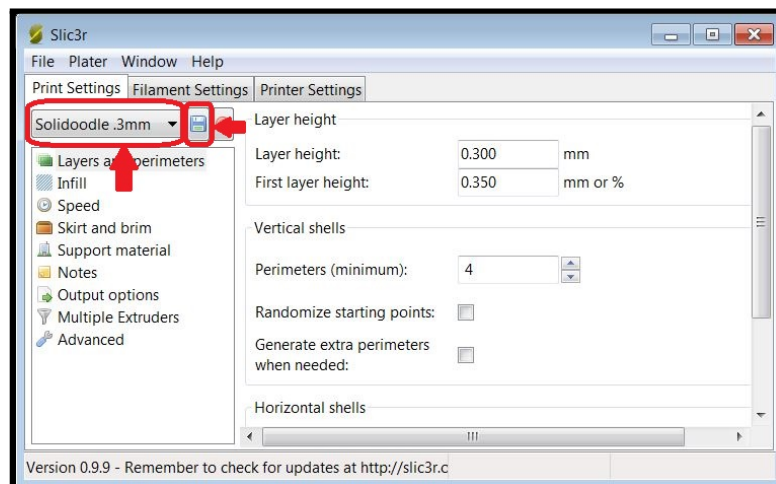


Figure 4.15: Print Settings With .3mm Resolution


34. Click on the **Printer Settings** tab. In the column on the left side of the Printer

Settings tab window, click **Custom G-code**. Add the following code to the box under the “Start G-code” heading:

**G28 ;home allaxes**

**G1 z5 F5000 ;liftnozzle**

This code will run once you begin printing your 3D part. The code’s function is to first calibrate the location of the nozzle (this will cause the extruder to move around, finding it’s home position). Next, the code will wait for the extruder nozzle to heat up to the specified G-code settings, which are automatically generated for you later on.

35. Click the **Save Current Printer Settings** button  next to the “Solidoodle 8x8” drop-down menu then click on the **Filament Settings** tab.

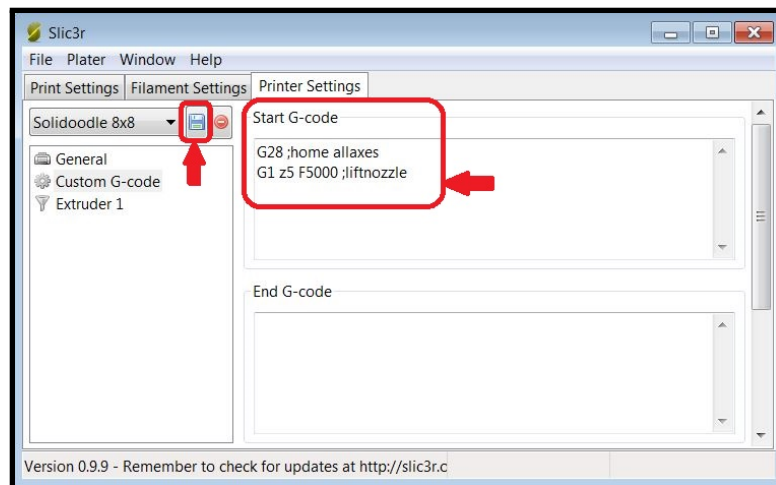



Figure 4.16: Printer Settings

36. Click on the drop-down menu under the Filament Settings tab and click **Solidoodle PLA 1.75mm**. Click the **Save Current Filament Settings** button  to the right of the drop-down menu.



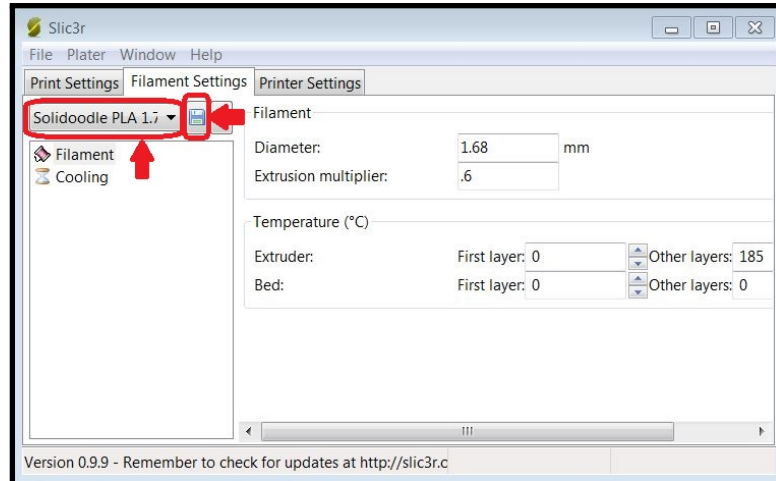



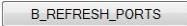
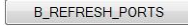
Figure 4.17: Filament Settings

37. Click **Close**  at the top of right-hand corner of the Slicer window.


38. In the main Repetier Host program window, click on **Printer Settings**  at the top right-hand corner of the window (You can also use the keyboard combination **Ctrl-P**).

39. Click the drop-down menu **Baud Rate** and choose 57600.

40. Next to “Port:” click on the drop-down menu and choose the COM port that corresponds to the 3D printer. If you only have one option to choose for the COM port (while the printer is connected) then choose that option and skip the next step. Otherwise, continue to the next step.

41. If multiple COM ports are listed, unplug the USB-B cable from the Solidoodle printer and click . Check the ports listed in the drop-down menu. Take note of the ports currently listed in the drop-down menu. Plug the USB-B cable back into the Solidoodle printer and click  once more. Check the ports. The port that was not listed before is your printer’s COM port number. Choose your printer’s COM port number by clicking on it in the drop-down menu.



42. Click **OK** at the bottom of the Printer Settings window. On the right-hand side of the main Repetier Host program window, choose the **Slicer** tab. Click on the big button that says **Slice with Slic3r**. A small window that says “Slicing info” will appear temporarily.
43. Once the G-Code tab comes into view, click on the “Run Job” button .
44. The printer should now begin heating up the extruder (if the extruder is not already hot).

### Leveling the Bed

45. Open the Solidoodle printer door once the extruder begins moving.

**CAUTION:** Do not put any body parts near the extruder head. It will be extremely hot and can result in burns when touched.

46. Find the three wing nuts beneath the print bed.

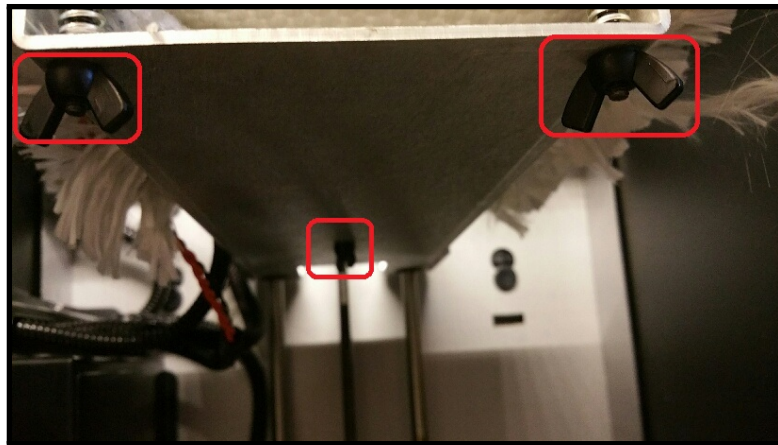



Figure 4.18: Wingnuts Under Printing Surface

47. After the extruder has made two or three loops around the plate, observe the plastic coming out of the extruder nozzle. Follow the guidelines below for correcting the level of the plate:

- If the plastic appears to be thin and drooping (extruder is too far away from print surface) on one side of the plate, twist the wingnut corresponding to that side to the left. This will raise the plate.
- If the plastic appears to be thick and flat (extruder is too close to print surface) on one side of the plate, twist the wingnut corresponding to that side to the right. This will lower the plate.
- If the plastic is desirable on one side, do not change the level of that side.



Figure 4.19: Desired vs. Thin Plastic On Plate

48. Continue adjusting the plate levels until all of the plastic on the plate is extruded as desired. After this, you may click **Kill Job**  on the main Repetier Host window.
49. This completes the bed leveling exercise. Close Repetier Host.

### 4.3.2 Exercise 2: Printing Motor Load

50. Open Repetier Host. Load the **Motor Load.stl** file from the website into the Repetier Host program. Figure 4.20 shows the part as it should appear in the 3D viewer.

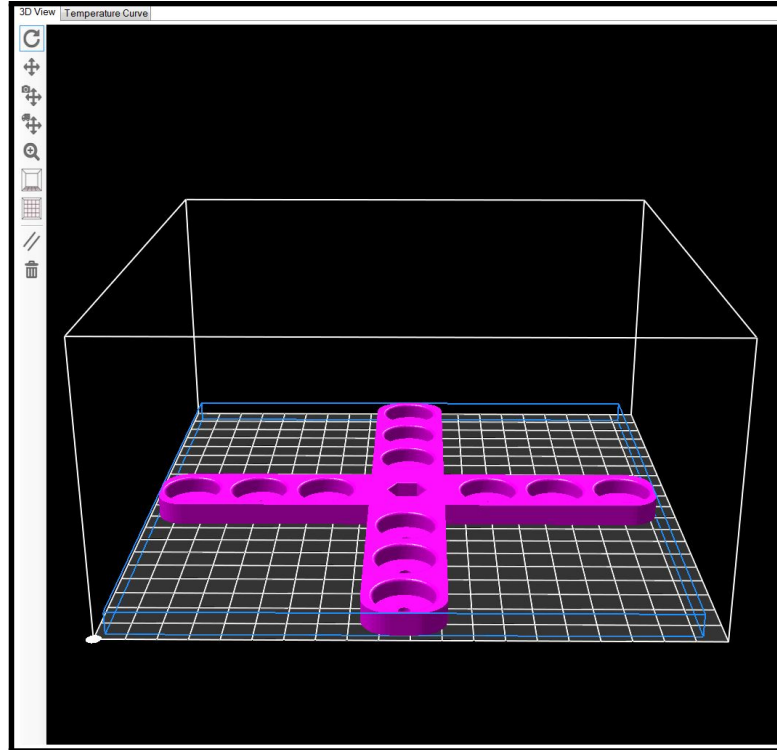


Figure 4.20: Motor Load in 3D Viewer

51. Repeat steps 32-43 with the following exceptions.

- If the filament settings are already setup as they were in steps 32-36 then you may skip those steps.
- Skip steps 38-42.
- After the first layer of the part has been laid on the plate (you will notice the bed level lower) click on the **Manual Control** tab on the main Repetier Host program window. You should notice that the **Heat Printbed** button  Heat Printbed is illuminated. This means the print bed is now heating. In the box next to “Temp” type **50** and press **Enter** on the keyboard. The print bed should now heat up to 50 degrees Celsius.

52. Observe the 3D printed part as it is layered by the extruder. Click the **Kill**

**Job** button if any of the following things occur:

- The part begins to warp. This indicates that the print bed is too hot, or the tape was not pressed to the plate enough.
- The extruder position becomes offset from where the part was originally being laid to the plate. This can occur if the spool gets wound up on the back, restricting the movement of the extruder. It is important to pay attention to the printer, if just by sound, during runtime. This will eliminate many issues you might run into later on.
- Anything catches on fire or becomes too hot.
- The printer begins to make loud, peculiar noises. The only noises you should hear during runtime are the fans and the extruder moving along (producing a light, whirring noise).

53. Once the part has fully printed, this exercise is complete. Once the extruder and tray bed stop moving, you may close out of Repetier Host.

### 4.3.3 Exercise 3: Printing Insert

54. Open Repetier Host. Load the **Load Insert.stl** file from the website into the Repetier Host program. Figure 4.21 shows the part as it should appear in the 3D viewer.

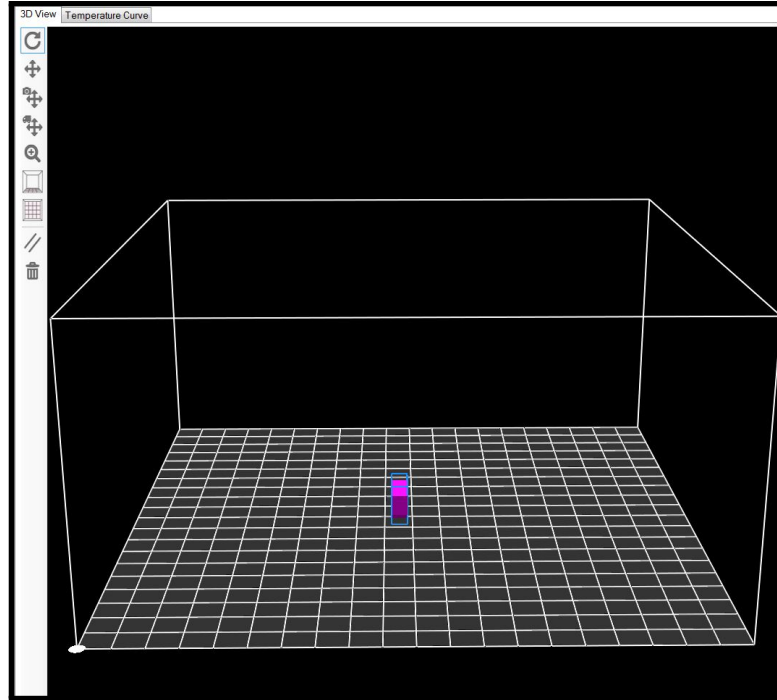


Figure 4.21: Gear Insert in 3D Viewer

55. Notice that the part is standing on one side. Under the **Object Placement** tab on the right-hand side of the screen, change the values for Translation, Scale, and Rotation to have the values for their respective X, Y, and Z values as shown in Figure 4.22.

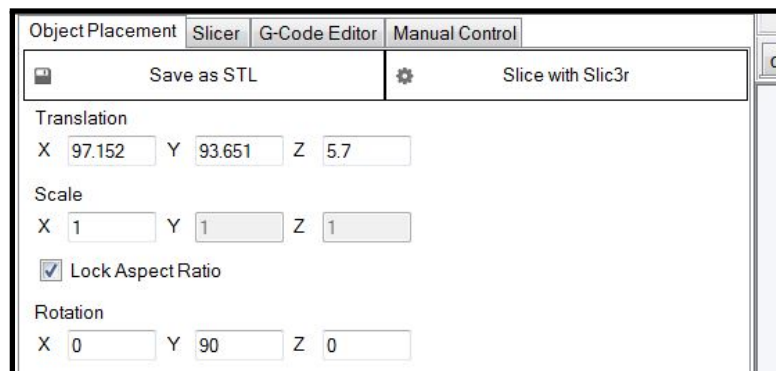


Figure 4.22: Correct Object Placement Values

56. After making the changes to the Object Placement values, the 3D viewer should look like Figure 4.23.

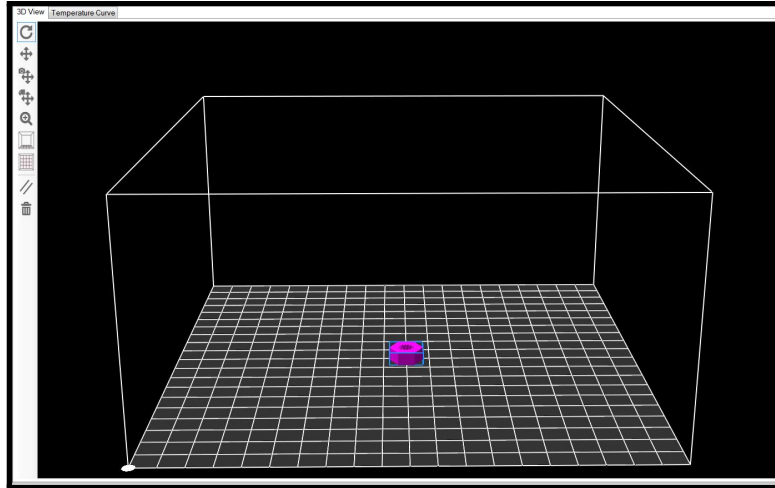


Figure 4.23: Gear Insert in 3D Viewer (Corrected)

57. Repeat steps 32-43 with the following exceptions.

- In step 33 select **Solidoodle .1mm** in the drop down menu instead of **Solidoodle .3mm**.
- Skip steps 38-42.
- After the first layer of the part has been laid on the plate (you will notice the bed level lower) click on the **Manual Control** tab on the main Repetier Host program window. You should notice that the **Heat Printbed** button  Heat Printbed is illuminated. This means the print bed is now heating. In the box next to “Temp” type **50** and press **Enter** on the keyboard. The print bed should now heat up to 50 degrees Celsius.

58. Repeat also steps 52 and 53.

59. You have now completed this experiment.

#### **4.4 Conclusion/Student Feedback**

This chapter provided a basic approach to 3D printing. All of the parts created in this experiment will be used in later experiments. This experiment serves as an introductory experiment and is by no means an “end all” to 3D printing. As students gain exposure to using the 3D printers, they will begin to create more complex methods that can be applied to the hardware.

## CHAPTER 5

### Open Loop Step Response

#### 5.1 Objective

This chapter describes a laboratory that will be used in an undergraduate “Dynamic Systems or Control” course. The objective is for students to find a first-order model for a DC motor using the open loop step response. Students will be given some of the motor parameters from the motor data sheet and will derive the rest of the parameters using the motor step response. They will then compare the actual response of the motor with the theoretical response.

#### 5.2 Setup

##### 5.2.1 Required Materials

###### Hardware

All hardware from the *Simple DC Motor* experiment will be required. Additionally, the following hardware will be used:

- Motor Load and insert created from the *Introduction to 3D Printing* experiment
- 3D printed Arduino Base Holder
- 3D printed Motor Holder
- 3D printed Motor Fastener (Two of them)
- M3 x 1/4 Inch Screws (6 PC Mounting Screws)



- #4-40 x 1/2 Inch Screws (6 with corresponding nuts)
- Sticky Tack (2 Ounce Package or Greater)
- 4 wires

## Software

- Matlab/Simulink 2014b
- Windows 7

## Previous Experiments

This laboratory requires that the following laboratories have been completed:

- Simple DC Motor
- Sampling and Data Acquisition
- Introduction to 3D Printing

### 5.2.2 Hardware Setup

#### Mounting Arduino and Motor to Case

1. In addition to all of the hardware from the *Simple DC Motor* laboratory, obtain the 3D printed Arduino Base, Motor Holder, 2 Motor Fasteners, and both the M3 and #4-40 screw sets (12 total).



Figure 5.1: All Hardware

2. Unplug the USB B to A cable from the Arduino. (There should be no wires besides the motor and power connector wires connected to the Arduino.)

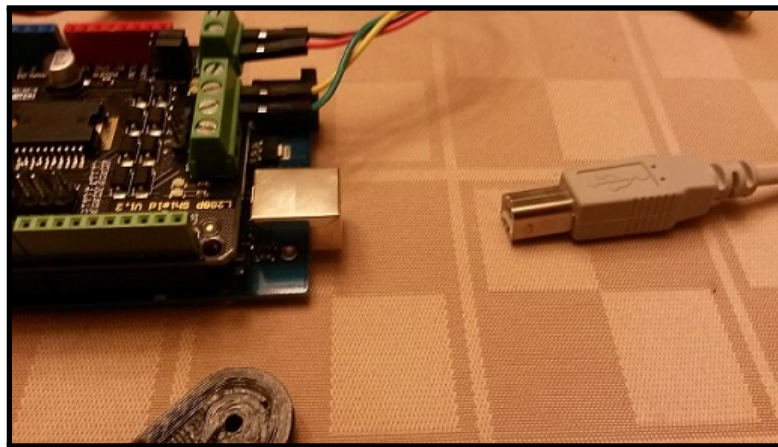


Figure 5.2: Unplug USB B to A cable from the Arduino

3. Remove the motor shield from the Arduino.

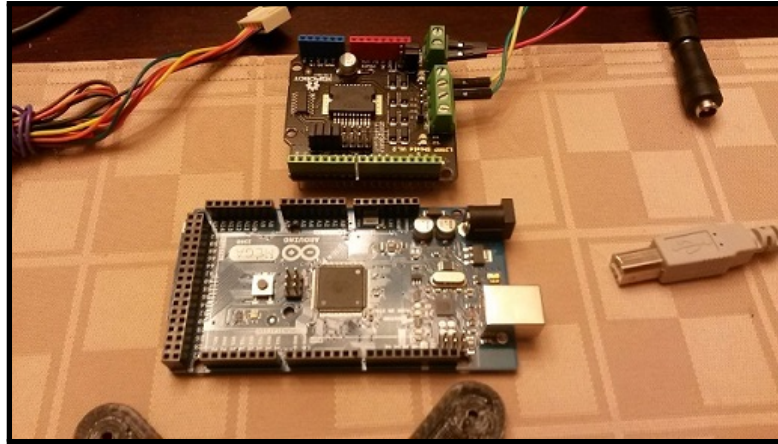


Figure 5.3: Remove Motor Shield from the Arduino

4. Place the bare Arduino onto the 3D printed Arduino case. Ensure that all of the holes in the case line up with the holes on the Arduino.
5. Fasten the M3 x 1/4 inch screws into each hole with the Phillips head screw driver. (Make sure the the screws are snug, but do not over-tighten.)

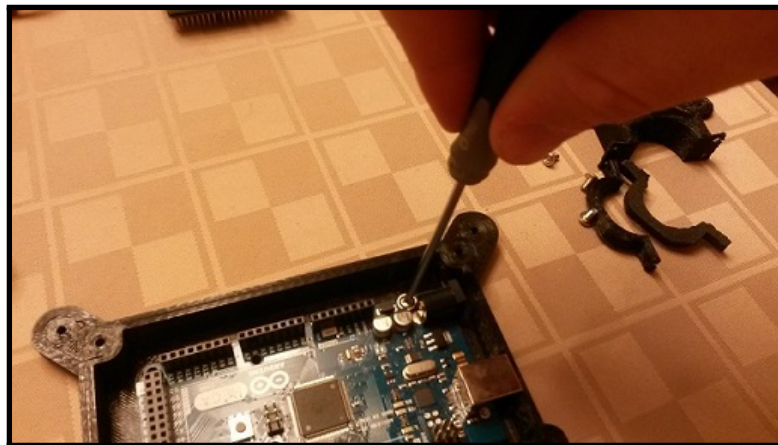


Figure 5.4: Fasten Screws with Phillips Head Screw Driver

6. Attach the 3D printed motor holder to the end of the case opposite the USB port on the Arduino. Use two #4-40 screws to fasten the motor holder to the end of the case.

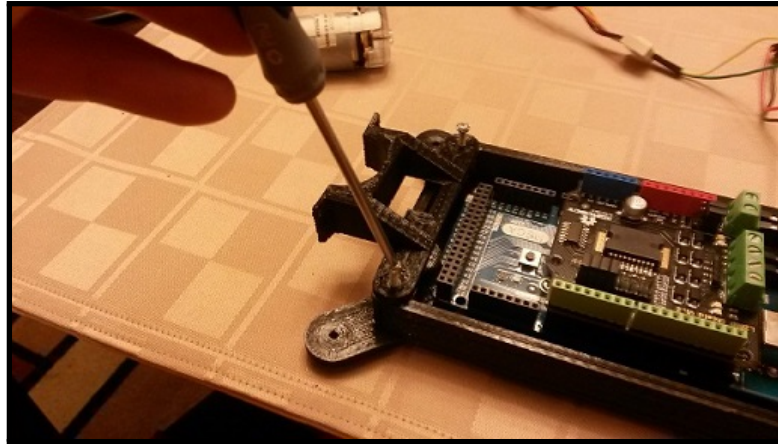


Figure 5.5: Attach the 3D Printed Motor Holder to the Arduino

7. Place the motor into the curved surface of the motor holder. Attach the clamp pieces (one at a time) to the outer flanges of the motor holder (around the motor) with nuts and screws. Do not over-tighten the nuts and screws. (When attaching the motor, make sure the encoder is pointed to the side, so the case can sit flat on a surface.)

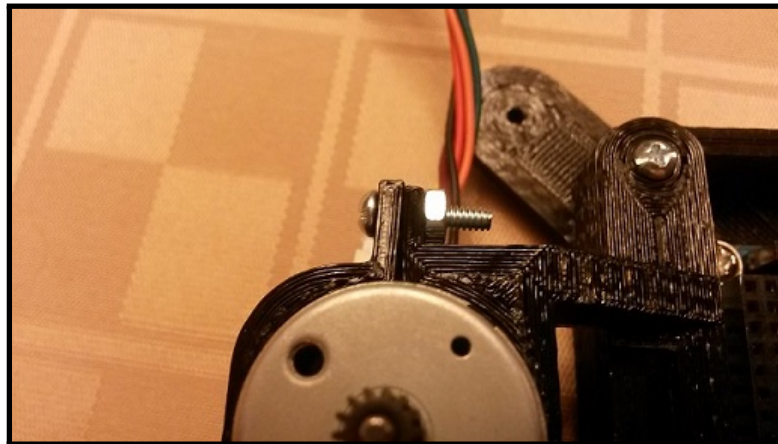


Figure 5.6: Motor Placed Inside Motor Holder With Clamp Pieces Attached

8. Plug the motor shield and USB cable back into the Arduino.



Figure 5.7: Plug Motor Shield Back Into Arduino

9. This completes the case attachment for the Arduino.

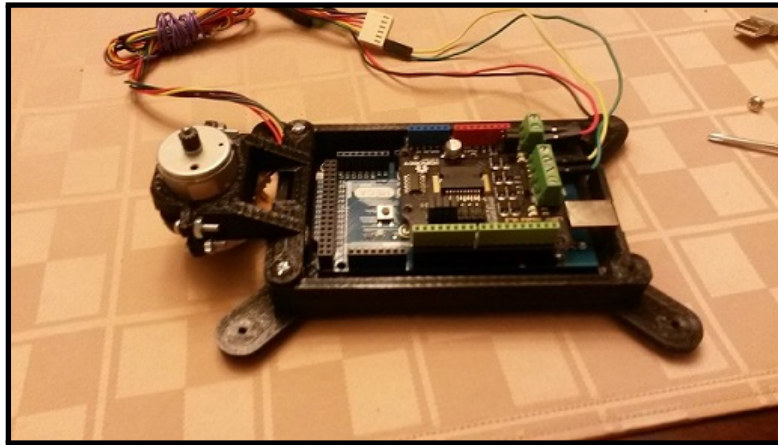


Figure 5.8: Completed Case Setup

## Connecting Encoder to Arduino

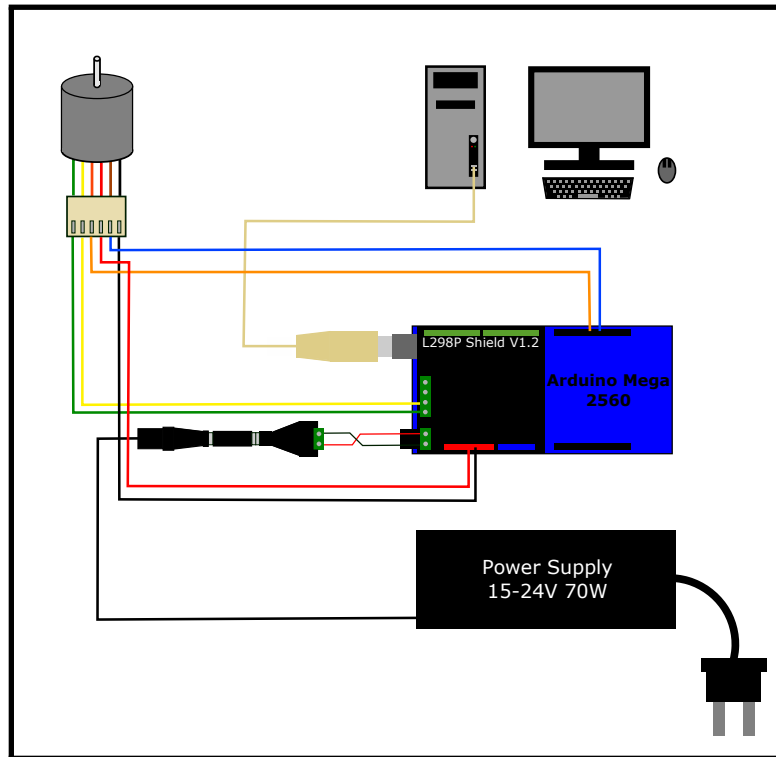


Figure 5.9: Circuit Diagram for Simple DC Motor Hardware

10. Connect the “PWR” and “GND” ports of the motor encoder to pins “5V” and “GND” on the Arduino, respectively, using two wires. There are two “GND” pins located on the red port strip of the motor shield. Connect to the GND pin closest to the “5V” pin.



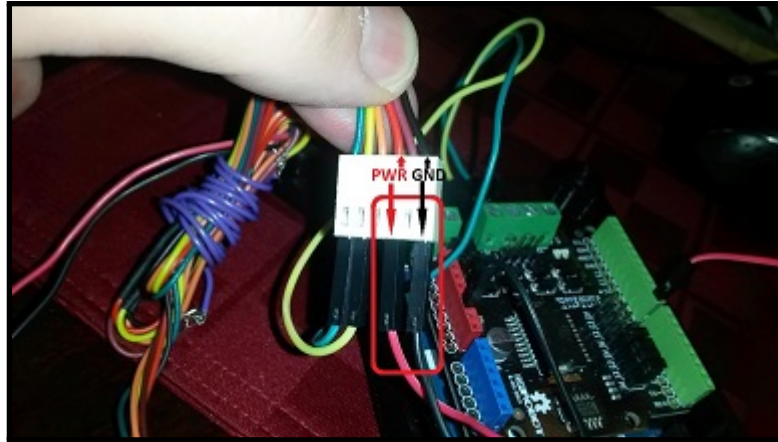


Figure 5.10: Connecting Power and Ground of Encoder to Arduino (Motor Connector Side)

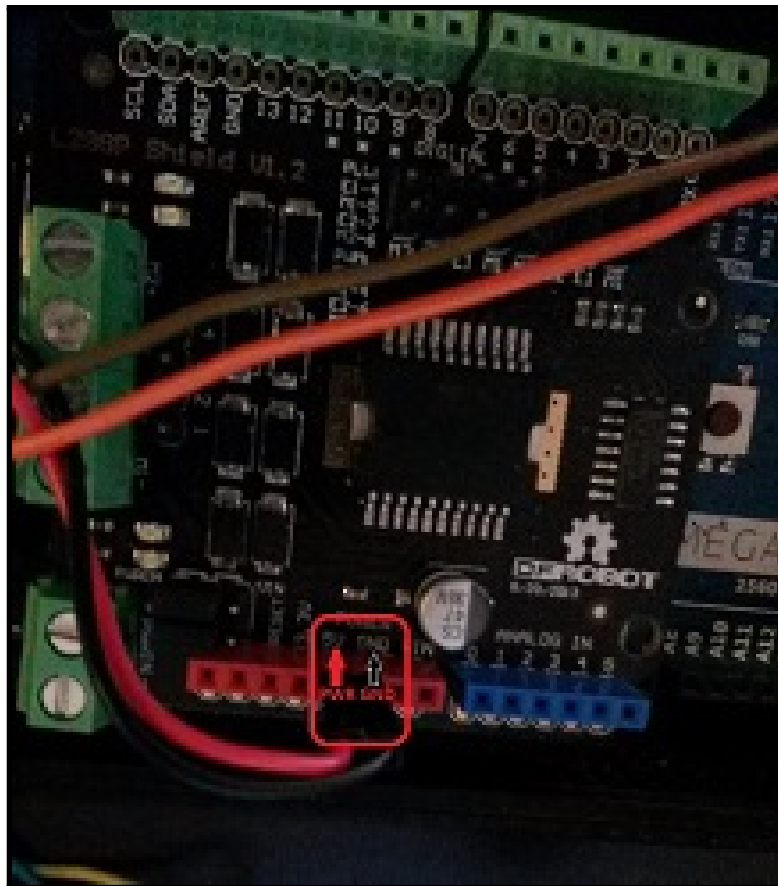


Figure 5.11: Connecting Power and Ground of Encoder to Arduino (Motor Shield Side)

11. Connect the “EN-A” and “EN-B” ports of the Mitsumi Motor to ports 18 and 19 on the Arduino, respectively, using the last two wires.

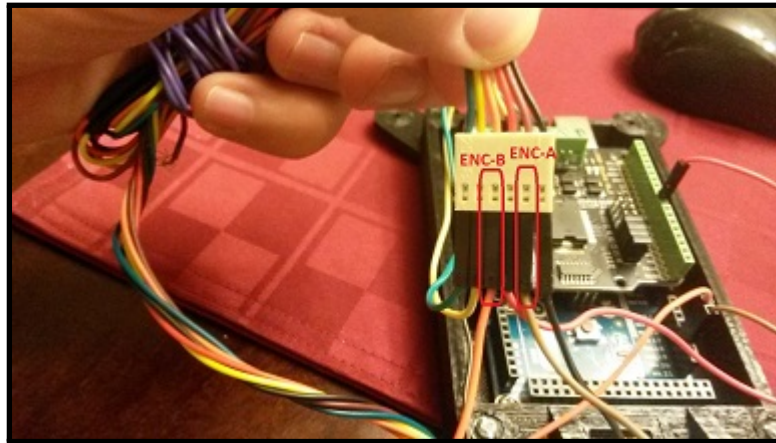


Figure 5.12: Connecting Encoder A and B ports to pins 18 and 19 on the Arduino(Motor Encoder Side)

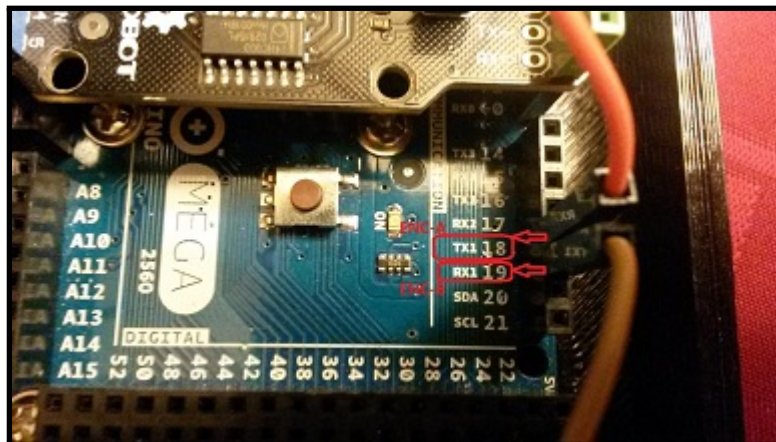


Figure 5.13: Connecting Encoder A and B ports to pins 18 and 19 on the Arduino (Motor Shield Side)

### Attaching Insert and Load to Motor

12. Obtain the gear insert and motor load from the *Introduction to 3D Printing* Laboratory.



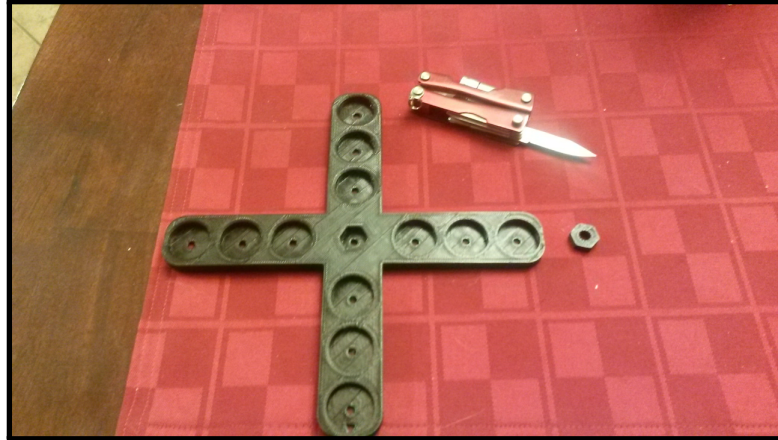


Figure 5.14: 3D Printed Parts From *Introduction to 3D Printing* Experiment

13. The professor or laboratory director for this experiment will tell you how many pennies to place in each slot. Insert the pennies into the slots you are given and then tape them over so that the pennies do not fall out during the experimentation portion of this laboratory.



Figure 5.15: Pennies Inserted Into Mass (Suspended with Scotch Tape)

14. Push the gear (on the motor shaft) into the gear insert. There are notches that the gear teeth will fit into. The more open they are, the easier the insert will fit on the gear. If the insert does not fit on the gear at first, take a knife and cut out each notch (inside the insert) until each notch is wider. Remove the gear insert from the gear once it fits.

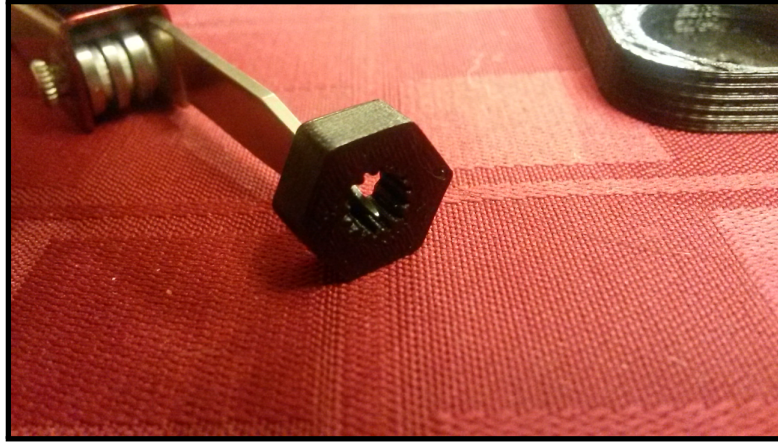


Figure 5.16: Opening Notches of Gear Insert With Knife

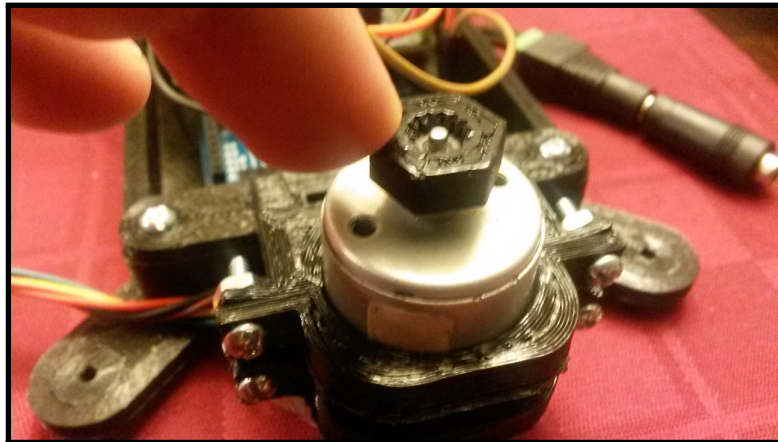


Figure 5.17: Gear Pushed Into Gear Insert

15. Insert the gear insert into the center of the 3D printed mass. Ensure that the hexagonal shape of the gear insert fits into the center of the mass. (This may require that you use sand paper to sand down the sides of the gear insert. Do not sand too much, as the insert needs to fit snugly. Fine grain or 400 grit sandpaper is highly recommended.)



Figure 5.18: Sanding Gear Insert

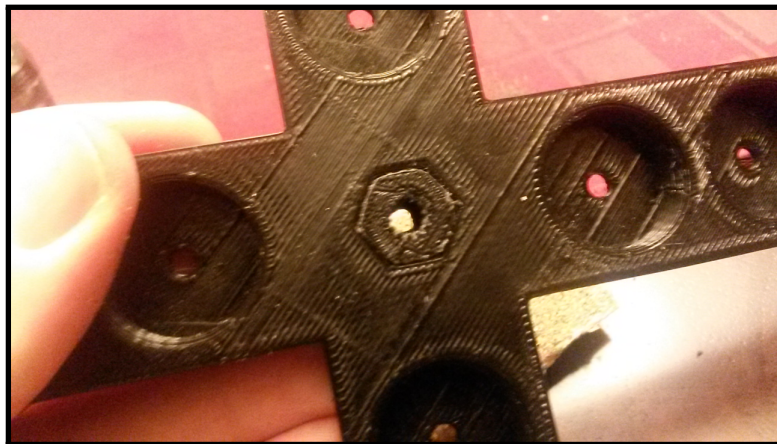


Figure 5.19: Gear Insert Inside 3D Printed Mass

16. Push the gear into the gear insert/3D printed load combination. Make sure that the plane of the load is orthogonal to the gear shaft (parallel with the table the project is on).



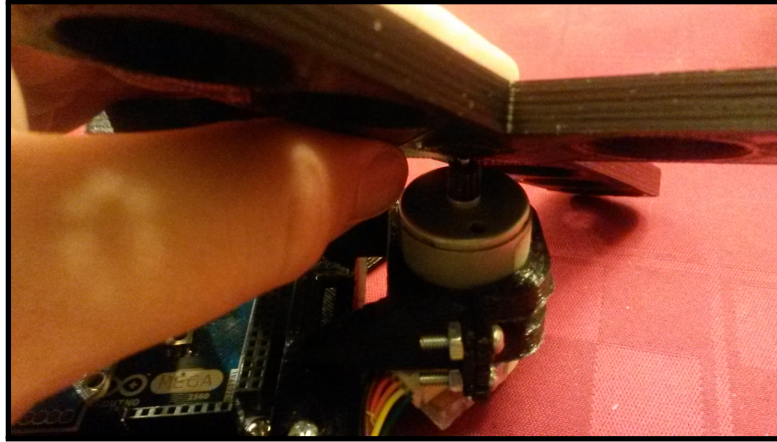


Figure 5.20: Gear Pushed into Mass/Gear Insert Combination

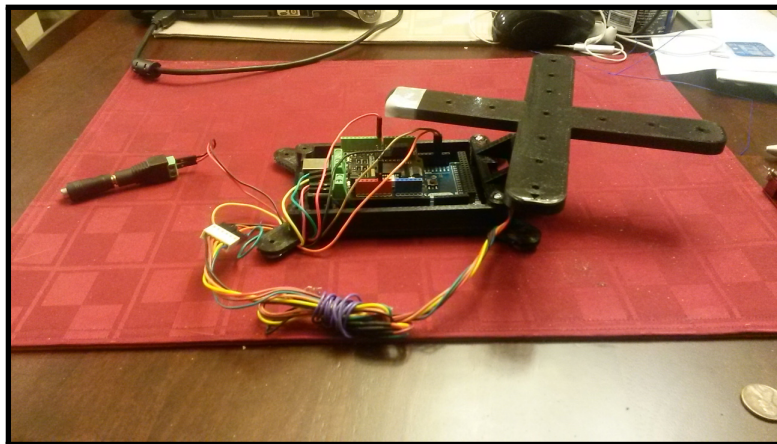


Figure 5.21: Completed Load Setup

### Mounting Full Hardware Setup to Flat Surface

17. Find a flat surface (on top of a desk or table.) Wipe off the surface to eliminate any loose particles that may be on it.
18. Obtain sticky tack. Roll the sticky tack into 5 different balls about .5 inches in diameter.



Figure 5.22: Sticky Tack Rolled Balls

19. Take one of the balls and split it into two equal halves. Take each half and roll them into a long, cylindrical strands.



Figure 5.23: Cylindrical-shaped Sticky Tack

20. Place the sticky tack to the four corners of the case and the two strands to the bottom of the encoder on the motor.

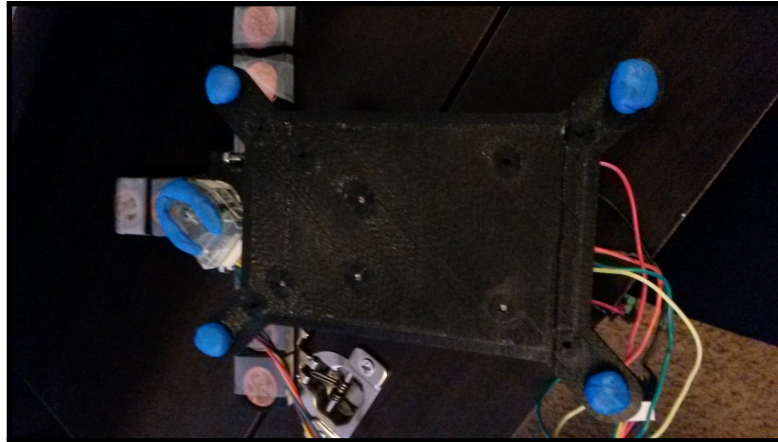


Figure 5.24: Sticky Tack On The Bottom of the Case and Motor Encoder

**CAUTION:** Do not place the sticky tack directly below the motor shaft. The tack can easily attach to the motor shaft and inhibit the motion by adding unwanted friction.

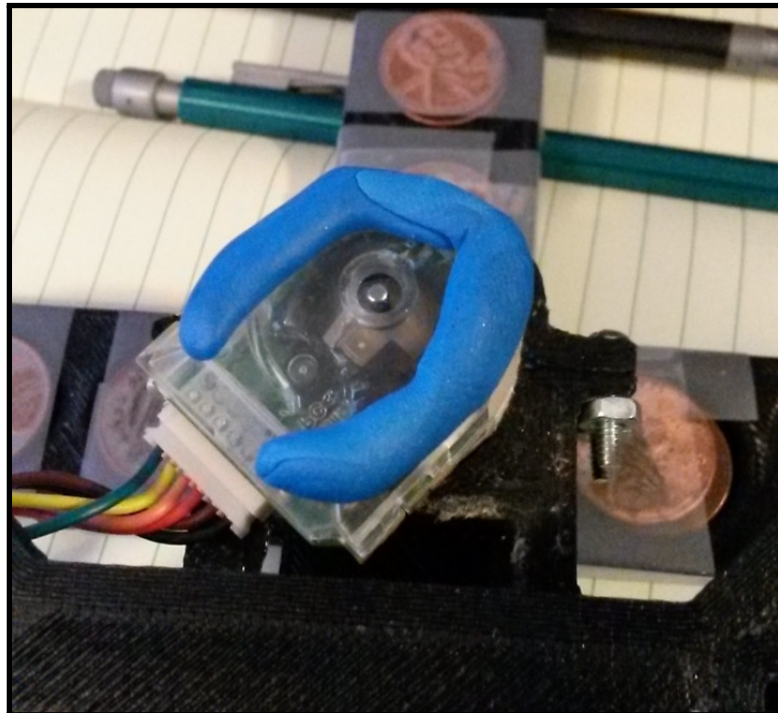


Figure 5.25: Correct Sticky Tack Placement on Motor Encoder

21. Take the wires going to the Motor Encoder and wrap them under the leg of the case closest to the power on the motor shield (This will keep the wires together

and away from the moving motor load.) After the wires are under the leg, press all four corners of the case and the bottom of the motor encoder to a flat surface.

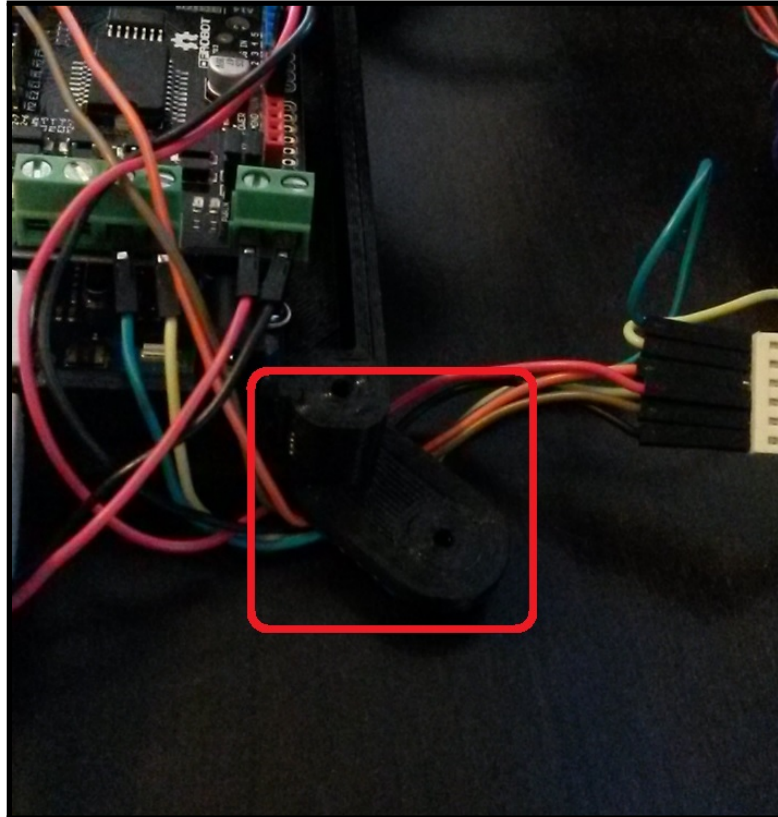


Figure 5.26: Wrap Wires Under Leg of Case



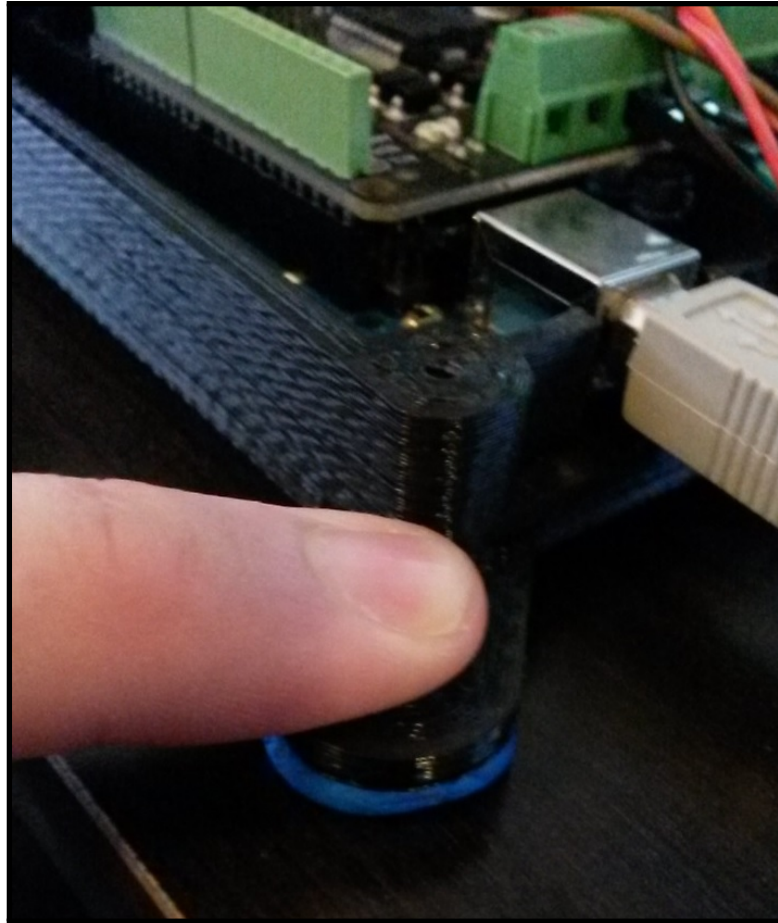


Figure 5.27: Press Each Corner of the Case and the Motor Encoder to Flat Surface

22. Plug the USB B connector back into the Arduino.



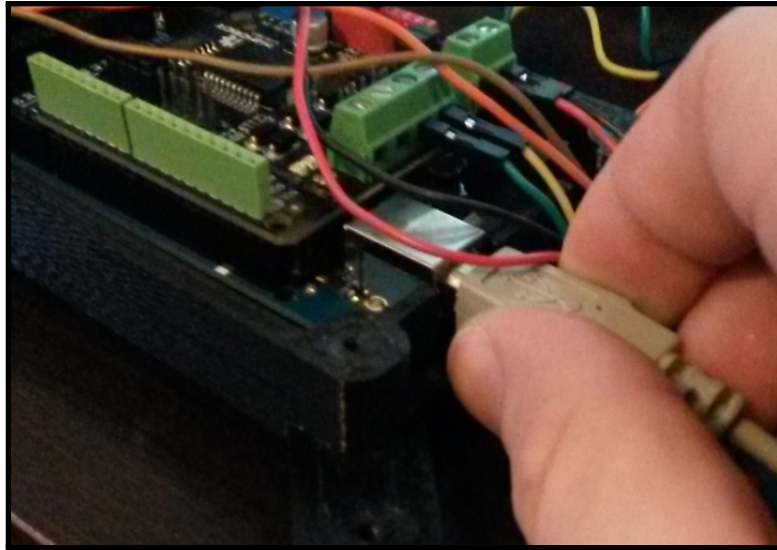


Figure 5.28: Plug USB B Connector Into Arduino

23. The hardware setup is now complete.

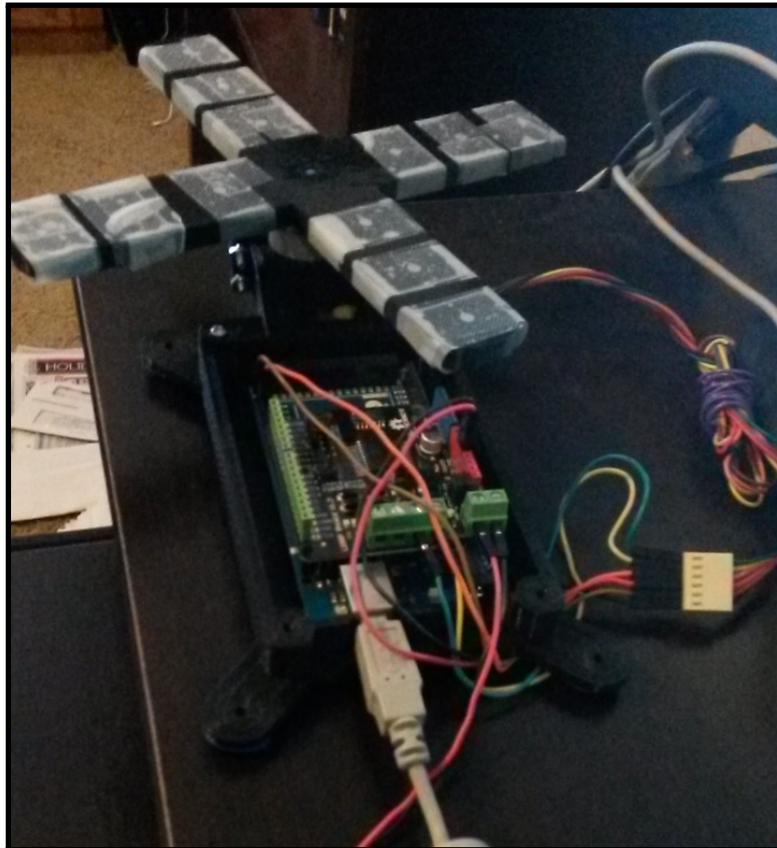


Figure 5.29: Completed Hardware Setup

## 5.3 Experimental Procedures

### 5.3.1 Exercise 1: Deriving Motor Transfer Function

Figure 5.30 below is the circuit diagram for the DC motor:

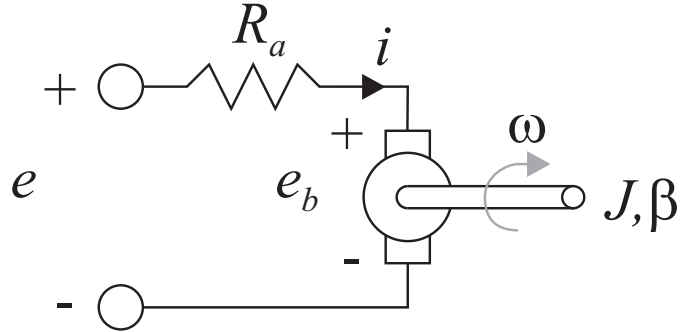


Figure 5.30: Circuit Diagram for DC Motor

The equations of motion for the system are:

$$e = R_a i + e_b \quad (5.1)$$

$$J\dot{\omega} = \tau - \beta\omega \quad (5.2)$$

$$\tau = K_t i \quad (5.3)$$

$$e_b = K_b \omega \quad (5.4)$$

where  $J$  is the inertia of the armature and the load,  $\beta$  is the viscous friction coefficient,  $K_t$  is the torque constant,  $R_a$  is the armature resistance,  $K_b$  is the back emf constant,  $\tau$  is the torque,  $i$  is the armature current,  $e$  is the voltage applied to the motor,  $\omega$  is the angular velocity of the motor, and  $e_b$  is the motor back emf.

24. Solve for the transfer functions  $G_1, G_2, G_3,$  and  $G_4$  in Figure 5.31, using Equations (5.1) - (5.4) .

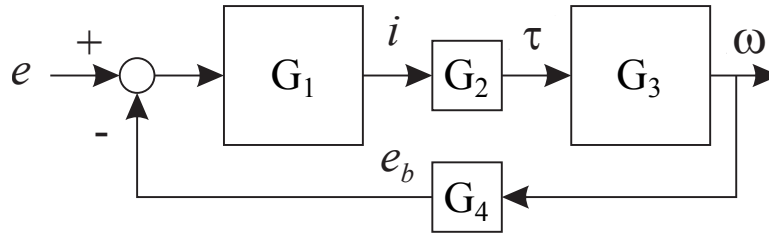


Figure 5.31: Empty Block Diagram for Motor

25. Using block diagram reduction, find the overall open loop transfer function  $\frac{\Omega(s)}{E(s)}$ .

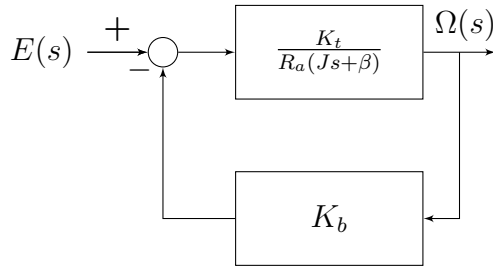
### Expected Hand-Written Results (For Instructors)

Students are first asked in **step 24** to solve for the transfer functions that go into blocks  $G_1, G_2, G_3$ , and  $G_4$  from Figure 5.4. To solve this, we first take the Laplace Transform Equations 5.1 - 5.4. After that, the correct transfer functions for each block will look like the values in Table 5.1 below:

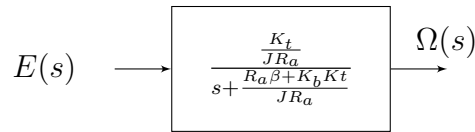
Table 5.1: Transfer Function Values for G blocks

Block	Value
$G_1$	$\frac{1}{R_a}$
$G_2$	$K_t$
$G_3$	$\frac{1}{Js+\beta}$
$G_4$	$K_b$

Students are also asked to use block reduction and solve for the overall transfer function  $\frac{\Omega(s)}{E(s)}$  in **step 25**. First, we combine blocks  $G_1, G_2$ , and  $G_3$  we have the following result:



We can further reduce this block diagram using the negative feedback loop rule. The result is the solution of the transfer function  $\frac{\Omega(s)}{E(s)}$  is in the block diagram below:



### 5.3.2 Exercise 2: Theoretical Open Loop Step Response

In this exercise you will derive the theoretical open loop step response for the DC motor. In later exercises, you will compare this theoretical response to experimental and simulated step responses.

26. Arrange your open loop transfer function  $G(s) = \frac{\Omega(s)}{E(s)}$  from the previous step in the form

$$G(s) = \frac{Y(s)}{U(s)} = \frac{K_m}{\tau_m s + 1} \quad (5.5)$$

27.  $K_m$  is the open loop DC gain, and  $\tau_m$  is the time constant. Assume that a step input of  $A$  volts is applied to the motor, derive the motor velocity  $\omega(t)$  using partial fraction expansions, and plot the velocity versus time. Your step response should be a function of  $K_m$  and  $\tau_m$ .

28. How does  $K_m$  affect the step response? How does  $\tau_m$  affect the step response? In a later exercise you will find the step response of your DC motor experimentally, and you will use the step response to determine  $K_m$  and  $\tau_m$ .

### 5.3.3 Exercise 3: Open Loop Step Response Variables

To produce the open loop step response (both experimentally and in simulation), you will need to create a Matlab file (used to store variables into the Matlab workspace) before running Simulink. Follow the steps below to create this Matlab file.

#### Setting Up Matlab File

29. Open Matlab 2014b. In the top left-hand corner of the main Matlab page, click the button labeled “New Script” (See Figure 5.32).

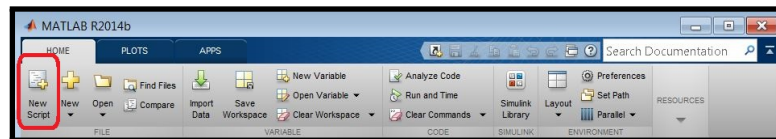



Figure 5.32: New Script Button Location on Main Matlab Page

30. A page labeled “Editor-Untitled” should be visible. This is the Matlab “m-file” editor. Save the m-file by clicking the **Save** button  at the top left hand corner.

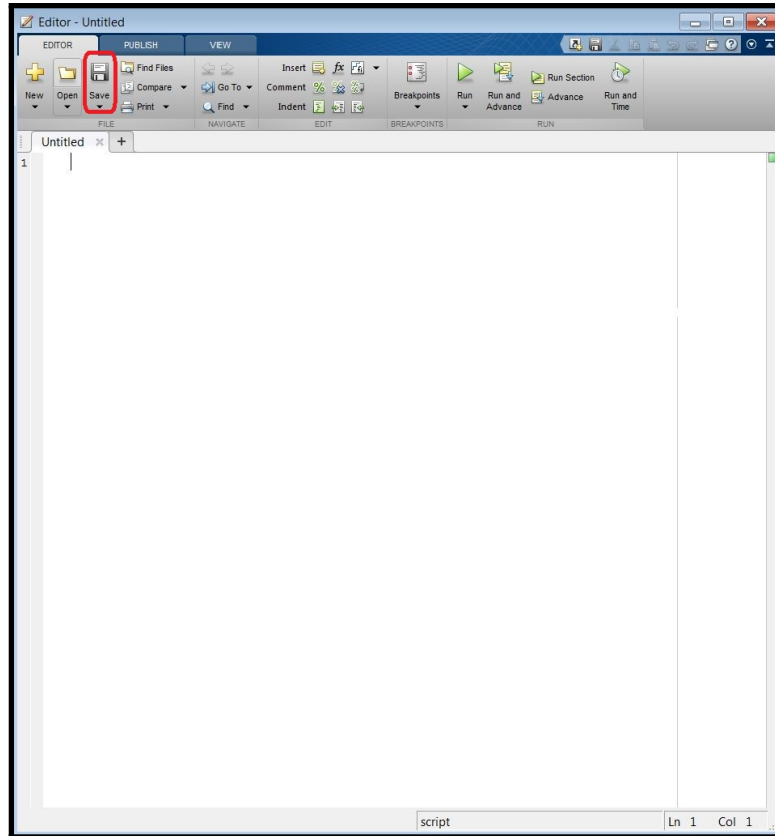


Figure 5.33: Save Button Location on m-file Editor Page

31. A page labeled “Select File for Save As” should now be visible. Navigate the browser to the directory where your previous experiments were saved. Type **OL\_Constants.m** as the File name. Save the file in this directory by clicking **Save** at the bottom.
32. Type **Ts = 0.01;** (See Figure 5.34). This will be the sampling time.

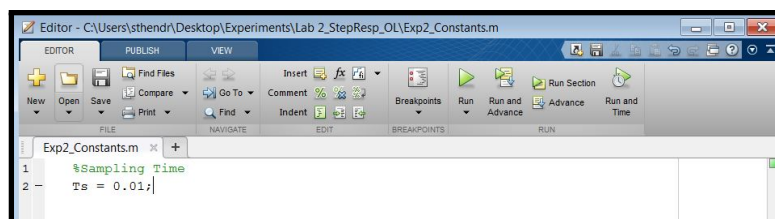


Figure 5.34: Sampling Time Variable “Ts” Added to m-file


33. Type **rp = 2\*pi/1336;** the Mitsumi motor/encoder has a resolution of 1336

counts per revolution. This means that every time the motor has completed a full revolution ( $2\pi$  radians), the encoder has provided 1336 counts. Multiplying the output of the encoder count by **rp** produces the angular position of the motor shaft (in radians).

34. Type **Vmax = 15;** this will be the maximum voltage used to power the motor.
35. Type **v2d = 255/Vmax;** this variable will be used to convert the motor input voltage into a PWM duty cycle percentage. Figure 5.35 shows the completed lines of code used in the Exp2\_Constants.m file.

```
11 %Sampling Time
12 - Ts = 0.01;|
13
14 %Encoder Resolution
15 - rp = 2*pi/1336;
16
17 %Supplied Voltage
18 - Vmax = 15;
19
20 %Gain used to convert voltages into duty cycles
21 - v2d = 255/Vmax;
```

Figure 5.35: Final Matlab Variables

36. Save the file again by clicking **Save**  at the top of the page (as in Step 27). This completes the Matlab file setup.

#### 5.3.4 Exercise 4: Experimental Open Loop Step Response

In this exercise, you will create an experimental Simulink model to load to the Arduino. We call it experimental because it will be loaded to the Arduino's memory and run to collect data from the open loop step response of the motor. This open loop step response data will then be interpreted in the next exercises to find the appropriate  $K_m$  and  $\tau_m$  values.

## Setting Up Simulink File (Arduino)

37. Open the Simulink model “SimpleDCMotor.slx” that was created in the *Simple DC Motor* laboratory.
38. Click on Model Configuration Parameters. Once the Configuration Parameter page is visible, change the “Fixed-step size (fundamental sample time):” to **Ts**. Click Apply and then OK.
39. In the Simple DC Motor Simulink model, double-click on the “Direction” block and change the “Constant value:” to **1**. Also, double-click on the “Voltage” block and change the “Constant value:” to **5**. Finally, double-click on the “Voltage to PWM” gain block and change the “Gain:” to v2d. The final fixes to these blocks should look like Figure 5.36.

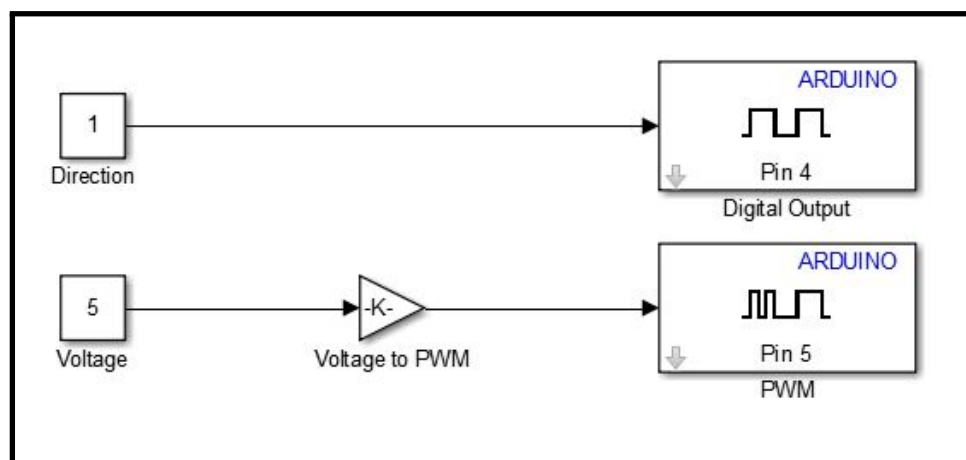


Figure 5.36: Modified Simulink Model from Simple DC Motor

40. Navigate to the Library Browser and add a **Constant** and **Digital Output** block to the Simulink window.
41. Rename the Constant block to “Encoder Power”. Change the “Constant value:” in the Encoder Power block to **1**.
42. Add a **Dual Encoder Positions** block, 2 **Gain** blocks, a **Difference** block,



and a **Discrete Filter** block to the model. Rename one Gain block to “Steps to Radians” and the other to “Velocity Scaling”. Leave the other two blocks with their default names.

43. Double-click on the **Steps to Radians** block and type **rp** into the “Gain:” box and click OK. Double-click on the **Velocity Scaling** block and type **1/Ts** into the “Gain:” box and click OK. Double-click on the **Discrete Filter** block, and under “Numerator” input the value **0.1**. In “Denominator:” type **[1 -0.9]** and then click OK. The Discrete Filter block will smooth the velocity measurements from the encoder output.
44. Connect the output “Enc1 Position” of the **Dual Encoder Positions** block to the input of the **Steps to Radians** block. Connect the output of the **Steps to Radians** block to the input of the **Difference** block. Connect the output of the **Difference** block to the input of the **Velocity Scaling** block. Connect the output of the **Velocity Scaling** block to the input of the **Discrete Filter** block.

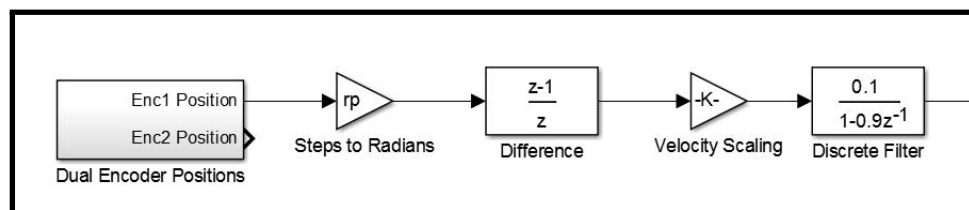


Figure 5.37: Encoder Output Position to Velocity

45. Now add a **Data Type Conversion** block. Double-click on the block. Next to “Output data type:” click the drop-down menu and choose the option “single”. Click Okay and attach the output of the Discrete Filter to the input of the Data Type Conversion block.
46. In the Library Browser, navigate to the “Rensselaer Arduino Support Package” title. Right-click over the name and click “Open Rensselaer Arduino Support

Package Library.” In the Library, left-click and drag the **Serial Send single Port0** block over to your Simulink model. Additionally, left-click and drag the text **Plot Data ‘single’** over to the model and place it under the Serial Send single Port0 block.

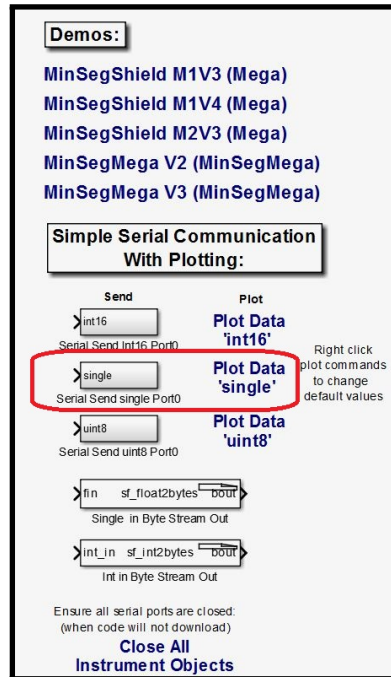


Figure 5.38: **Serial Send single Port 0** and **Plot Data ‘single’** Block and Text Locations, Respectively

47. Connect the output of the **Data Type Conversion** block to the input of the **Serial Send single Port0** block. (When the model is running on the Arduino, double-clicking on the text will open up the serial port and allow you to read the data from the serial port.)
48. Now the Simulink file that will be deployed to the Arduino is complete (see Figure 5.39 for the final Simulink model). Before loading this model to the Arduino, ensure that the Arduino is connected to your computer and the power supply is connected to the motor shield. Save this file as **OL.Step\_Resp\_Arduino.slx** to the same folder where **OL\_Constants.m** is saved.

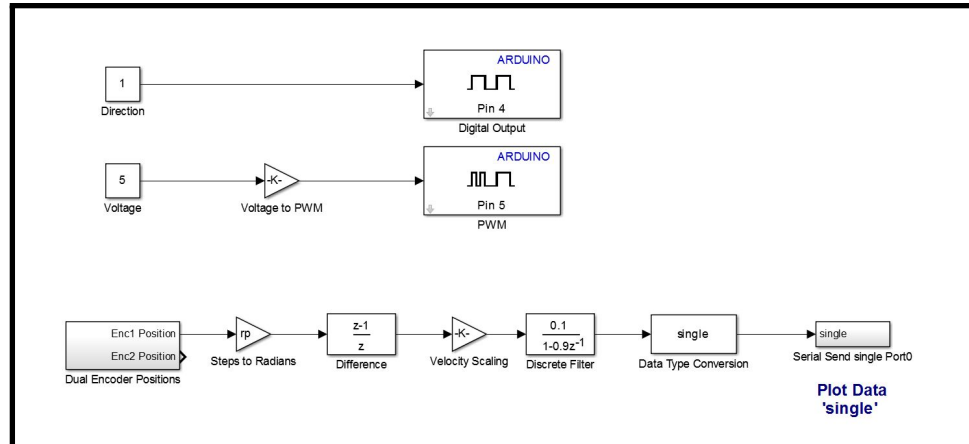




Figure 5.39: Final Simulink Diagram to Load on Arduino

## Collecting Experimental Data

The steps below will explain the common method for capturing data from the Arduino. If you run into issues, refer back to the *Sampling and Data Acquisition* experiment in Exercise 2 steps 78 - 82.

49. Open **OL\_Constants.m** and click the Run button  at the top of the page.
50. Open **OL\_Step\_Resp\_Arduino.slx** and click the “Deploy to Hardware” button  at the top-right of the page.
51. Once the model has successfully deployed to the Arduino, double click on the text “Plot Data ‘single’” inside the model window.
52. When the small window labeled “Plot Ser...” appears, enter the Arduino COM port number under “Enter COM port to collect data:.” Also under “Enter Number of Samples to plot:” type **single** and for “Enter Number of samples to plot:” type **2000**.

**NOTE:** To find the COM port number for your Arduino, refer to the *Simple DC Motor* experiment under the section “Software Setup → Installing Arduino Mega 2560 Drivers,” specifically steps 20-24.


53. Click Okay. Once the plot appears, plug the power cord from the power supply into the motor shield.

**CAUTION:** Do not put your hands or any other parts of your body in front of the motor load's trajectory. Also, if the load does not begin spinning once the motor shield is plugged in, immediately unplug the power and check to see if everything is connected properly (review the "Hardware Setup" section in this experiment for the proper hardware connections.)

54. Observe the plot. If the plot appears to be very jumpy (meaning that the values do not look to be "smooth" and vary from extremely positive to negative values) or if the values are not changing from zero, proceed to the next step. If the plot appears to be smooth (rising to a value and staying around that value) then **skip to step 55** of this section; otherwise, **continue to the next step**.
55. **Unplug the power** from the motor shield, ensuring the motor load has come to a complete stop. **Press the reset button**, which should be located under the motor load trajectory on the Arduino.
56. Once the Arduino has fully reset, the plotting window should appear to output a value of zero (flat line). If this is not the case, press reset once more until the plot displays zero.
57. Plug the power cable back into the motor shield. If the data appears to be increasing smoothly (not varying to very large and small values), **continue to the next step**. If the data is still not smooth and largely varying, **repeat steps 52-53**.
58. Let the data fill the plot window as it moves to the left in the plot window. Once the data (starting with a value of zero) has filled the plot window completely,

click the “Stop” button at the bottom of the screen. You should now have 2000 data points inside the window (which is 20 seconds.)

**NOTE:** you should expect to see a curve beginning at zero, rising up, and settling to the steady-state value. Try to make the first value of the plot be equal to zero by clicking stop once the zero velocity point crosses the origin.

59. In the upper left-hand corner of the figure click “File” and click “Save As.”
60. Save the figure as **OL\_data.fig** in the same folder as all of the other files you have created in this experiment and click Okay. Close the figure. Another “Plot Ser...” window should appear. Click the close button  on the window.
61. Navigate over to the main Matlab window. Under the Workspace, find the variable **WindowDat**, right-click it and click “Save As”. Name the file **OL\_data.mat** and save it into the same folder where you saved **OL\_data.fig**.
62. You now have the experimental data for the open loop step response of this experiment.

### 5.3.5 Exercise 4: Find Transfer Function from Experimental Data

#### Experimental Plotting File




63. Open the main Matlab 2014b window and click **New** at the top and then click **Script**.
64. Once the new Untitled m-file appears, Click **Save**  at the top of the page. Save the file as **OL\_Plot.m**.
65. Copy and paste the text in Table 5.2 into the Matlab file. After adding the code click **Save**  and then click **Run** .

Table 5.2: Code for Plotting Experimental Results

---

```

%Load the experimental data and store into a variable
Exp_dat = load('OL_data.mat');
Exp_dat = OL_data.WindowDat;

%Set the time vector to be the same length as Exp_dat (20 Seconds)
T = 0:Ts:20;

%Plot the Experimental Data with Respect to Time
figure(1);
plot(T,OL_data, ...
'Color', 'r', ...
'LineWidth', 1);
title('Experimental Open Loop Step Response');           %Title
legend('Experimental', 'Location', 'northwest');        %Legend
xlabel('Time (seconds)')                                 % x-axis label
ylabel('w (radians/second)')                            % y-axis label

```

---

66. Calculate the time constant ( $\tau_m$ ) from the angular velocity vs. time plot found in the previous step.
67. Calculate the DC gain ( $K_m$ ) from the experimental plot. (Remember that, for this experiment, the step magnitude was  $A = 5$ .)
68. Using the calculated ( $\tau_m$ ) and ( $K_m$ ), find the first order transfer function. Use the following form for the transfer function:

$$G(s) = \frac{Y(s)}{U(s)} = \frac{K_m}{\tau_m s + 1} \quad (5.6)$$


where  $K_m$  is the steady-state value and  $\tau_s$  is the time constant.

69. Compare the transfer function found in Equation 5.6 with the transfer function found using the block diagram reduction in Exercise 1. How are the transfer functions related? Find expressions for  $K_m$  and  $\tau_m$  in terms of  $K_t$ ,  $K_b$ ,  $J$ ,  $\beta$  and  $R_a$ .
70. Given that  $K_t = 0.0267 \frac{Nm}{A}$ ,  $K_b = 0.0269 \frac{V}{rad/s}$  and  $R_a = 13.5 \Omega$ , calculate the remaining motor parameters ( $\beta$  and  $J$ ).
71. You may wish to rerun the experiment to see if you get consistent values for  $K_m$  and  $\tau_m$ . If the results are not consistent, explain why that could be.

### 5.3.6 Exercise 6: Compare Simulation and Experimental Results

Typical engineering practices involve first simulation followed by experimental testing on a real system. You are unable to simulate the model for the motor and load combination without parameters. For this experiment, the experimental values are obtained first in order to find the model parameters for simulation. Now that those values have been found, a Simulink model for simulation will be setup. The simulated results will then be compared with the experimental results by overlaying each plot on top of one another in the same figure. Steps explaining the setup of the Simulink simulation file will be explained, and then you will be given steps to take the data from the simulation and plot it in the same figure as the experimental results.

## Setting Up Simulink File (Simulation)

72. With MATLAB R2014b open, click the button labeled “New” to reveal the dropdown menu. Under the blue bar labeled “Simulink” click the button labeled “Simulink Model”.
73. With a new Simulink Model labeled “untitled” visible, click on “File” at the top left hand corner of the page. In the drop-down menu, click “Save As...”
74. In the “Save As” page, navigate to the folder where you have saved the Matlab file created from section **Setting Up Matlab File**. Name the file **OL\_Simulation.slx** and click **Save**.
75. Click the **Model Configuration Parameters** icon  at the top of the page.
76. On the left column of the page click **Solver** (“Simulation time” should appear at the top of the window.)
77. Change the “Stop time:” to **20**.
78. Next to “Type:” click the drop down menu and choose **Fixed-step**. Once available, change the value for “Fixed-step size (fundamental sample time):” to **Ts** and then click “Okay.”
79. In the Simulink model, open the Library Browser and place a **Constants** block. Double-click on the Constants block and change the value to **5**. Change the name of the block from “Constants” to “Input Voltage”.
80. Add an **LTI System** block to the Simulink model. Double-click on the block and add the value **tf(Km/5,[tau 1])** to the box labeled “LTI system variable” and then click OK (See Figure 5.41).



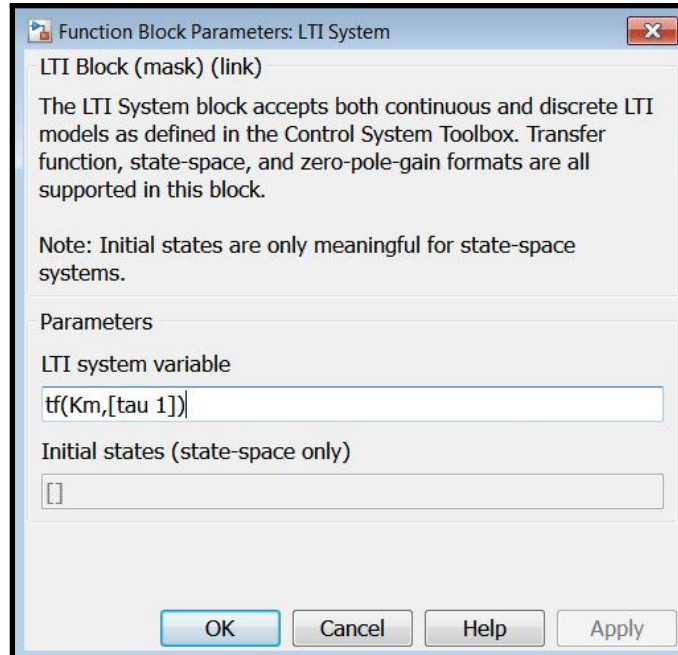


Figure 5.40: Function Block Parameters

**NOTE:** Km is divided by 5 in order to take out the scaling effect that the 5 volt input creates. Without dividing by 5, the output of the loop step response will be 5 times as large as we want it to be. Also, once you have typed in the information into the **LTI System** block, the text on the block may show up with question marks ???, if there is simply not enough room to display the text on the block.

81. Add a **Scope** block to the Simulink model. Connect the input of the Scope block to the output of the LTI System block.
82. Add a **To Workspace** block to the Simulink model. Connect the output of the LTI System block to the input of the **To Workspace** block.
83. The Simulink simulation model is now complete (see Figure 5.41).

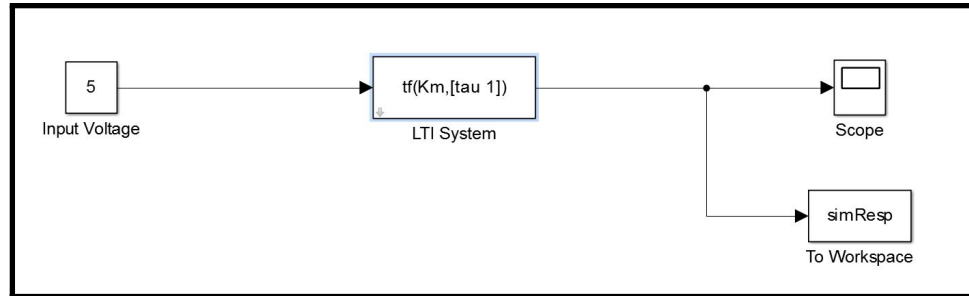


Figure 5.41: Final Simulink Diagram to be Simulated and Compared With the Experimental Results

### Collecting Simulation Data


84. Open the **OL\_Constants.m** file created in Exercise 1.
85. Add the lines of code found in Figure 5.42 to the bottom of the m-file.

```

16     %Observed Km and tau from experimental results
17 -   Km = 0; %*** REPLACE 0 WITH Km VALUE
18 -   tau = 0; %*** REPLACE 0 WITH tau VALUE



```

Figure 5.42: Add These Two Lines to Code


86. Replace each **0** value in the **OL\_Constants.m** file (shown in Figure 5.42) with the values you found for **Km** and **tau** in steps 64 and 65.
87. Save the file and then press the “Run” button  at the top of the page. Navigate to the “MATLAB 2014b” home page. Under “Workspace” on the right-hand side of the page, all of the variables from “OL\_Constants.m” should be listed below.

**NOTE:** if any variables created in the “Setting Up Matlab File” section are not listed under “Workspace,” simply add the missing variables to the bottom of the **OL\_Constants.m** file and click **Save** once more. If any variables are

missing from the OL\_Constants.m file, the “OL\_Simulation.slx” file will not run correctly.

88. Open **OL\_Simulation.slx**. Click the Run button  at the top of the page.
89. Once the model has finished running, double-click on the **Scope** block. Click the **Autoscale** button .

**NOTE:** Observe the plot; does the response look similar to the response found in step 55? If not, check the **Km** and **tau** values you found in steps 64 and 65. For now, the exact **Km** and **tau** values do not matter, just the general shape of the plot (which should be increasing until it reaches a steady state value, where it stays constant.)

90. Navigate back to the “MATLAB 2014b” home page. Under “Workspace” a new variable **simResp** should now be available. Right click on **simResp** and click “Save As...” Navigate to the folder in which you have saved this project, name the “File name:” as **Sim\_1.mat**, and click **Save**  at the bottom of the page.

### **Experimental vs. Simulation Comparison Matlab File**


91. Open the **OL\_Plot.m** file you created in steps 63 - 71.
92. Copy and paste the text from Table 5.3 to the bottom of **OL\_Plot.m**. Click **Save** .


Table 5.3: Code for Plotting Simulation Results (Added to **OL\_Plot.m**)

---

```
%Load the simulated data and store into a variable
Sim_dat = load('OL_data.mat');
Sim_dat = Sim_dat.simResp.Data;

%Plot the Simulated Data with Respect to Time
hold on;
plot(T,Sim_dat, ...
'Color', 'k', ...
'LineWidth', 2);
legend('Experimental', 'Simulation', 'Location', 'northwest');
```

---

93. Click **Run**  at the top of the page.
94. Compare the responses of the simulation with the experimental data. Does the simulated response match the experimental response? Describe the similarities and differences. **Write down the current value for Km and tau on a piece of paper to keep track of them.**
95. What would cause the differences between the responses?
96. If the plots look different, **change the values for Km and tau** to match the simulation with the experimental open loop step response. After finding the new values, **change the Km and tau values in the OL\_Constants.m file**. After changing the **OL\_Constants.m** file, **follow steps 84 - 93 once more**. If the simulated and experimental open loop step responses look almost the same, **continue to the next step**.

97. How much (if any) did the new values for  $K_m$  and  $\tau_m$  change from your original values? For the new values of  $K_m$  and  $\tau_m$ , repeat step 67. How much did the values for  $J$  and  $\beta$  change?

### Expected Experimental vs. Simulation Results (For Instructors)

Since every student in the “Dynamic Systems or Control” course will be given a different number of pennies to put into the load, two extreme cases for the open loop step response were tested - the load with a full complement of pennies and the load without pennies. The open loop step response was tested 3 different times for each case, and the following results were observed:

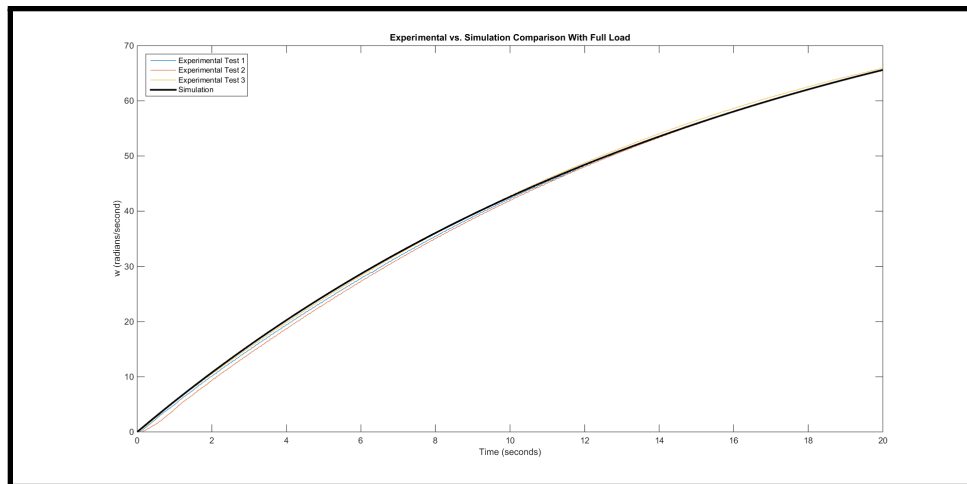


Figure 5.43: Three Open Loop Step Responses With Full Pennies in Load (Using  $K_m = 18.5 \frac{rad}{Vs}$  and  $\tau_m = 16.2 s$  For Simulation)

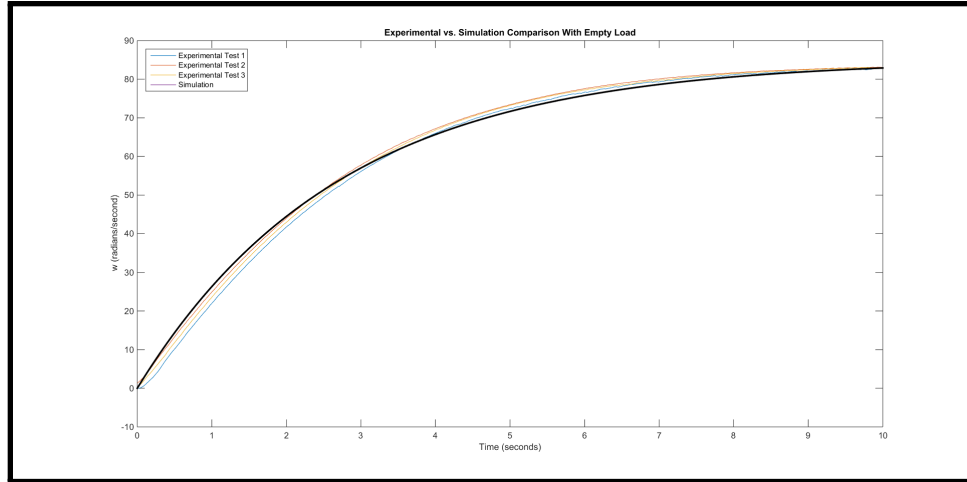


Figure 5.44: Three 10 Second Open Loop Step Responses With No Pennies in Load (Using  $\mathbf{Km} = 17 \frac{rad}{Vs}$  and  $\mathbf{tau} = 2.7 s$  For Simulation)

In order to better observe the steady state value for the full penny case, the open loop step response plot ran for 20 seconds, as opposed to 10 seconds for the load without pennies. The major change in response is the settling time. The settling time is about 10 seconds longer for the load with full pennies. Expect for students running this lab to observe a steady state value of  $\mathbf{Km}$  equal to around 20 radians/second and a settling time  $\mathbf{tau}$  anywhere between 2 and 13 seconds, depending on the number of pennies used.

In step 67 students are asked if they can solve for  $\beta$  and  $J$  given that constants  $K_t = 0.0267 \frac{Nm}{A}$ ,  $K_b = 0.0269 \frac{V}{rad/s}$  and  $R_a = 13.5 \Omega$ . They can be solved by setting  $G(s)$  in **Equation 5.6** equal to the transfer function found for  $\frac{\Omega(s)}{E(s)}$ . We found the simplified version of  $\frac{\Omega(s)}{E(s)}$  in Exercise 1 under the section labeled **Expected Hand-Written Results (For Instructors)**. So all we need to do is set them equal to each other. Through calculations, it was found that for the **Full Penny Load** case,  $\beta = 0.000054$  and  $J = 0.001732 \frac{Kg}{m^2}$ . For the **Empty Penny Load** case,  $\beta = 0.000198$  and  $J = 0.000314 \frac{Kg}{m^2}$ . As you can see, the inertia increased with the full penny load case compared with empty penny load case.  $\beta$ , the viscous friction coefficient, appears to

quadruple when the load is empty.

#### **5.4 Conclusion/Student Feedback**

This chapter provided an undergraduate level (Systems) approach to analyzing the open loop step response of a DC motor. Both experimental and simulated responses were generated and compared. This experiment utilized the *Simple DC Motor*, *Serial Communications Laboratory*, *Sampling and Data Acquisition*, and *Introduction to 3D Printing* experiments.

## CHAPTER 6

### Closed Loop Step Response

#### 6.1 Objective

This chapter describes an experiment that adds feedback to the *Open Loop Step Response* experiment. Students will find the poles of the open loop motor model from the *Open Loop Step Response* experiment. The objective is for students to observe the effects of a proportional-derivative controller that will produce under-damped and over-damped closed loop step responses. Students will find the closed loop poles of the system and compare them with the open loop poles. Two sets of gains corresponding to under-damped and over-damped responses will be given and students will simulate the closed loop step responses corresponding to those gains. The experimental response will then be found and compared with the simulated response.

#### 6.2 Setup

##### 6.2.1 Required Materials

##### Hardware

- All hardware from the *Open Loop Step Response* experiment is required for this lab. (No additional hardware is required)



## Software

- All software from the *Open Loop Step Response* experiment is required for this lab.

## Previous Experiments

- *Open Loop Step Response*

### 6.2.2 Hardware Setup

No hardware setup is required. You should have completed the hardware setup in the *Open Loop Step Response* experiment.

### 6.2.3 Software Setup

No software setup is required. You should have completed the software setup in the *Open Loop Step Response* experiment.

## 6.3 Experimental Procedures

### 6.3.1 Exercise 1: Deriving Closed Loop Motor Transfer Function

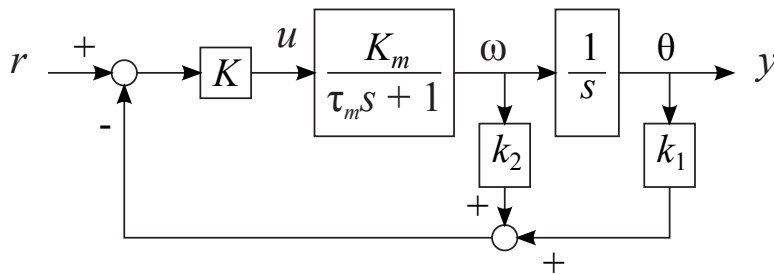


Figure 6.1: Empty Block Diagram for Closed Loop Motor

1. Using block diagram reduction on the block diagram in Figure 6.1, find the closed loop transfer function  $\frac{Y(s)}{R(s)}$ .

2. Assume that the reference input,  $r(t)$ , is a step function of amplitude  $\frac{\pi}{4}$ . You calculated  $K_m$  and  $\tau_m$  from the *Open Loop Step Response* experiment. Consider the following two sets of gains:

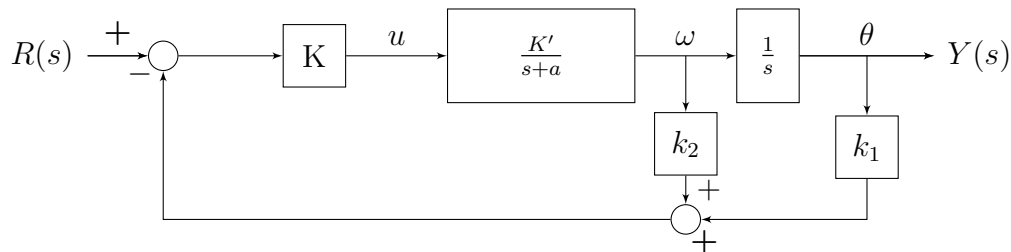
- $K = 56, K_2 = 0.06, K_1 = 1$
- $K = 15, K_2 = 0.5, K_1 = 1$

First, find the closed loop poles for both sets of gains. Based on the pole locations, estimate the settling time, percent overshoot, and frequency of oscillation of the closed loop step response for each set of gains. Also, find the steady state value from the closed loop transfer function.

3. Using the values you computed in the previous step, sketch the closed loop step responses for each set of gains. (You will make two different plots, one for each set of gains.) Show scales on all axes.

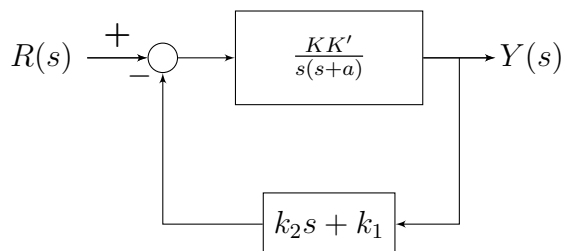
### Expected Hand-Written Results (For Instructors)

Students are asked to use block reduction and solve for the overall transfer function  $\frac{Y(s)}{R(s)}$  from Figure 6.1. To solve the block reduction, we must first take the open loop transfer function ( $\frac{K_m}{\tau_m s + 1}$ ) and put it into the closed loop block diagram form as below:



Where  $K' = \frac{K_m}{\tau_m}$  and  $a = \frac{1}{\tau_m}$ . This simplification makes it easier to compute the closed loop poles from the final closed loop transfer function. Next, pass the  $k_2$  gain

over the integrator, multiply  $K$  times the motor transfer function, and multiply the transfer function by  $\frac{1}{s}$  to get the following result:



Setting  $k_1 = 1$  we can do simple feedback block reduction to find the closed loop transfer function  $\frac{Y(s)}{E(s)}$ :

$$E(s) \rightarrow \left[ \frac{KK'}{s^2 + (a + KK'k_2)s + KK'} \right] \rightarrow Y(s)$$

Recall that for the *Open Loop Step Response* experiment, we found the values for  $K_m$  and  $\tau_m$  when the load was full of pennies and when it was empty. The values for those parameters are:

- Full Penny Load:  $K_m = 18.5$  and  $\tau_m = 16.2$
- Empty Penny Load:  $K_m = 17$  and  $\tau_m = 2.7$

Given the gains  $K$  and  $k_2$  and the values for  $K_m$ , and  $\tau_m$  we can solve for the closed loop poles (There will be 4 different sets of poles corresponding to each gain applied to each load configuration) with the following equation:

$$Poles = \frac{-(a + KK'k_2) \pm \sqrt{(a + KK'k_2)^2 - 4(KK')}}{2} \quad (6.1)$$

After calculating the closed loop poles, we can also calculate the settling time and frequency of oscillation. For this solution, the 2% settling time and frequency of

oscillation are:

$$Settling\ Time = \frac{4}{\|Re\{pole\}\|} \quad (seconds) \quad (6.2)$$

$$Frequency\ of\ Oscillation = Im\{pole\} \quad \left(\frac{radians}{second}\right) \quad (6.3)$$

Additionally, the natural frequency  $\omega_n$  and the damping ratio  $\zeta$  are calculated by setting the denominator of the closed loop transfer function equal to:

$$s^2 + 2\zeta\omega_n s + \omega_n^2 \quad (6.4)$$

Which gives us the relations:

$$a + KK'k_2 = 2\zeta\omega_n \quad (6.5)$$

$$KK' = \omega_n^2 \quad (6.6)$$

$\zeta$  may also be calculated from the following equation:

$$\zeta = \sin\left(\text{atan}\left(\frac{Re\{pole\}}{Im\{pole\}}\right)\right) \quad (6.7)$$

Students are given the gain values for  $K$  and  $k_2$  and have the motor model parameters  $K_m$  and  $\tau_m$ ; therefore, they can solve the relationships in Equations 6.5 - 6.7 to find the appropriate values for  $\zeta$  and  $\omega_n$ . The damping ratio is also important in finding the percent overshoot, which is calculated from  $\zeta$  in the following way:

$$PercentOvershoot = 100 * e^{-\frac{\zeta\pi}{\sqrt{1-\zeta^2}}} \quad (6.8)$$

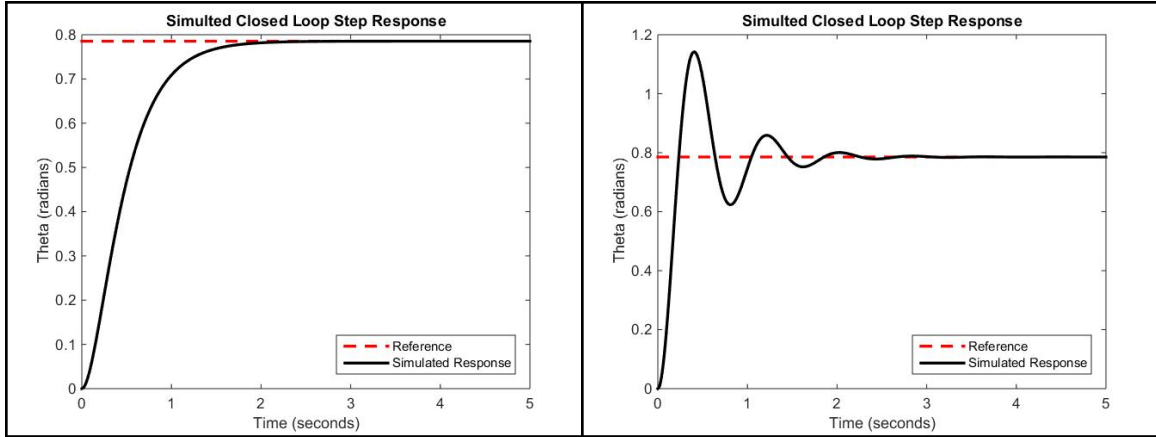
Table 6.1 below shows all of the closed loop poles, settling times, percentage overshoots, and frequencies of oscillation for the following configurations: full penny load

under-damped, full penny load over-damped, empty load under-damped, and empty load over-damped. The plots below the table show the closed loop step responses (from Simulink) for each load configuration. In the table the settling times found for each load configuration will serve as the upper and lower limits of the possible settling times students can produce from their experiments. Similarly, the calculated percentage overshoot for both the full and empty penny loads (underdamped) will be the bounds for their values. If their calculated settling times or percentage overshoot lie beyond these bounds, it is likely they have made a mistake with their calculations.

**NOTE:** The reference value used to calculate the closed loop step response is equal to  $\frac{\pi}{4}$ , which is the constant, dotted line shown in Figures 6.2 and 6.3. This value also serves as the steady-state value of the closed loop step response.

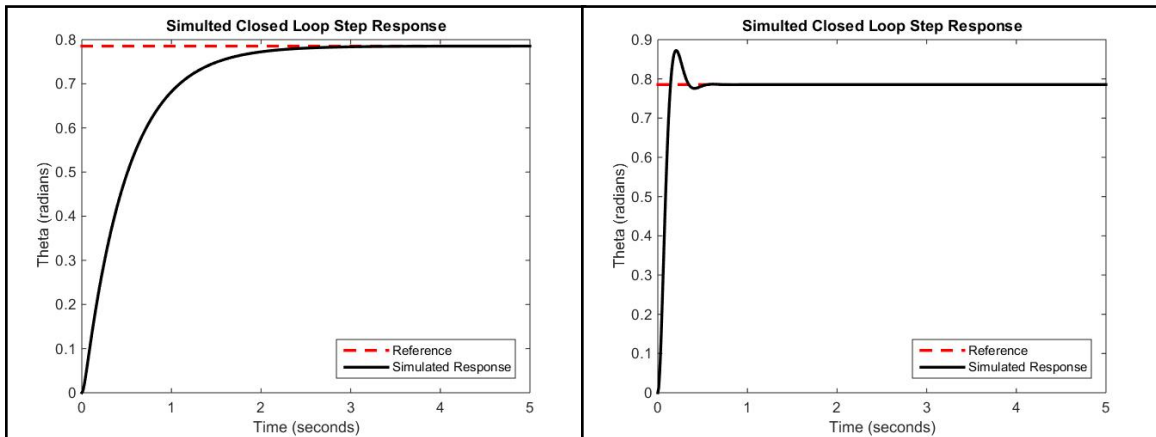
Table 6.1: Results Found From Simulating Different Gains and Load Configurations

Load Config	Damp	Closed Loop Poles	Set. Time (s)	% Overshoot (%)	Damp Ratio	Freq. of Osc. ( $\frac{rad}{s}$ )
Full Penny	Over	-5.53 -3.10	1.29	0	1.042	0
Full Penny	Under	-1.95±7.75i	2.05	45.36	0.244	7.76
Empty	Over	-45.52 -2.07	1.93	0	2.449	0
Empty	Under	-10.76±15.39i	0.37	11.12	0.573	15.39



(a) Full Pennies Over-Damped Simulation    (b) Full Pennies Under-Damped Simulation

Figure 6.2: Full Pennies Over-Damped and Under-Damped Simulation Results



(a) Empty Load Over-Damped Simulation    (b) Empty Load Under-Damped Simulation

Figure 6.3: Empty Load Over-Damped and Under-Damped Simulation Results

## Setting Up Matlab File

4. Open the **OL\_Constants.m** file created in the open loop step response experiment.
5. Save the file as **CL\_Constants.m**.
6. Type **ref=pi/4;** at the bottom of the Matlab file. This will be the reference position for the closed loop step response.
7. On the next line type **K =56;** This will be the gain that is multiplied by  $r - (k_2\omega + k_1\theta)$ . Since  $k_1$  is always equal to 1, K effectively multiplies  $(r - \theta) - k_2\dot{\theta} = e - k_2\dot{\theta}$ . The term  $Ke$  is called proportional feedback, since it produces an input that is proportional to the error. The term  $Kk_2\dot{\theta}$  is the derivative feedback, and has a damping effect, like viscous friction.

**NOTE:**  $k_1$  will not be added to this Matlab file, since it will always be equal to 1 for this experiment.

8. Type **K2 = 0.06;**

**NOTE:** The gains K and  $k_2$  (given in Step 2) will be reassigned with new values corresponding to the second set of gains in Exercise 74.

```
1 %Sampling Time
2 - Ts = 0.01;
3
4 %Encoder Resolution
5 - rp = 2*pi/1336;
6
7 %Supplied Voltage
8 - Vmax = 15;
9
10 %Gain used to convert voltages into duty cycles
11 - v2d = 255/Vmax;
12
13 %Observed Km and tau from experimental results
14 - Km = 0; %*** REPLACE 0 WITH Km VALUE
15 - tau = 0; %*** REPLACE 0 WITH tau VALUE
16
17 %Reference value
18 - ref=pi/4;
19
20 %Gains
21 - K = 42; %Proportional Gain
22 - K2 = 0.06; %Derivative Gain
```

Figure 6.4: Matlab File for Closed Loop Experiment

### 6.3.2 Exercise 2: Simulated Closed Loop Step Response (First Feedback Gain Set)

You will first simulate the closed loop step response before finding the step response experimentally. This exercise will provide steps for editing the **OL\_Simulation.slx** file you created in the *Open Loop Step Response* laboratory to produce a simulated closed loop step response.

#### Setting Up Simulink File (Simulation)

9. Open the **OL\_Simulation.slx** file created in the *Open Loop Step Response* experiment.
10. Save the file as **CL\_Simulation.slx**.
11. Change the simulation time to 10 seconds.  
  
**NOTE:** You will run the file for 10 seconds instead of 20 seconds because the closed loop step response will have a shorter settling time than the open loop step response.
12. Delete the **Scope** and **To Workspace** connections to the **LTI System** block by left-clicking on the connection and pressing **Delete** on the keyboard.
13. Additionally, delete the connection between the **Input Voltage** and the **LTI System** blocks.
14. Add 2 **Sum** blocks, 2 **Gain** blocks, and 1 **Integrator** block to the Simulink model.
15. Rename the **Input Voltage** block to **Reference**. Double-click on the same block and change the “Constant value:” to **ref**.



16. Rename one of the **Gain** blocks to **K**. Double-click and change the “Gain:” value to **K**.
17. Rename the other **Gain** block to **K2**. Change the **Gain:** value to **K2**.
18. Connect the output of the **Reference** block to the input of one of the **Sum** blocks. Connect the output of the **Sum** block to the input of the **K** block. Connect the output of the **K** block to the input of the **LTI System** block.
19. Double-click on the **Sum** block from the previous step and next to “List of signs:” change the second + sign (farthest to the right) to a - sign and click okay.
20. Double-click the **LTI System** block and change the “LTI system variable” text to **tf(Km,[tau 1])**. Connect the output of the **LTI System** block to the input of **Integrator** block. Connect the output of the **Integrator** block to the input of one of the **Scope** blocks. Rename the same **Scope** block to **Angular Position**.
21. Right-click on **K2** and hover the mouse over **Rotate & Flip**. Left-click on **Clockwise**.
22. Connect the input of **K2** to the connection between **LTI System** and **Integrator**.
23. Double-click on the unused **Sum** block. Next to “List of signs:” move the two + signs to the left of the | sign (See Figure 6.5)

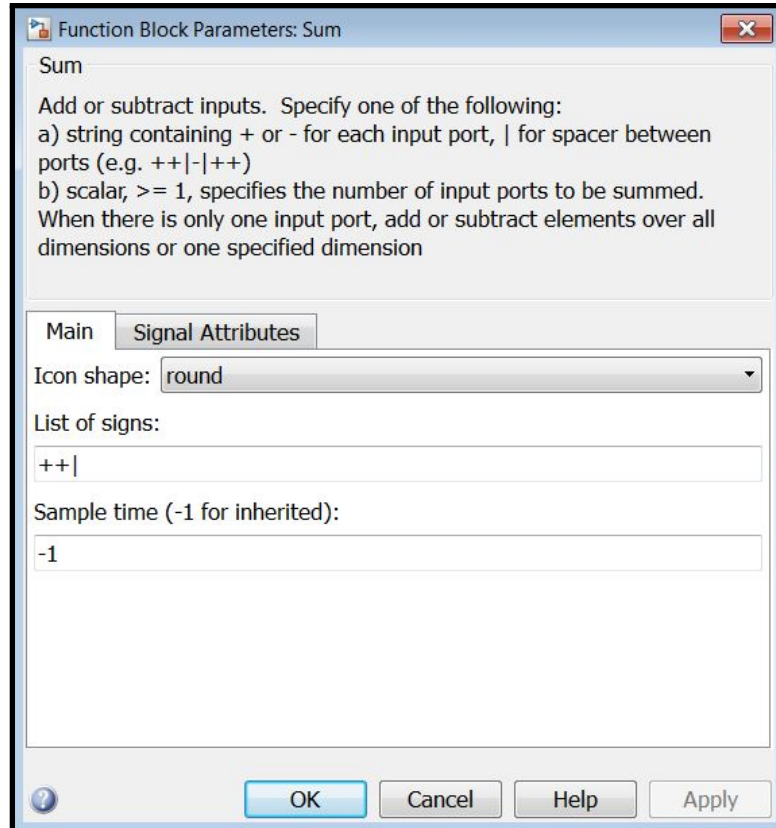


Figure 6.5: Correct Setup for Second Sum Block

24. While the **Sum** block is still highlighted (the block is outlined in blue) hold down the key combination **Ctrl-R** to rotate the block Clockwise and then let go. Once again, hold down **Ctrl-R** and let go of the keys. The **Sum** block should now have two inputs: one facing up and the other facing right. The output should be facing to the left.
25. Connect the output of **K2** to the input (facing in the upward direction) of the **Sum** block from the previous step. Connect the input (facing to the right) of the **Sum** block to the output of the **Integrator** block and the input of the **Angular Position** scope block.
26. Connect the output of the **Sum** block from the previous step to the - input of the other **Sum** block.
27. Connect the input of the **To Workspace** block (with value **simResp**) to the

connection between the output of the **Integrator** block and the input of the **Angular Position** scope block.

28. This concludes the setup of the simulation file for the closed loop step response of the motor. See Figure 6.6 for the final Simulink simulation file.

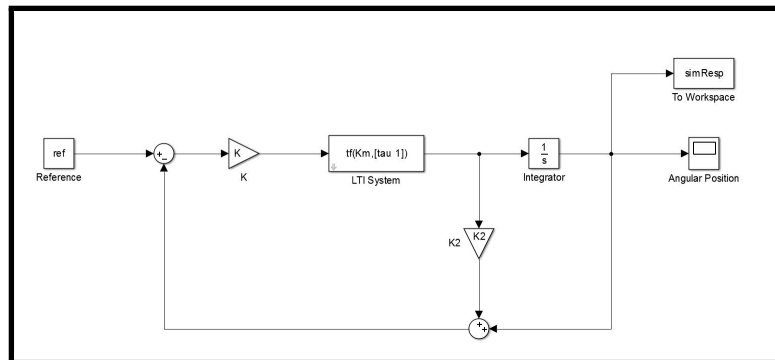





Figure 6.6: Final Simulink Simulation Model Used in the Closed Loop Step Response Experiment

### Collecting Simulation Data

29. Open **CL\_Constants.m** and then press the **Run** button  at the top of the page. Navigate to the **MATLAB 2014b** home page. Under “Workspace” on the right-hand side of the page, all of the variables from **CL\_Constants.m** should be listed.




**NOTE:** if any variables created in the **Setting Up Matlab File** section are not listed under “Workspace,” simply add the missing variables to the bottom of the **CL\_Constants.m** and click Save once more. If any variables are missing from the file, the **CL\_Simulation.slx** file will not run correctly.

30. Open **CL\_Simulation.slx**. Click the Run button  at the top of the page.
31. Once the model has finished running, double-click on the **Angular Position** scope block. Click the **Autoscale** button . Observe the plot. Does the

closed loop step response appear to rise up from zero and settle to the reference value (as in your theoretical sketch in Exercise 1)? If the plot looks to be correct (with a run time of 10 seconds) continue to the next step. Otherwise, go back to the **Setting Up Simulink File (Simulation)** section to ensure your Simulink file is correct.

32. Navigate back to the **MATLAB 2014b** home page. Under “Workspace” a variable (**simResp**) should now be available. Right click on **simResp** and click “Save As...” Navigate to the folder in which you have saved this project, type next to “File name:” **CL\_simResp\_1.mat**, and click “Save” at the bottom of the page.
33. You now have the simulation data found from Simulink for the first set of gains.

### Simulation Plot File

34. Open the main Matlab 2014b window and click **New** at the top and then click **Script**.
35. Once the new Untitled m-file appears, Click **Save**  at the top of the page. Save the file as **CL\_Plot.m**.
36. Copy and paste the text in Table 6.2 into the Matlab file. After adding the code click **Save**  and then click **Run** .
37. Save the figure as **CL\_S\_1.fig** into your folder for this project. Refer to this figure for the remaining steps in this section.
38. Compare the simulation results with the hand-written results you found in Exercise 1. Does the simulated plot resemble the hand-written plot found from the first set of gains? Discuss similarities and explain any differences.

39. Estimate the settling time, percent overshoot, and frequency of oscillation of the closed loop step response from the simulated plot and compare with the handwritten plot.
40. Also compare the steady state values from each plot. Discuss the similarities and explain any differences.

Table 6.2: Code for Plotting the Closed Loop Step Response Simulated Results

---

```

%Load the simulation data and store into a variable
CL_simResp_1 = load('CL_simResp_1.mat');
CL_simResp_1 = CL_simResp_1.simResp.Data;

%Set the time vector to be 10 Seconds with Ts time steps
T = 0:Ts:10;

%Plot the simulation data with respect to time
figure(1);
plot(T,ones(size(T))*ref,'-', 'Color', 'r');
hold on;
plot(T,CL_simResp_1, ...
'Color', 'k', ...
'LineWidth', 2);
title('Simulated Closed Loop Step Response');           %Title
legend('Reference','Simulated', 'Location', 'southeast'); %Legend
xlabel('Time (seconds)')                                % x-axis label
ylabel('Theta (radians)')                              % y-axis label

```

---

### 6.3.3 Exercise 3: Experimental Closed Loop Step Response (First Feedback Gain Set)

This section will provide the setup of the Simulink file for the Arduino.

#### Setting Up Simulink File (Arduino)

41. Open the Simulink file created in the *Open Loop Step Response* experiment named **OL\_Step\_Resp\_Arduino.slx**.
42. Add 2 **Gain** blocks, 1 **Abs** block, 2 **Sum** blocks, and 1 **User Defined Fcn** block to the file. In the upper left hand corner of the Simulink model, click “File” and then “Save As”. Save this file as **CL\_Step\_Resp\_Arduino.slx** in the same folder where you saved the *Open Loop Step Response* Simulink file **OL\_Step\_Resp\_Arduino.slx**.
43. Delete the connection from the **Voltage** block going into the **Voltage to PWM** block. Connect the output of the **Voltage** block to the input of one of the **Sum** blocks and the output of the **Sum** block to the input of one of the **Gain** blocks. Double-click the **Sum** block and under “List of signs:” change the second + sign (farthest to the right) to -. Click Okay. The **Sum** block should have a minus sign for the bottom input.
44. Rename the **Gain** block from the previous step to **K**. Double-click and input a value of **K** for the “Gain:” value. Connect the output of the **K** block to the input of the **Abs** block.
45. Connect the output of the **Abs** block to the input of the **Voltage to PWM** block.
46. Rename the **Voltage** block to **Ref**. Double-click on the block, change the “Constant value:” to **ref**, and then click Okay. Connect the output of **Ref** to

the (left) input of the **Sum** block from the previous step.


47. Change name of the **User Defined Function** block to **Sign to Direction**. Double-click the block to reveal the Matlab editor window with the tab “Sign to Direction” visible. Delete all of the code in the function and copy-paste the text from the Matlab file shown in Table 6.3 into the block and click **Save**  at the top.

Table 6.3: Code for Sign to Direction Function)

---

```
function y = fcn(u)
%#Function that sets the output to 1 if the input is positive and 0 if the
% input is negative

if u >= 0
y = 1;
else
y = 0;
end
```

---

48. Connect the input (u) of the **Sign to Direction** block to the output of the **K** block. To do this, left-click on the input of the **Sign to Direction** block, hold, and drag the (red-dashed) line down to the connection where the **K** block goes into the **Abs** block. Once on top of the connection (the red-dashed line becomes a solid black line) let go of the mouse. (The output of the **K** block should now be connected to both the **Sign to Direction** block and the **Abs** block.)

49. Delete the **Direction** block and its output connection to the **Digital Output** block. Connect the output ( $y$ ) of the **Sign to Direction** block to the input of the **Digital Output** block (where the output of the **Direction** block was connected).
50. Delete the **Discrete Filter** block and the connection the block has with the **Data Type Conversion** block. Connect the output of the **Velocity Scaling** block to the input of the remaining **Gain** block. Rename the **Gain** block to **K2** and double-click the block to change the “Constant value:” to **K2**.
51. Left-click on the **K2** and hold down the keyboard combination **Ctrl-I**. This will flip the direction of the **K2** block to the output facing left. (This will help the Simulink file look cleaner later in this setup.)
52. Left-click on the remaining **Sum** block and hold down the keyboard combination **Ctrl-I**. The sum block output should now be on the right and the side input should be facing to the right. Connect the output of the **K2** block into the right input of the **Sum** block. Connect the output of the **Sum** block to the - input of the other sum block.
53. Connect the bottom + input of the **Sum** block to the connection between the **Steps to Radians** block. Also connect the input of the **Data Type Conversion** block to the same connection. The output of **Steps to Radians** should be going to the inputs of the **Difference**, **Sum**, and **Data Type Conversion** blocks.
54. Click **Save**. The Arduino Simulink file for the experimental closed loop step response is now complete. See Figure 6.7 for the completed model.



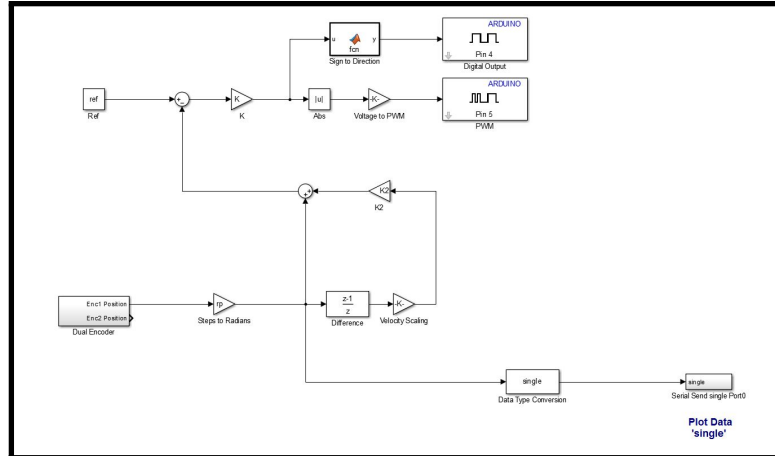




Figure 6.7: Final Closed Loop Step Response Simulink Model for Arduino

## Collecting Experimental Data

55. Open `CL_Constants.m` and click the Run button  at the top of the page.
56. Open `CL_Step_Resp_Arduino.slx` and click the “Deploy to Hardware” button  at the top-right of the page.
57. Once the model has successfully deployed to the Arduino, double click on the text **Plot Data ‘single’** inside the model window.
58. When the small window labeled “Plot Ser...” appears, enter the Arduino COM port number under “Enter COM port to collect data:.” The default values for “Enter Number of Samples to plot:” is “single” and for “Enter Number of samples to plot:” is **1000**. If those values are anything different, change them back to their default values.

**Note:** To find the COM port number for your Arduino, refer to the *Simple DC Motor* experiment under the section “Software Setup → Installing Arduino Mega 2560 Drivers,” specifically steps 20-24.

59. Click Okay. Once the plot appears, plug the power cord from the power supply into the motor shield.

**CAUTION:** Do not put your hands or any other parts of your body in front of the motor load trajectory. If the load does not begin spinning once the motor shield is plugged in, immediately unplug the power and check to see if everything is connected properly (review the **Hardware Setup** section in the *Open Loop Step Response* experiment for the proper hardware connections.)

60. Observe the plot. If the plot appears to be very jumpy (meaning that the values do not look to be “smooth” and vary from extremely positive to negative values) or if the values are not changing from zero, proceed to the next step. If the plot appears to be smooth (rising to a value and staying around that value, as you found in Exercise 1,) then skip to step 64 of this section.
61. Unplug the power from the motor shield. On the Arduino, click the reset button, which should be located under the motor load.
62. Once the Arduino has fully reset, the plotting window should appear to output a value of zero (flat line). If this is not the case, press reset once more until the plot displays zero.
63. Plug the power cable back into the motor shield, being careful not to bump the motor load (This will throw off the initial angular position of the load.) The load on the motor should move. If the data appears to be increasing smoothly, continue to the next step. If the data is still not smooth (or the initial angle is off from zero), repeat steps 61-63.
64. Let the data fill the plot window as it moves to the left. Once the data (starting with a value of zero) has filled the screen completely, click the “Stop” button at the bottom of the screen. You should now have 1000 data points on the screen (which is 10 seconds)

**Note:** you should expect to see a curve beginning at zero, rising up, and settling

to the reference value, as you found in Exercise 1. Try to make the first value of the plot be equal to zero by clicking stop once the zero velocity point crosses the origin. For more info on doing this, refer to the *Sampling and Normal vs External Mode* experiment.

65. Navigate back to the **MATLAB 2014b** main page. Under “Workspace” the variable **WindowDat** should now be present. Right-click on it and click “Save As.” Name the file **CL\_expResp\_1.mat** and save it into the folder where the **CL\_Step\_Resp\_Arduino.slx** file is saved.
66. You now have the experimental data for the closed loop step response and the first gain set.

### Experimental Plotting File



67. Open the **CL\_Plot.m** file you created in the **Simulation Plot File** section.
68. Add the text in Table 6.4 to the bottom of the **CL\_Plot.m** file. After adding the code, click **Save**  and then click **Run** .

Table 6.4: Code for Plotting the Closed Loop Step Response Experimental Results

---

```
%Load the experimental data and store into a variable
CL_expResp_1 = load('CL_expResp_1.mat');
CL_expResp_1 = CL_expResp_1.WindowDat;


%Plot the simulation data with respect to time
hold on;
plot(T,CL_expResp_1, ...
'Color', 'g', ...
'LineWidth', 1);
title('Simulated vs. Experimental Closed Loop Step Response');           %Title
legend('Reference', 'Simulated', 'Experiemental', 'Location', 'southeast'); %Legend
xlabel('Time (seconds)')                                                % x-axis label
ylabel('Theta (radians)')                                              % y-axis label
```

---

69. Save the figure as **CL\_SE\_1.fig** into your folder for this project. Refer to this figure for the remaining steps in this section.
  
70. Compare the simulation results with the experimental results you found. Do the simulated plot and the theoretical plot from Exercise 1 resemble the experimental plot found from the first set of gains? Discuss similarities and explain any differences.
  
71. Estimate the settling time, percent overshoot, and frequency of oscillation of the closed loop step response from the experimental plot. Compare with the handwritten and simulated plot.

72. Also compare the steady state values from each plot discussing similarities and explaining differences. What could cause the simulated response to differ from the experimental response?

#### 6.3.4 Exercise 4: Simulated Closed Loop Step Response (Second Feedback Gain Set)

73. Open the file **CL\_Constants.m**.
74. Change the gain values to **K = 15**; and **K2 = 0.5**; and click **Save** .
75. Repeat the experiment beginning with Section **Collecting Simulation Data**, steps 29-40, with the following exceptions:
- In step 32 name the data you find from simulation as **CL\_simResp\_2.mat** for the second set of gains.
  - Change each **CL\_simResp\_1.mat** in the **CL\_Plot.m** file you created in Table 6.2 to **CL\_simResp\_2.mat**.
  - Comment out the second half of the **CL\_Plot.m** file pertaining to the experimental values you in Table 6.4 (for the time being, in order to observe only the Simulated Plot)
  - In step 38, save the figure as **CL\_S\_2.fig**.
  - In steps 39 and 40, compare **CL\_S\_2.fig** with the hand-written results you found for the second set of gains.

#### 6.3.5 Exercise 5: Experimental Closed Loop Step Response (Second Feedback Gain Set)

76. Repeat Section **Collecting Experimental Data** steps 55-72 with the following exceptions:

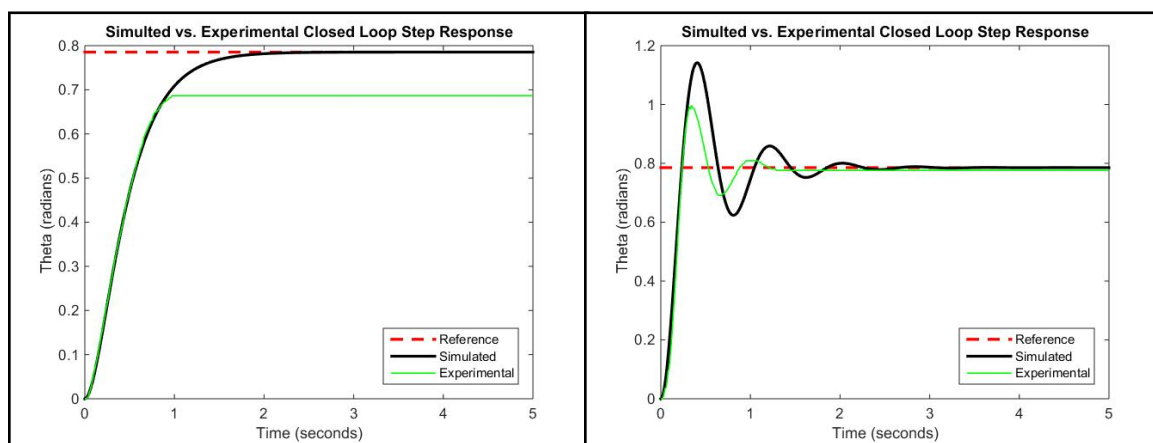
- In step 65 name the experimental data as **CL\_expResp\_2.mat**.
- Before running the **CL\_Plot.m** file in step 68, uncomment the second half of the code in the **CL\_Plot.m** file (Created in Table 6.4.) Change each **CL\_expResp\_1.mat** to **CL\_expResp\_2.mat**. After making these changes, then run the file.
- In step 69 save the figure as **CL\_SE\_2.fig**.
- In steps 70 - 72, compare the simulated and experimental responses in **CL\_SE\_2.fig** with the handwritten results you found for the second set of gains.

## **Expected Simulated vs. Experimental Results Using Second Gain Set (For Instructors)**

The results for the simulated vs. experimental responses (See Figures 6.8 and 6.9) show that the experimental results are not exactly like the simulation results. For instance, the overdamped response full penny load configuration appears to have a larger steady-state error than the empty penny load overdamped response. This is due to the nonlinear friction present at the motor shaft. For the overdamped empty penny configuration, it appears that the load is able to follow the simulation up to a higher steady-state value before staying constant. Since there is less mass in the empty penny configuration, it takes less voltage (thus less gain) to overcome the nonlinear friction, which in turn produces a lower steady-state error. Something also to note is that for the full penny configuration with overdamping the damping ratio is equal to 1.042, which is very close to an almost critically damped case. This is also more obvious with the settling time for the overdamped full penny configuration (1.29 seconds) which is smaller than both the underdamped and overdamped cases of the empty penny configuration (2.05 seconds and 1.93 seconds, respectively.) Since the damping ratio is almost optimal for the full penny configuration, the settling time is much faster.

For underdamped responses of each configuration, it appears that the full penny configuration is able to settle closer to the reference value than the empty penny configuration. This is likely by coincidence. Since the steady-state value for each response is very close to the actual reference value, we have a satisfactory result. Both responses are subject to nonlinear frictions in that neither completely reach the peak values of the simulation, and they don't exactly settle to the reference value. The underdamped full penny configuration even appears to become out of phase with the underdamped simulation response. While there appears to be a steady-state error for every response (both overdamped and underdamped for both configurations) they

all appear to follow very closely in the beginning to the predicted rising responses, showing that in a range where the voltage is larger, the masses are able to overcome the nonlinearities in the friction and behave with satisfactory results. It should be expected that students will run into issues with the nonlinear frictions with their various load configurations. Testing out each extreme (full penny and empty penny) we were able to find gains that produce acceptable overdamped and underdamped responses, for all possible combinations of load inertias.

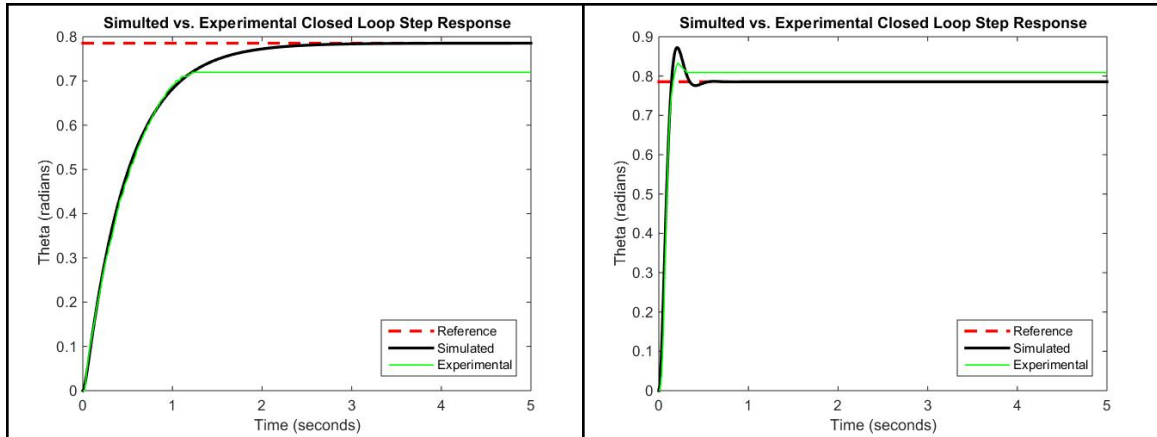


(a) Full Pennies Over-Damped

(b) Full Pennies Under-Damped

Figure 6.8: Full Pennies Over-Damped and Under-Damped Simulation Vs. Experimental Results





(a) Empty Load Over-Damped

(b) Empty Load Under-Damped

Figure 6.9: Empty Load Over-Damped and Under-Damped Simulation Vs. Experimental Results

#### 6.4 Conclusion/Student Feedback

This chapter provided an experiment that adds feedback to the *Open Loop Step Response* experiment. Simulink models for simulation were created and compared with the experimental closed loop step response results.

## CHAPTER 7

### Pole Positioning State Feedback

#### 7.1 Objective

This chapter describes an experiment that uses pole positioning on a Furuta pendulum (also called a Rotary Inverted Pendulum.) Students will create a Simulink model based on the equations of motion for the Furuta Pendulum provided to them. The objective is for students to use pole positioning with full state feedback to control the Furuta Pendulum. The open loop simulations of both the linear and nonlinear models will be verified. After the open loop verification, the closed loop model will be simulated. After completing the simulations, the controller will be loaded to the Arduino, and the experimental data will be captured from the serial port and compared with the simulations.

#### 7.2 Setup

##### 7.2.1 Required Materials

###### Hardware

All hardware from the *Open Loop Step Response* experiment will be required (with the exception of the 3D printed motor load and insert created in the *Introduction to 3D Printing Experiment*). Additionally, the following hardware will be used:

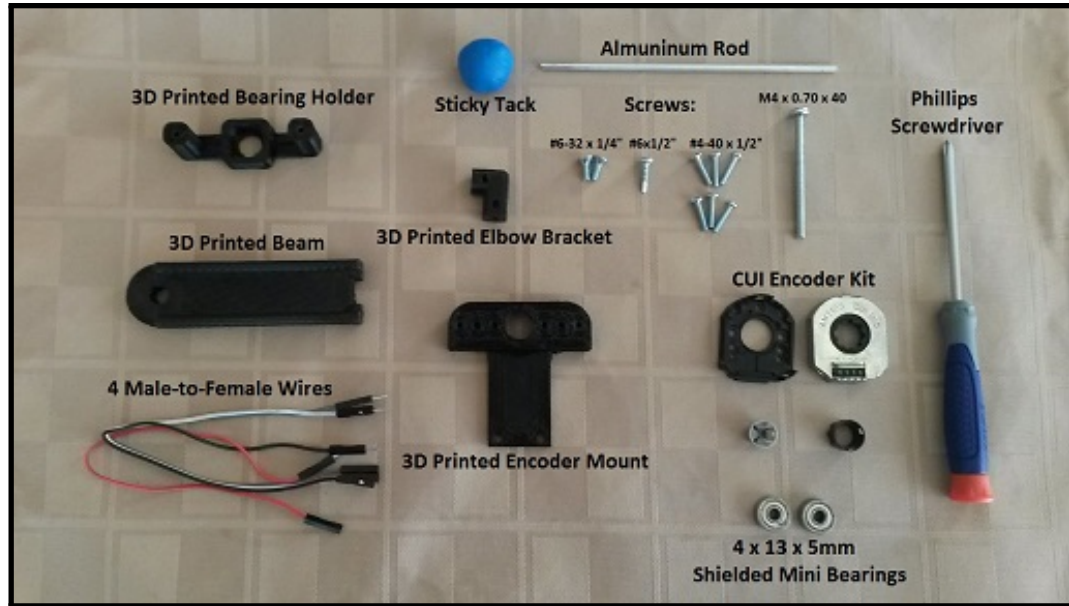


Figure 7.1: Hardware Required for Pole Positioning State Feedback Experiment

**NOTE:** All 3D printed materials were printed using .3mm resolution.

- 3D Printed Beam
- 3D Printed Encoder Mount
- 3D Printed Bearing Holder
- 3D Printed Elbow Bracket
- CUI Encoder Kit (AMT 103-V)
- Small Screw Driver (Phillips)
- Aluminum Rod (1/2-ft x 1/8 in. Aluminum Metal Rounds)
- 4 x 13 x 5mm Shielded Mini Bearings (2)
- M4 x 0.70 x 40 Coarse Thread Phillips Pan Head Screw
- #4-40 x 1/2 in. Screws (6)

- #6-32 x 1/4 in. Screws (2)
- #6 x 1/2 in. Self-Drilling Screw
- 4 Male to Female Wires (In addition to the *Open Loop Step Response* experiment wires)

## Software

- Matlab/Simulink 2014b
- Windows 7

## Prerequisite Experiments

- Simple DC Motor
- Sampling and Data Acquisition
- Introduction to 3D Printing
- Open Loop Step Response (Only the Hardware Setup Section, Steps 1-23.)

### 7.2.2 Software Setup

- No Software Setup is required. Use the same software as the *Simple DC Motor* experiment.

## 7.2.3 Hardware Setup

### Encoder Setup



Figure 7.2: Hardware for Encoder Setup

1. Obtain the CUI encoder kit (with gray and black inserts), 2 bearings, 3D printed bearing holder, 3D printed encoder mount, M4 x 0.70 x 40 coarse thread phillips pan head screw, #4-40 x 1/2 Inch Screws (4 of them), and a small screw driver.

**NOTE:** You can find the .stl files for the 3D printed files on the website. Simply 3D print the objects with a 3D printer. (Refer to the *Introduction to 3D Printing Experiment* for any help in doing this.)

2. Press the bearings into the 3D printed bearing holder and encoder mount.



(a) Press Bearing Into 3D Printed Encoder Mount (b) Press Bearing Into 3D Printed Bearing Holder

Figure 7.3: Pressing Bearings Into 3D Printed Objects

- Using 2 of the #4-40 screws, mount the black backing of the CUI encoder kit to the encoder mount.

**NOTE:** It is okay if the ends of the screws mounting the black backing to 3D printed encoder mount protrude through the other side of the 3D printed encoder mount.



Figure 7.4: Mount Black Backing of the CUI Encoder Kit to the Encoder Mount

- Stick the M4 x 0.70 x 40 screw through the back of the bearing holder (through the bearing. See Figure 7.5)

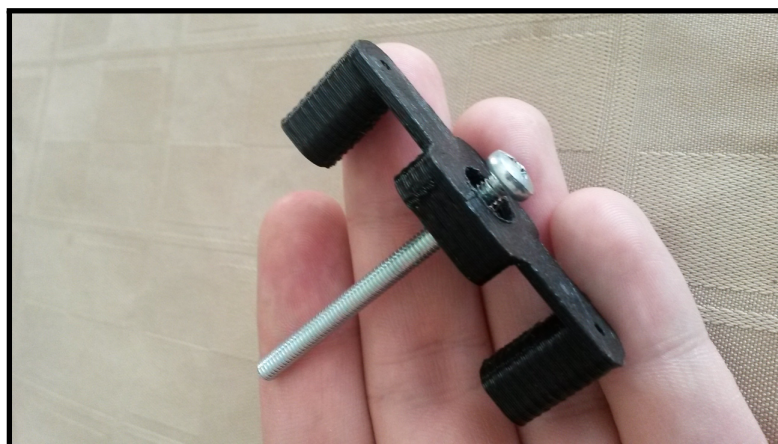


Figure 7.5: Stick the M4 x 0.70 x 40 screw through the back of the bearing holder

5. The encoder should be configured with all of the white switches in the same position as shown in Figure 7.6. This configuration sets the encoder resolution to be 8192 counts per revolution.

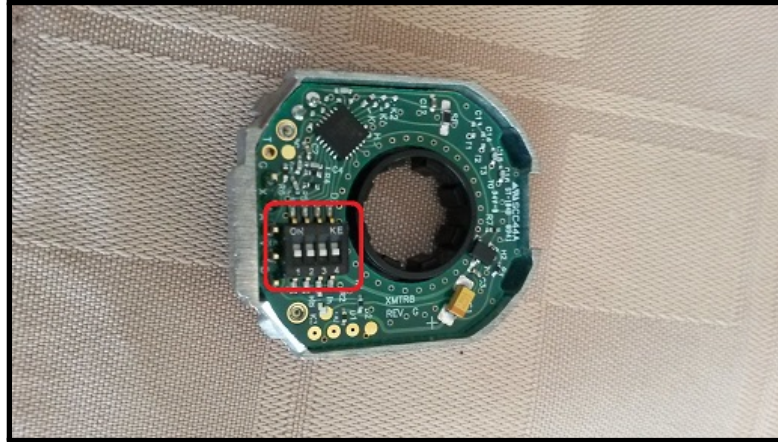


Figure 7.6: Correct Encoder Configuration

6. Put the M4 x 0.70 x 40 screw through the encoder as seen in Figure 7.7.

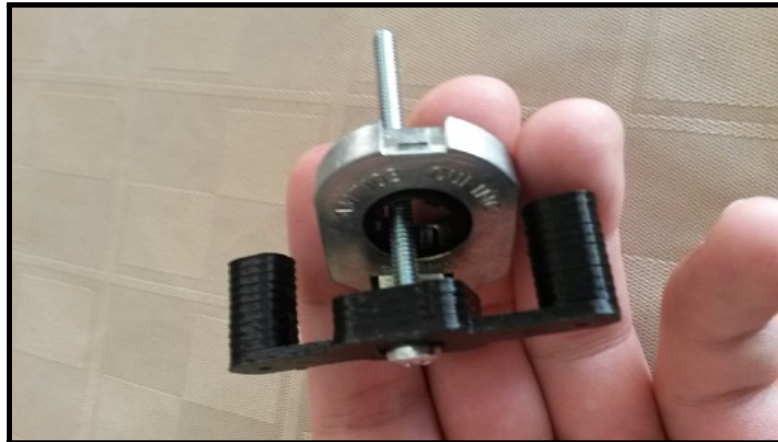


Figure 7.7: Stick the M4 x 0.70 x 40 screw through the Encoder

7. Place the black encoder insert into the encoder (the narrower end will go into the encoder.)



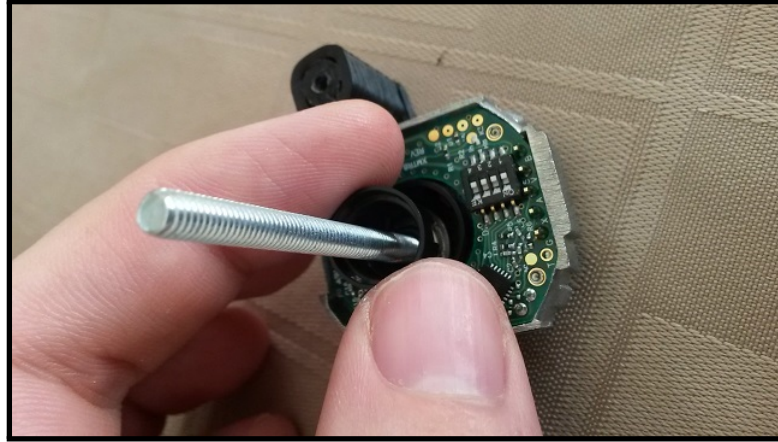
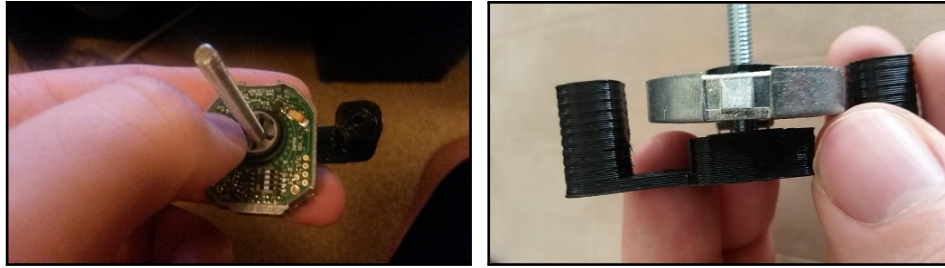


Figure 7.8: Place the Black Encoder Insert Into the Encoder

8. Place the gray insert (narrow end down) into the black insert.

**NOTE:** Placing the gray insert inside the black insert will cause the black and gray insert to stick to the position along the M4 x 0.70 x 40 screw in which they are joined (also holding the metal encoder face locked in the same position along the screw). When pressing the gray insert into the black insert, make sure there is a small gap between the bearing holder and encoder. If the bearing holder is too close to the encoder from the gray/black insert placement, you can hold the encoder with one hand while pressing the longer end of the M4 x 0.70 x 40 screw down toward the encoder and twisting left. This will cause the encoder assembly to function as a nut and move along the shaft, creating a gap between the bearing holder and encoder assembly. The opposite is true if you hold the gray/black insert and pull and twist right with the longer end of the M4 x 0.70 x 40 screw. This will cause the encoder assembly to move closer to the bearing holder.





(a) Press Gray Insert Into Black Insert  
 (b) Press Black/Gray Insert Combo into Encoder

Figure 7.9: Press the Gray Insert Into the Black Insert and Ensure Gap is Reasonable

- Slide the encoder mount assembly down onto the encoder along the M4 x 0.70 x 40 screw. Do not clip the backing and metal encoder face together yet.



Figure 7.10: Slide the Encoder Mount Assembly Down (Do Not Clip)

- Ensure that the legs of the bearing holder will reach the encoder mount. If the legs look like they will be too far away to reach the encoder mount, once the metal encoder face is clipped with the black backing, move the encoder assembly along the bolt closer to the bearing holder by twisting as mentioned in step 8 (to close the gap between them.) If the legs are so close that they will bend once the encoder assembly is clipped together, then move the assembly along

the longer part of the M4 x 0.70 x 40 screw. Once you are certain that the legs will reach without issue, clip the metal encoder face to the black backing.



(a) Clip Top of Encoder

(b) Clip Bottom of Encoder

Figure 7.11: Clip the Encoder Black Backing to the Metal Encoder Face

11. Once the encoder assembly is clipped together, take the remaining two #4-40 screws and (using the small phillips screw driver) mount the encoder mount to the bearing holder (see Figure 7.12.)



Figure 7.12: Mount the Encoder Mount to the Bearing Holder

12. This completes the encoder setup.

**NOTE:** Spin the M4 x 0.70 x 40 screw to ensure there isn't too much friction in the pivot. If you find it difficult to spin the M4 x 0.70 x 40 screw, then back

out the two screws mounting the bearing holder to the encoder mount, unclip the metal encoder face from the black backing (a small flat-head screw driver works well for this), and move the encoder inserts up and down the M4 x 0.70 x 40 screw until it is in a reasonable position. Then follow steps 9 - 12 once more.

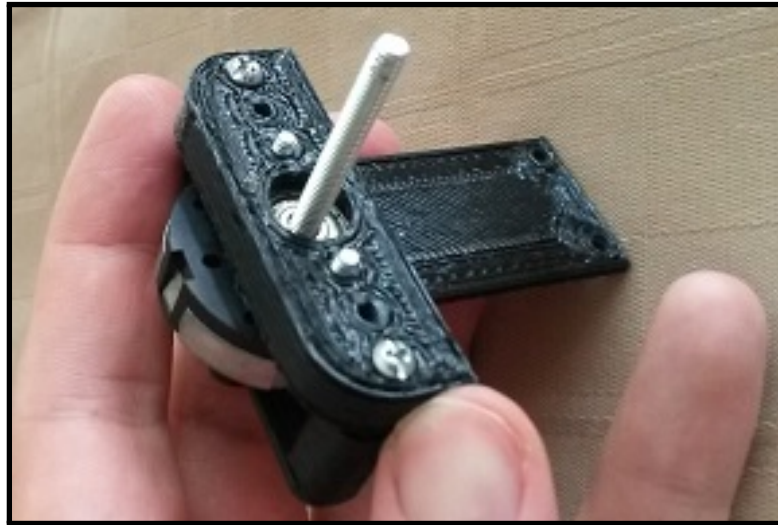


Figure 7.13: Final Encoder Assembly

### Attaching Encoder Assembly to Beam



Figure 7.14: Beam Hardware

13. Obtain the 3D printed beam, the final encoder assembly from the previous section, and the two remaining #4-40 screws.

14. Align the bottom holes on the encoder setup with the holes on the side of the beam.
15. Screw the two #4-40 screws through the holes of the encoder assembly into beam. This will attach the encoder assembly to the beam.



Figure 7.15: Mount Encoder Assembly to Beam

16. This completes the steps for attaching the encoder assembly to the beam.

#### Attaching Pendulum to Encoder Screw



Figure 7.16: Hardware Used to Attach the Pendulum to the Encoder Screw



17. Obtain the sticky tack, aluminum rod, 3D printed elbow bracket, and two 6-32 x 1/4 inch screws.
18. Cut the aluminum rod to about 4.5 inches in length using wire cutters.
19. Place the aluminum rod into the shorter end of the elbow bracket.

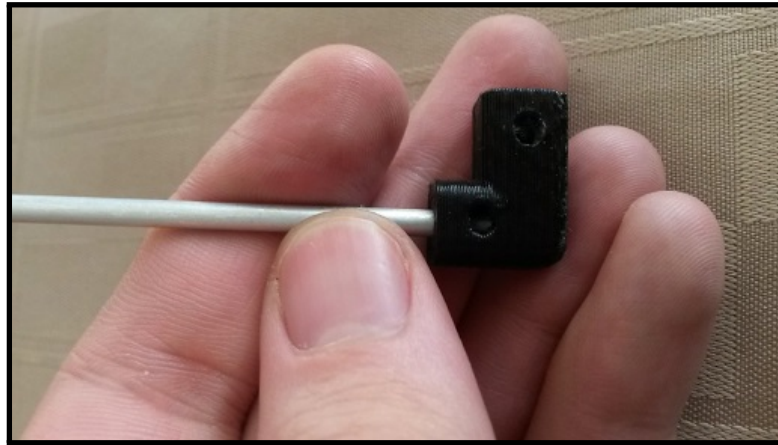


Figure 7.17: Place the Aluminum Rod Into the Elbow

20. Screw the 6-32 x 1/4 inch screw into one side of the smaller end of the elbow bracket.



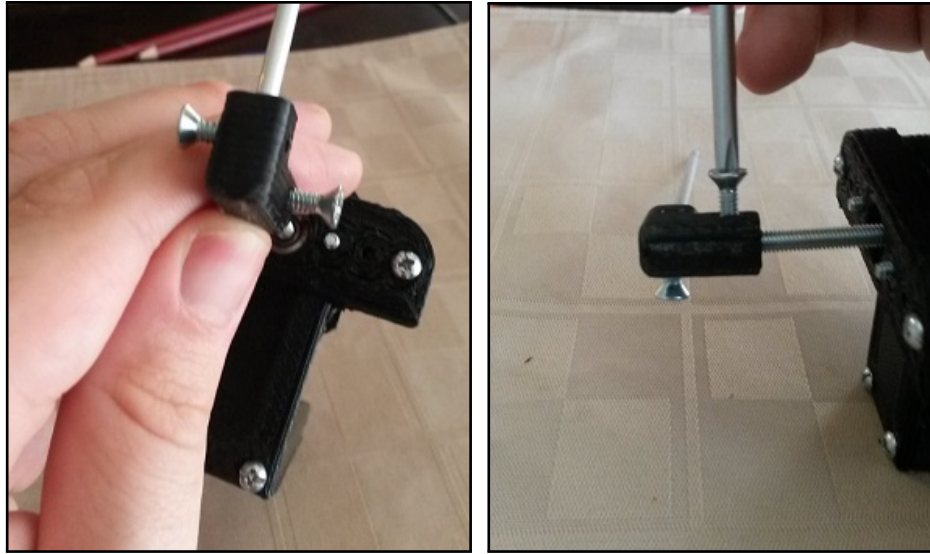
Figure 7.18: Hardware Used to Attach the Pendulum to the Encoder Screw

21. Slide the longer end of the bracket onto the encoder screw.



Figure 7.19: Slide the Elbow Onto the Encoder Screw

22. On the opposite side where you clamped the pendulum with the other screw, screw the remaining 6-32 x 1/4 inch screw (until snug) through the side of the longer end of the elbow bracket. This will clamp the pendulum assembly (elbow bracket and aluminum rod) to the encoder screw.



(a) Front View of Elbow Clamp

(b) Side View of Elbow Clamp

Figure 7.20: Screw to Clamp the Elbow Bracket to the Encoder Screw

23. Roll the sticky tack into a 1.5 inch diameter ball.



Figure 7.21: Roll the Sticky Tack into a Ball

24. Slide the ball of sticky tack onto the exposed tip of the aluminum rod on the pendulum assembly.

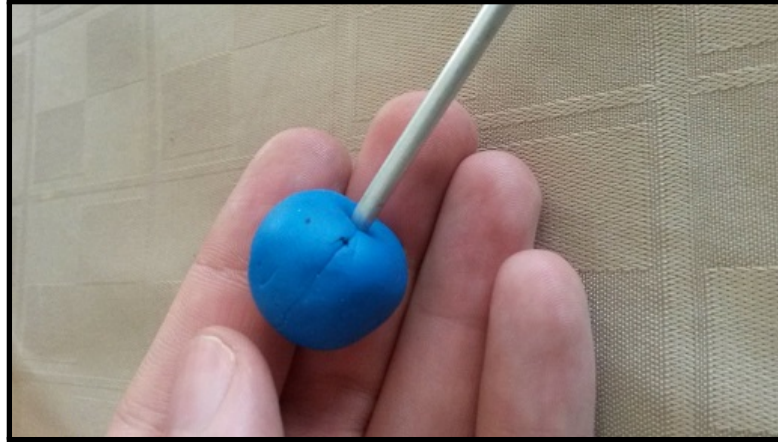


Figure 7.22: Slide Sticky Tack Ball Onto End of Aluminum Rod

25. The setup for attaching the pendulum assembly to the encoder/beam assembly is now complete.

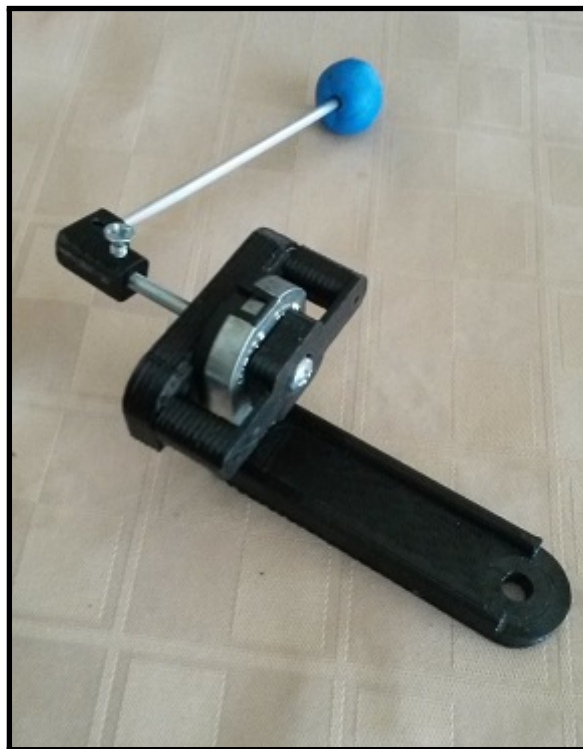
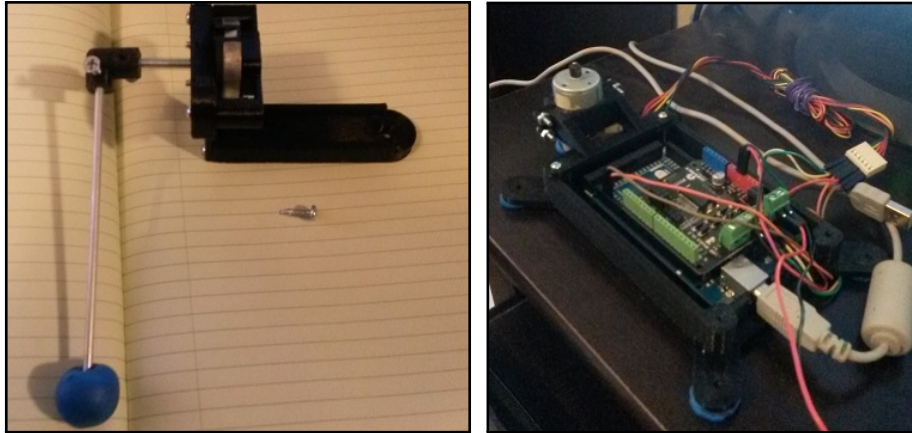


Figure 7.23: Final Pendulum/Beam/Encoder Assembly



## Attaching Pendulum/Beam/Encoder Assembly to Motor Shaft



(a) #6 x 1/2 inch Self-Drilling Screw (b) Hardware From *Open Loop* and Pendulum/Beam/Encoder As- *Step Response* Experiment  
sembly

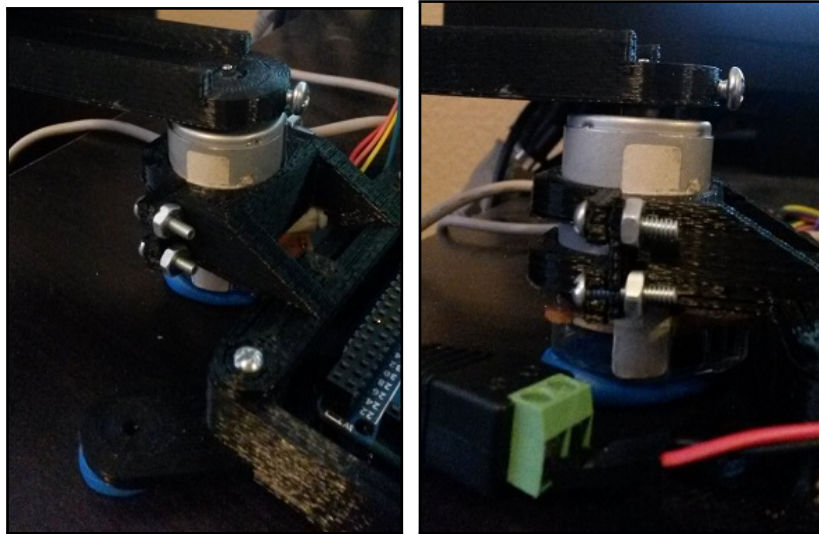
Figure 7.24: Hardware for Attaching the Pendulum/Beam/Encoder Assembly to the Motor Shaft

26. Obtain the #6 x 1/2 inch self-drilling screw, Pendulum/Beam/Encoder Assembly (created in the previous section), and the hardware assembly created in the *Open Loop Step Response* experiment
27. Screw the self-drilling screw into the back of the beam. (Do not go all the way through. Only turn the screw 2 or 3 times, so that it is hanging out.)



Figure 7.25: Screw Self-Drilling Screw Into the Back of the Beam

28. Place the motor shaft through the hole on the back of the beam. Screw the self drilling screw snugly to the motor shaft. Make sure there is a small gap between the beam and the motor. If both surfaces touch, it will add unwanted friction to the Furuta Pendulum system.



(a) Attach Beam to Motor Shaft (b) Gap Between Motor and Beam

Figure 7.26: Beam Attached to Motor Shaft With a Small Gap Between the Top of the Motor and the Beam

29. Once the Pendulum/Beam/Encoder Assembly is attached snugly to the motor shaft, the setup is complete.

### Connecting Encoder to Arduino

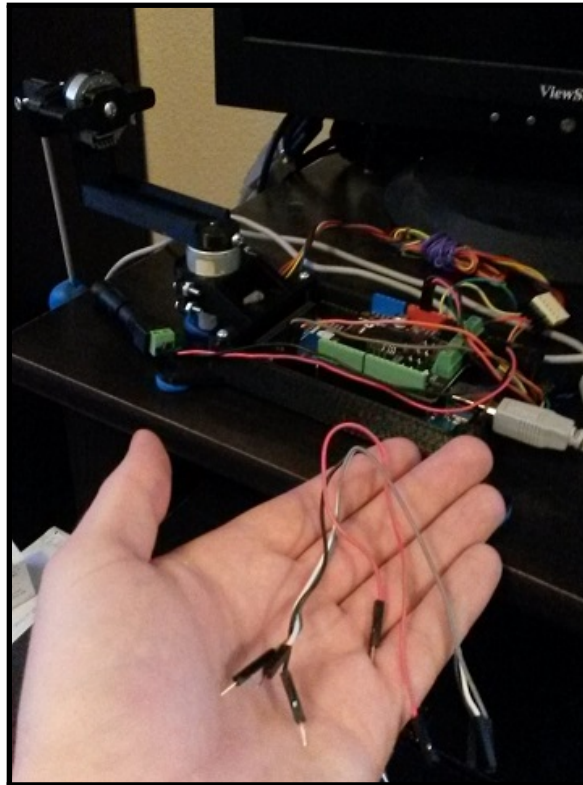
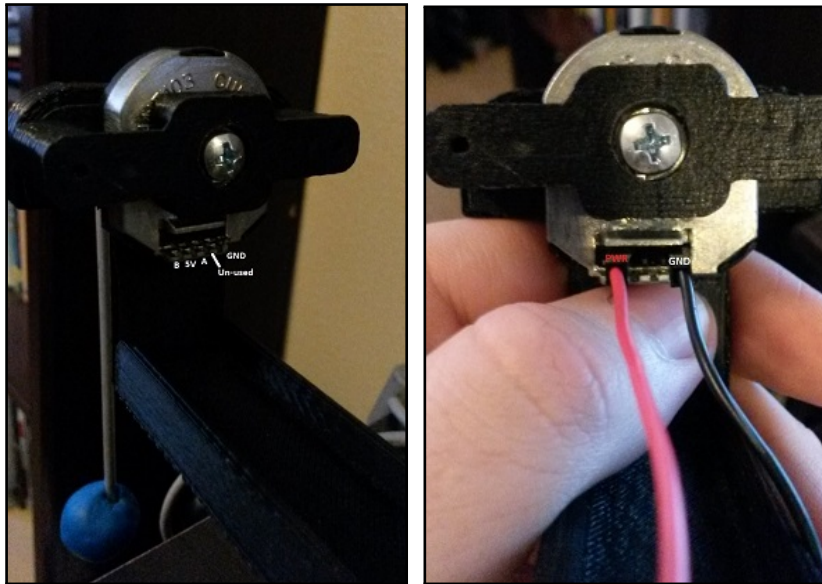


Figure 7.27: Hardware for Connecting the Encoder to the Arduino

30. Obtain the assembly created in the previous section and 4 male to female wires.
31. Connect the female end of 2 wires to the Power and Ground pins of the encoder.

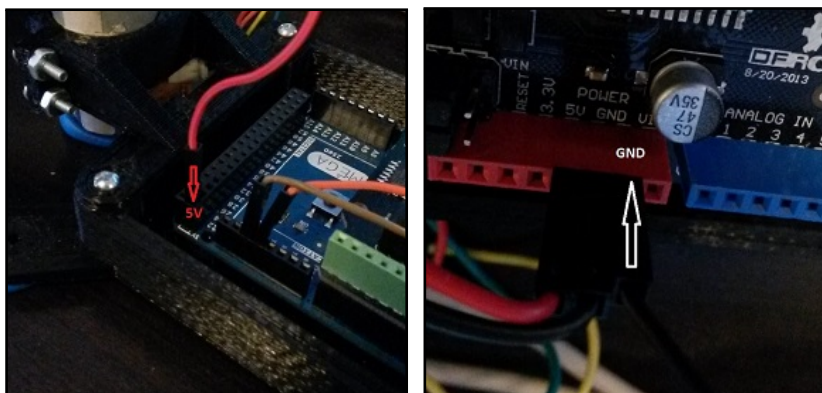


(a) Pins of Encoder

(b) Female End of Wires to Power and Ground On Encoder

Figure 7.28: Pin-out and Power and Ground Wire Connections to Encoder

32. Connect the Power and Ground Wires from the Encoder to the 5V and GND pins of the Arduino, respectively.



(a) 5V Pin Connection on Arduino

(b) GND Pin Connection on Arduino

Figure 7.29: Power and Ground Wire Connections to Arduino

33. Connect the remaining 2 wires' female ends to the Encoder A and Encoder B pins of the encoder.

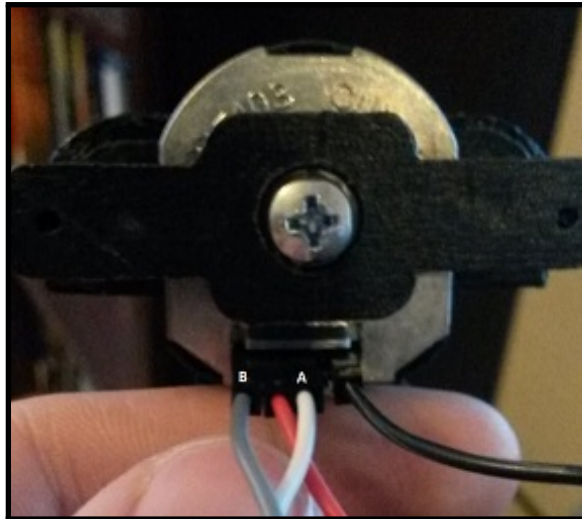


Figure 7.30: 2 Female Wires Connected to Encoder A and Encoder B Pins

34. Connect the Encoder A and Encoder B wires to pins 20 and 21, respectively, on the Arduino.

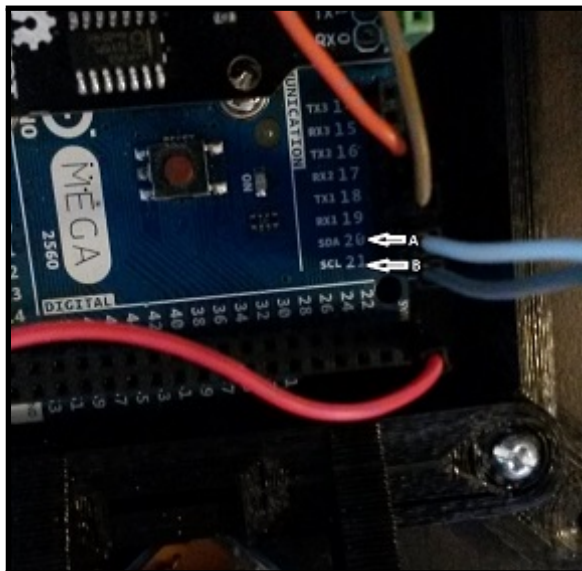


Figure 7.31: Encoder A and Encoder B Wires Connected to Pins 20 and 21 On the Arduino

35. **Optional:** Take a piece of scotch tape and wrap it around the wires and beam.



This will keep the wires more tidy and reduce the chance of individual wires becoming disconnected during runtime.

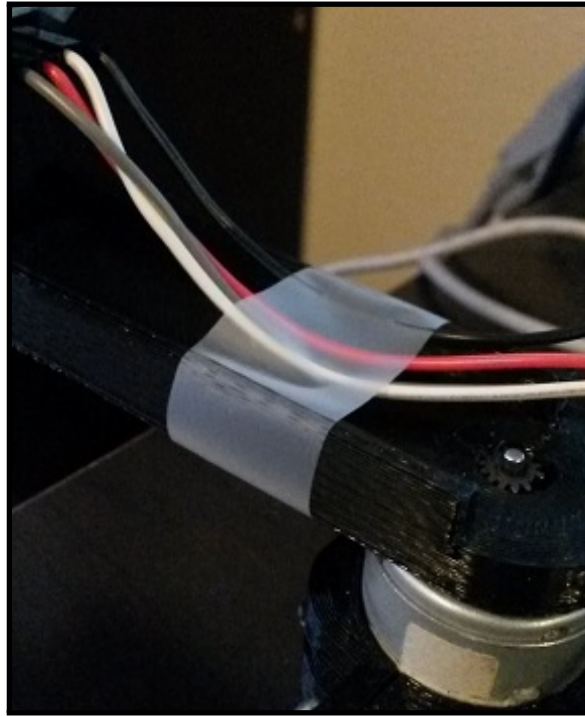


Figure 7.32: **Optional:** Scotch Tape Holding Wires to Beam

### 7.3 Experimental Procedures

This section will describe a physical plant model. For this experiment, you will design a state feedback model using pole positioning. You will also develop Simulink models (linear and nonlinear) that will verify their open loop and closed loop performance, before applying your pole positioning gains to the real system. The experimental portion of this experiment will help you develop a Simulink model that will be loaded to the Arduino. You will then build a file to capture the experimental data and observe the behavior of your real furuta pendulum.

### 7.3.1 Exercise 1: Open Loop System Verification

#### Furuta Pendulum Model

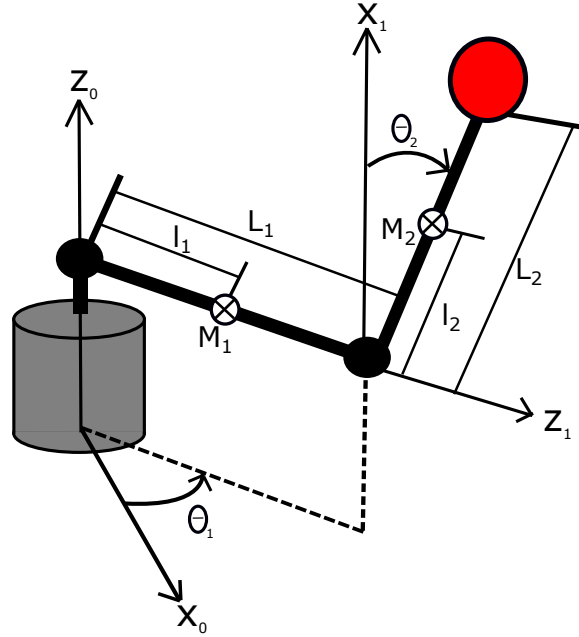


Figure 7.33: Furuta Pendulum Model (Circled x Denotes Center of Mass)

The following equations of motion describe the dynamics of the Furuta Pendulum model:

$$\begin{aligned} \ddot{\theta}_1 &= \frac{C_2 L_1 \cos(\theta_2) \dot{\theta}_2}{l_2 B} + \frac{M_2 L_1 l_2 \sin(\theta_2) \dot{\theta}_2^2}{B} + \frac{\tau}{B} - \frac{M_2 g L_1 \cos(\theta_2) \sin(\theta_2)}{B} \\ &\quad - \frac{C_1 \dot{\theta}_1}{B} - \frac{M_2 l_2^2 \cos(\theta_2) \sin(\theta_2) \dot{\theta}_1 \dot{\theta}_2}{B} \\ \ddot{\theta}_2 &= \frac{-L_1 \cos(\theta_2) \ddot{\theta}_1}{l_2} - \frac{C_2 \dot{\theta}_2}{M_2 l_2^2} + \sin(\theta_2) \cos(\theta_2) \dot{\theta}_1^2 + \frac{g \sin(\theta_2)}{l_2} \\ B &= -M_2 L_1^2 \cos^2(\theta_2) + M_1 l_1^2 + M_2 L_1^2 + M_2 l_2^2 \sin^2(\theta_2) \\ \tau &= [u - N K_m \dot{\theta}_1] \frac{K_t}{R_a} \end{aligned}$$

Where

$\theta_1$  : Angular Position of Base Arm

$\theta_2$  : Angular Position of Pendulum

$M_1$	: Mass of Base Arm
$M_2$	: Mass of Pendulum
$L_1$	: Length of Base Arm
$L_2$	: Length of Pendulum
$l_1$	: Length from Pivot to Center of Mass of Base Arm
$l_2$	: Length from Pivot to Center of Mass of Pendulum
$C_1$	: Viscous Friction Coefficient of Motor Pivot
$C_2$	: Viscous Friction Coefficient of Pendulum Pivot
$g$	: Gravitational Constant
$\tau$	: Torque
$u$	: Motor Voltage
$R_a$	: Armature Resistance of Motor
$K_t$	: Torque Constant of Motor
$K_m$	: Back EMF Constant of Motor
$N$	: Gear Ratio of Motor

Additionally, the States for this model are:

$$x_1 = \theta_1$$

$$x_2 = \dot{\theta}_1$$

$$x_3 = \theta_2$$

$$x_4 = \dot{\theta}_2$$



## Open Loop Verification

36. Using the equations of motion, develop a Simulink model of the Furuta pendulum system. Use the following constants for your Simulink Model:

$$M_1 = 0.0526 \text{ kg}$$

$$M_2 = 0.0155 \text{ kg}$$

$$L_1 = 0.093 \text{ m}$$

$$L_2 = 0.0121 \text{ m}$$

$$l_1 = 0.062 \text{ m}$$

$$l_2 = 0.095 \text{ m}$$

$$C_1 = 0.004$$

$$C_2 = 0.00011$$

$$g = 9.81 \frac{\text{m}}{\text{s}^2}$$

$$R_a = 13.5 \text{ ohms}$$

$$K_t = 0.0267 \text{ N} \frac{\text{m}}{\text{A}}$$

$$K_m = 0.0269 \frac{\text{Vs}}{\text{m}}$$

$$N = 1$$

37. Using an initial condition of a small pendulum angle ( $\frac{\pi}{40}$ ), simulate your model for 5 seconds. (Using step sizes of 0.01 seconds, this should give you about 500 data points.)
38. Plot each state vs. time to verify the open loop performance of your model.
- Does the pendulum and beam move in the correct direction?
  - Do they oscillate?

- What position(s) do they oscillate around?

### 7.3.2 Exercise 2: System Linearization

39. Develop a linear model of the system by using the MATLAB command **linmod**.
40. Compare the open loop response of the linear model with the open loop response of the nonlinear model you created in **Open Loop Verification** subsection.
  - How does the open loop linear response differ from the nonlinear response?
  - Do both the linear and nonlinear beam and pendulum angles follow one another?
  - What angles do the linear beam and pendulum angles settle around?
  - What angles do the nonlinear beam and pendulum angles settle around?

### Expected Open Loop Response (For Instructors)

The expected response for the open loop nonlinear pendulum angle and angular velocity (see Figure 7.34) is for the pendulum to fall, oscillate, and settle around  $\pi$  radians when the initial condition for the pendulum position is  $\frac{\pi}{40}$  radians. The velocity will also oscillate around 0  $\frac{\text{radians}}{\text{second}}$ . If the initial condition was  $-\frac{\pi}{40}$  radians then the pendulum would settle around  $-\pi$  radians. The base (or beam) angular position and velocity (see Figure 7.35) will always settle around 0.

When the open loop nonlinear pendulum response is compared with the linear responses in Figure 7.36, the linear pendulum angle appears to increase to infinity while the nonlinear pendulum angle starts to move in the opposite direction. Since the model is linearized around zero, the linear pendulum response will only follow close to the nonlinear response around zero for small angles. A similar result appears in Figure 7.37 for the base angle and angular velocity.

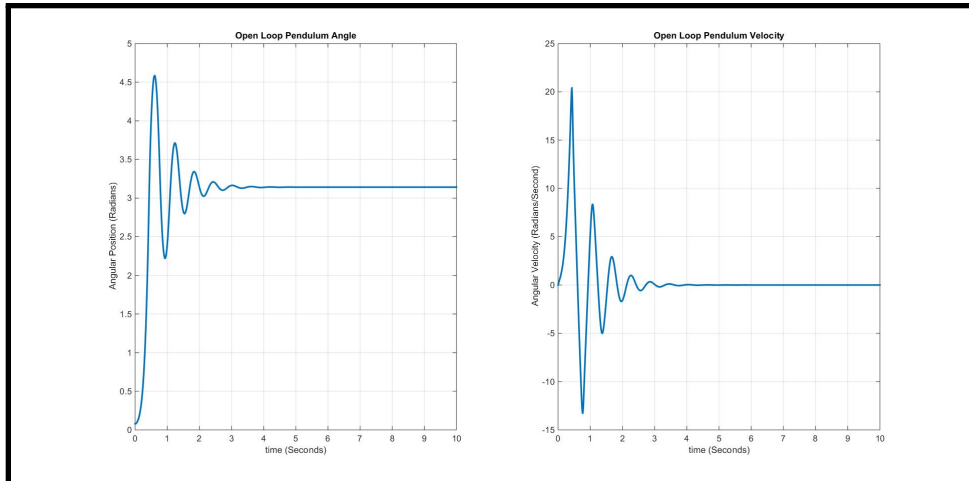


Figure 7.34: Open Loop Pendulum Response (Nonlinear)

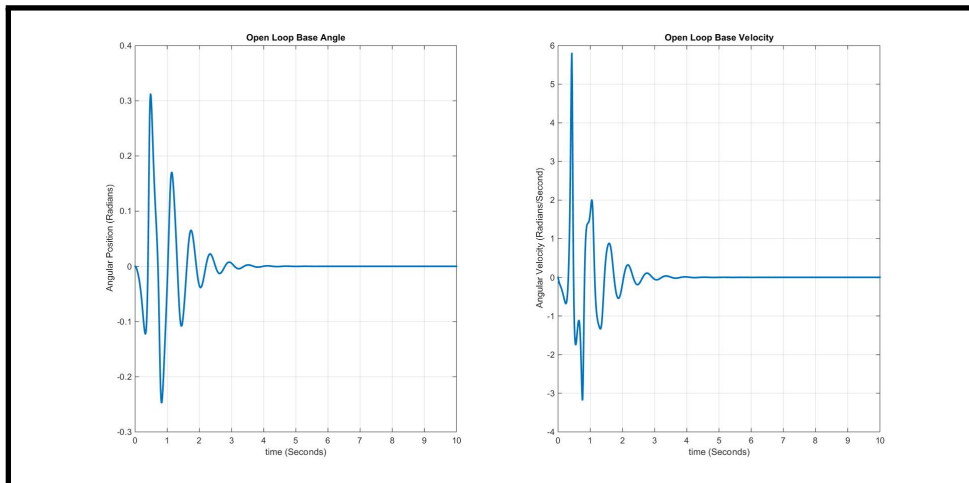


Figure 7.35: Open Loop Beam Response (Nonlinear)

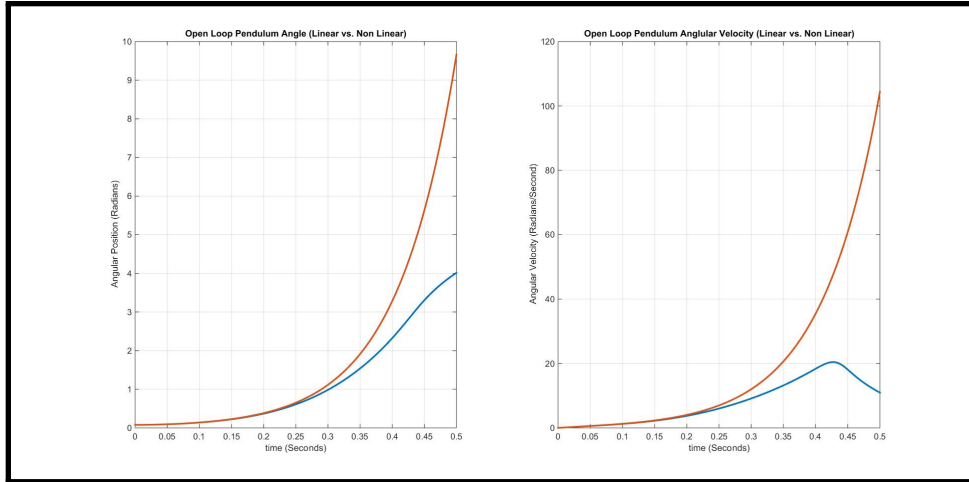


Figure 7.36: Open Loop Pendulum Response (Linear (red) and Nonlinear (blue) Comparison)

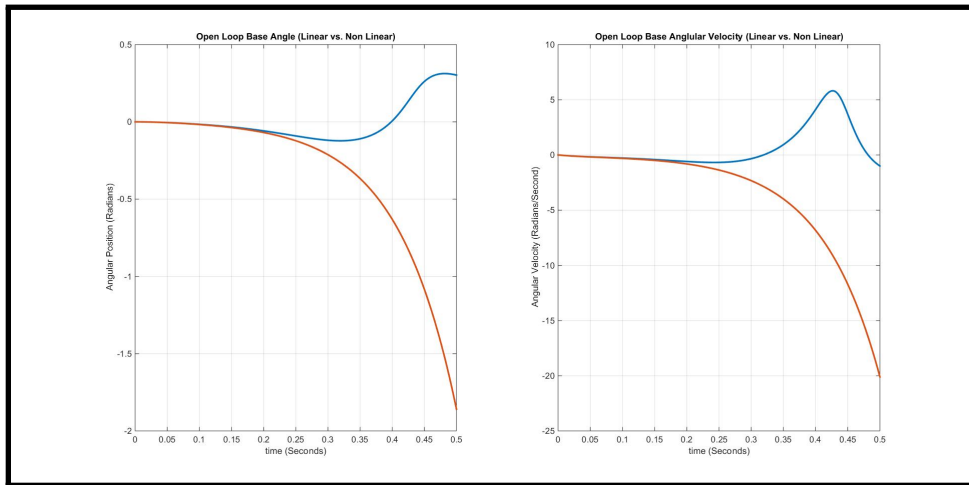


Figure 7.37: Open Loop Beam Response (Linear (red) and Nonlinear (blue) Comparison)

### 7.3.3 Exercise 3: Linear State Feedback Using Pole Positioning

41. Using the linear model, assume that all of the states are measurable. Find the open loop poles of the system using the MATLAB command `pole`.
42. Design the pole positioning state feedback controller. Decide where you would like to place the closed loop poles, and justify your decision.

**NOTE:** Use the MATLAB command **place** to place the open loop poles in your desired locations.

43. Using your linearized pendulum model, simulate the response of the state feedback system for the pendulum position initial conditions of  $\frac{\pi}{40}$ ,  $\frac{\pi}{10}$ , and  $\frac{\pi}{4}$ .

**NOTE:** The base angle and base angular velocity will be two subplots in the same figure. This will be the same for the pendulum angle and pendulum angular velocity, in a separate figure. Plot the input voltage on a separate figure for a total of 3 figures.

- Do the responses match your expectations, based on the pole locations you set? Explain.
- How does the system respond as the initial conditions are varied?
- Does the system become unstable?
- How does the input voltage change as the initial conditions are varied? Are the input voltages achievable?

44. Try to improve your design. Find the quickest possible response within physical limits.

#### **7.3.4 Exercise 4: Nonlinear State Feedback Using Pole Positioning**

45. Apply the gains you found in the previous step to the nonlinear model and simulate the response to the same initial conditions as the linear case.
46. Compare the nonlinear and linear responses by plotting both on the same figure. Plot the base and pendulum angles and the angular velocities on two separate figures. Also plot the input voltages on one figure. Explain any differences between the linear and nonlinear systems.

**NOTE:** Similar to step 42, plot the base angles and velocities (of both the linear and nonlinear feedback responses, in the same plot) as two subplots in the same figure. Do the same with the pendulum angles and velocities. Plot the input voltages of the linear and nonlinear state feedback simulations on the same plot, which produces 3 figures in total.

47. Try to improve your design.

### **Expected State Feedback Results (For Instructors)**

As the initial condition for the pendulum angular position is increased, the error between the linear and nonlinear responses begins to increase. Placing the closed loop poles at  $[-0.80 -36.3985 -11.8414 -16.2092]$  produces the state feedback gains  $K = [-5.7735 -9.5378 -187.3304 -17.9715]$ , Figure 7.38 shows the response for the pendulum angle with a very small initial condition of  $\frac{\pi}{40}$ . The linear (red) and nonlinear (blue) responses for both the pendulum angular position and velocity appear to have a small error. The beam angular position and angular velocity responses in Figure 7.39 also appear to have a small error between the linear and nonlinear responses. Figure 7.40 shows a max input voltage of about 14 volts, which is a reasonable voltage for the physical pendulum system.

When the initial condition is increased to  $\frac{\pi}{10}$  in Figure 7.41, the error between the linear and nonlinear responses for the pendulum angular position and velocity are similar to the previous initial condition responses. The base angular position and velocity plots in Figure 7.42 appear to have an even smaller error between the linear and nonlinear responses when compared with Figure 7.39. However, the largest change is the input voltage in Figure 7.43 where the input voltage increases to a max value of about 58. While this voltage could be attainable in a system where the supply voltage could be 58 volts or higher, the max supply voltage the students will

be using is only 15 volts. This means that the physical system will face difficulties when trying to overcome this larger initial condition.

Figure 7.44 shows the pendulum angular position and velocity for the pendulum when the initial position of the pendulum angle is increased to  $\frac{\pi}{4}$ . While the linear and nonlinear pendulum angular positions and velocities become stable with the initial condition, the error between the linear and nonlinear curves has greatly increased. The same result occurs in Figure 7.45 for the base angular position and velocity linear and nonlinear plots. Additionally, the input voltage in Figure 7.46 rises to about 150 volts, which is not attainable by the power supply.

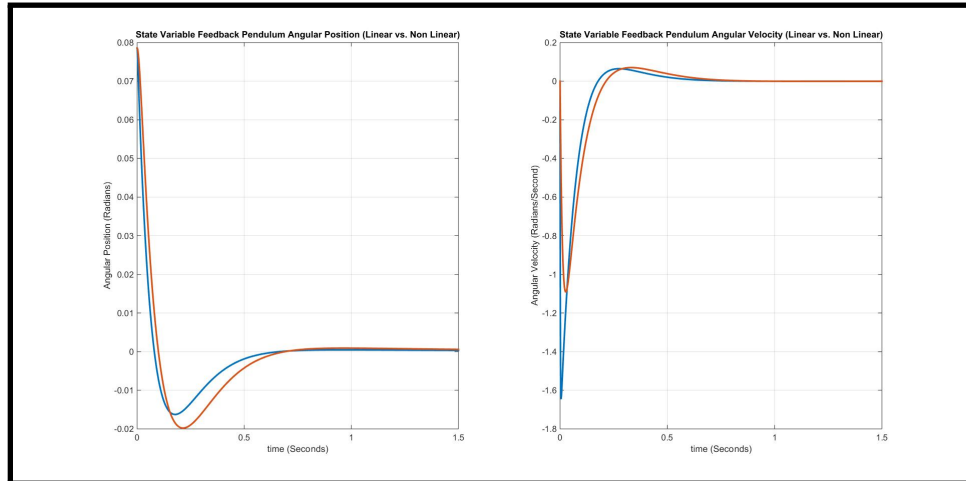


Figure 7.38: Closed Loop Pendulum Response (Linear (red) and Nonlinear (blue) Comparison with a Pendulum Initial Condition of  $\frac{\pi}{40}$ )

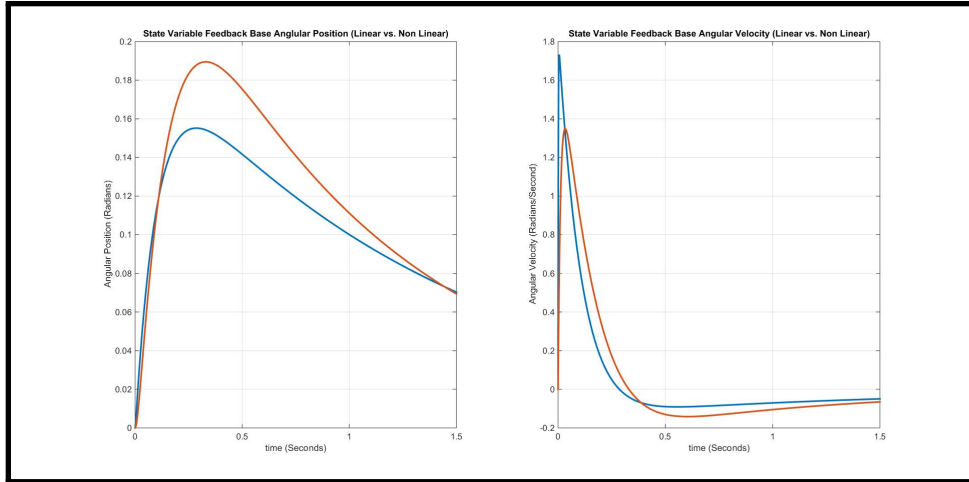


Figure 7.39: Closed Loop Beam Response (Linear (red) and Nonlinear (blue) Comparison with a Pendulum Initial Condition of  $\frac{\pi}{40}$ )

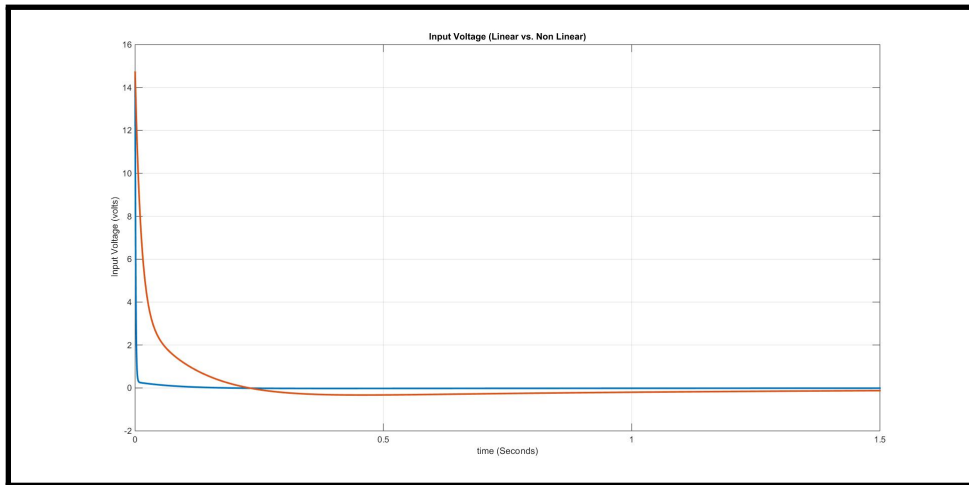


Figure 7.40: Closed Loop Input Voltage Response (Linear (red) and Nonlinear (blue) Comparison with a Pendulum Initial Condition of  $\frac{\pi}{40}$ )  
Initial Condition of  $\frac{\pi}{10}$



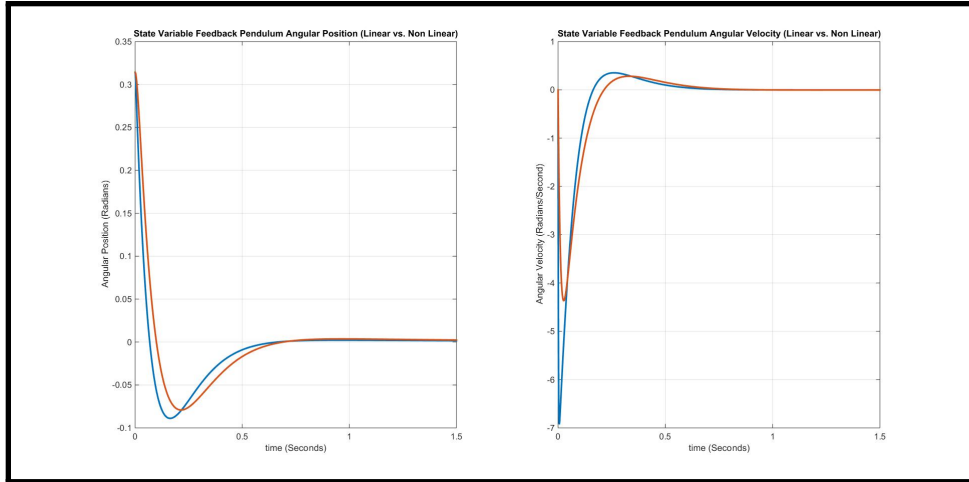


Figure 7.41: Closed Loop Pendulum Response (Linear (red) and Nonlinear (blue) Comparison with a Pendulum Initial Condition of  $\frac{\pi}{10}$ )

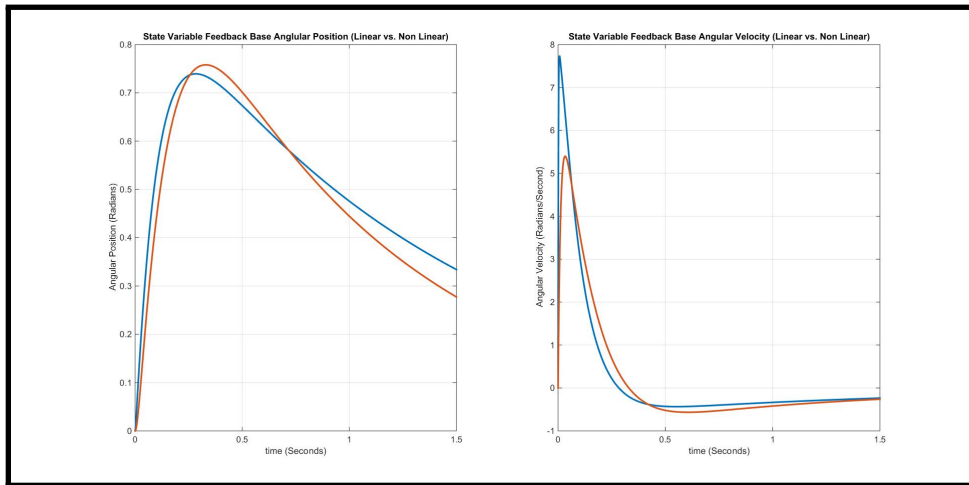


Figure 7.42: Closed Loop Beam Response (Linear (red) and Nonlinear (blue) Comparison with a Pendulum Initial Condition of  $\frac{\pi}{10}$ )

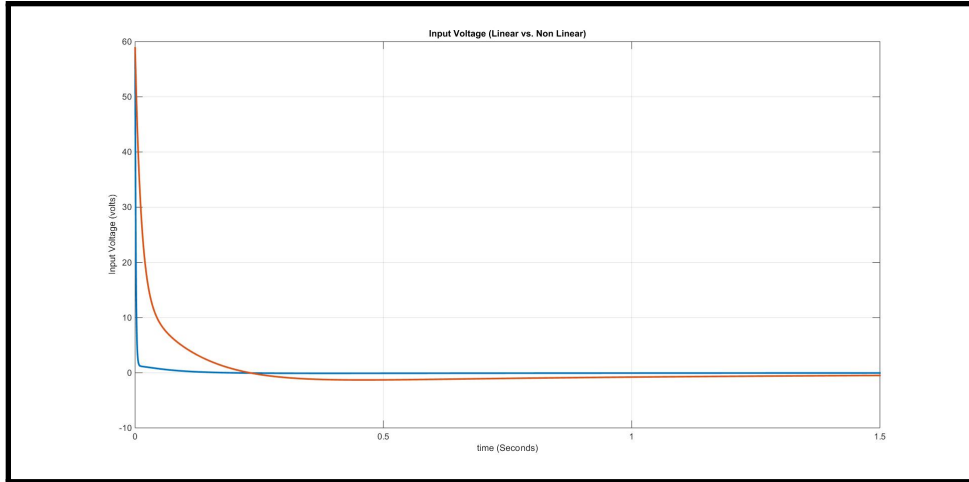


Figure 7.43: Closed Loop Input Voltage Response (Linear (red) and Nonlinear (blue) Comparison with a Pendulum Initial Condition of  $\frac{\pi}{10}$ )

Initial Condition of  $\frac{\pi}{4}$

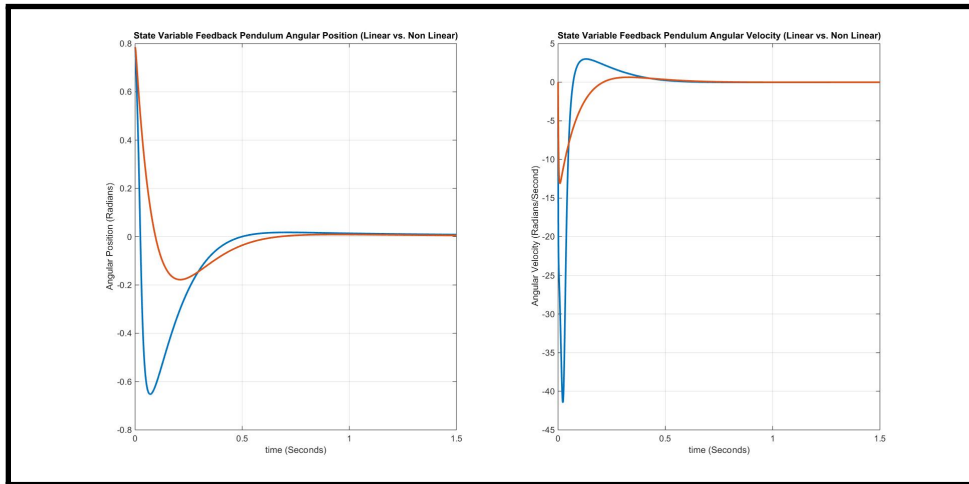


Figure 7.44: Closed Loop Pendulum Response (Linear (red) and Nonlinear (blue) Comparison with a Pendulum Initial Condition of  $\frac{\pi}{4}$ )

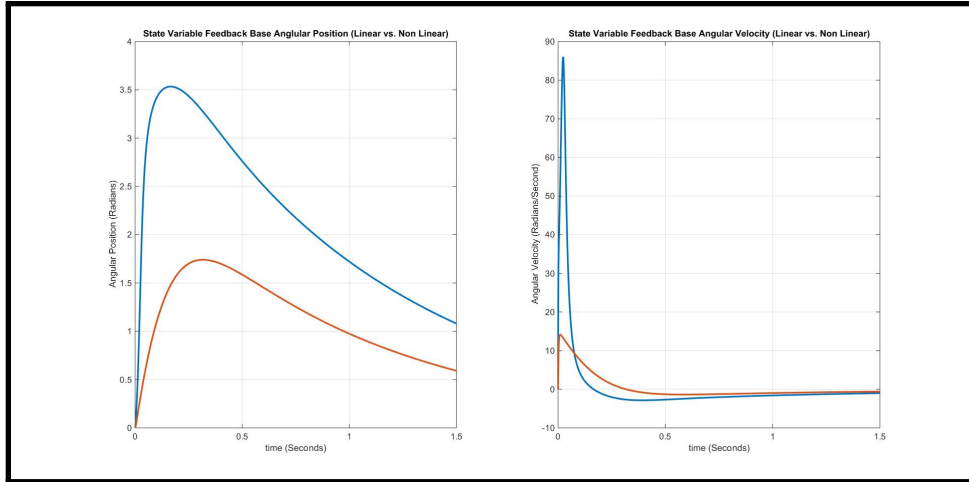


Figure 7.45: Closed Loop Beam Response (Linear (red) and Nonlinear (blue) Comparison with a Pendulum Initial Condition of  $\frac{\pi}{4}$ )

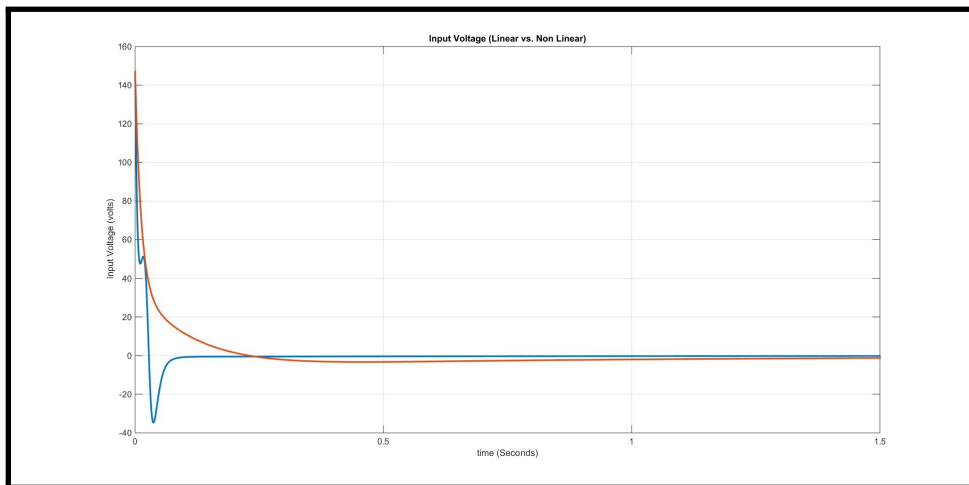


Figure 7.46: Closed Loop Input Voltage Response (Linear (red) and Nonlinear (blue) Comparison with a Pendulum Initial Condition of  $\frac{\pi}{4}$ )

### 7.3.5 Exercise 5: Experimental State Feedback Using Pole Positioning

#### Setting Up Simulink File (Arduino)


On the website for this experiment you will be given a number of Matlab and Simulink files that will be used to gather experimental data from the physical Furuta Pendulum. Follow the steps below to set up the Simulink file with the correct configuration

parameters and workspace constants to load the model to the Arduino and run the Furuta Pendulum.


48. Extract the **Pole\_Positioning.zip** folder to your Desktop.
49. Open the **Constants.m** file provided in the **Pole\_Positioning** folder.
50. At the bottom of the Constants.m file, add the gains you found from your Simulations as a 4x1 vector named **K**. Make sure the sequence of numbers correspond, from the top to the bottom of the vector, as the following:
  - Pendulum angular position (First K element)
  - Pendulum angular velocity (Second K element)
  - Base angular position (Third K element)
  - Base angular velocity (Fourth K element)

51. Run  the Constants.m file.

52. Open the **RIV\_Arduino.slx** file also included in the **Pole\_Positioning** folder.

53. Setup the Configuration Parameters by clicking on the Model Configuration Parameters  button. Ensure that the Solver type is “Fixed-step” with a “Fixed-step size (fundamental sample time):” of **Ts** (which should be 0.01 in the Constants.m file.) Also click on the “Run on Target Hardware” tab on the left and ensure that the “Set host COM port:” is set as either **Automatically** or to **Manually** and you have entered your “COM port number:” as the COM port for your Arduino.

**NOTE:** Refer to **Installing Arduino Mega 2560 Drivers** in the *Simple DC Motor* experiment to find the COM port if you do not remember it.

54. Load the Simulink model to the Arduino by clicking on the **Deploy to Hardware**  button. Do not plug in the power for the Furuta Pendulum.

## Collecting Experimental Data

55. Double-click on the subsystem block labeled “Plotting” in the **RIV\_Arduino.slx** file. Once inside the subsystem block, left-click on the text **Plot Data ‘single’**.
56. The “Plot Se...” box should now be visible. Under “Enter COM port to collect data:” input the COM port of the Arduino. Enter the data type as “single” and the number of samples as “2000.” Click Okay.
57. Hold the reset button (located on the exposed area of the Arduino) and pull the pendulum to the standing position (the sticky tack will be above the encoder.) Once you have the pendulum as vertical as possible, let go of the reset button. Once the data begins to fill the plot window, let the pendulum hang down (to the bottom position.)
58. Plug the power cord from the power supply into the motor shield.

**CAUTION:** Do not put your hands or any other parts of your body in front of the motor load trajectory. If the load does not begin moving immediately after the power is plugged in, immediately unplug the power and check to see if the hardware is connected properly (review the **Hardware Setup** section in the *Open Loop Step Response* experiment for the proper hardware connections.) Once you are assured the hardware is good, begin from step 50 once more.

59. Repeat step 57 again, except this time, when you let go of the reset button, hold the pendulum up until the motor begins trying to stabilize the pendulum to the standing positioning, then let go of the pendulum. After repeating step 57, skip to step 60.


**NOTE:** If at anytime (during the proceeding steps) the pendulum falls completely over, click “Stop” at the bottom of the plot window and close out of the

plotting window. Unplug the power from the motor shield. Revisit your gain calculations and simulations, making sure that they are correct. Once you have fixed the gain values, start from step 50 again.

60. Click **AutoScale** at the bottom of the figure. If the plot expands to a very large number ( $10^{38}th$  power) and appears to be very jumpy (meaning that the values do not look to be “smooth” and vary from extremely positive to negative values,) or if the values are not changing from zero, proceed to the next step. If the plot has 4 different lines (around the values 0, 50, 75, and 100) and they all appear to be smooth (each signal is hovering around fixed values), then skip to step 62 in this section.
61. Click the **Adjust Byte** button at the bottom of the screen. Continue to do this until it appears no values are jumping to high and low values. Once the data looks smooth, allow the junk data to empty the screen, and once it has left the screen, click **AutoScale** once more. Continue to repeat this step until the data auto-scales to the 4 different values mentioned in the previous step.
62. Let the data fill the plot window as it moves to the left. Once the data (starting with a value of zero) has filled the screen completely, click the “Stop” button at the bottom of the screen. You should now have 2000 data points (from all 4 states) on the screen.
63. Unplug the power from the pendulum system then navigate back to the **MAT-LAB 2014b** main page. Under “Workspace” the variable **WindowDat** should now be present. Right-click on it and click “Save As.” Name the file **PP\_expResp\_1.mat** and save it into the **Pole\_Positioning** folder.
64. You now have the experimental data for pole positioning on the physical system.

### 7.3.6 Exercise 6: Pole Positioning (Experimental vs. Simulations)

#### Experimental Plotting File

65. Open the **PP\_Plotting.m** file that was provided in the **Pole\_Positioning** folder.
66. Click Run .
67. Observe each plot. Compare each experimental plot with the states you found in your simulated responses:
  - Does the voltage appear to respond correctly to the changes in the pendulum angle?
  - Pick the same point on the pendulum angle, pendulum angular velocity, base angle, and base angular velocity plots and multiply each by their respective pole positioned gains then add those values together. Is the value close to the input voltage at that point?
  - Overall, do the experimental responses behave in a way that you would expect?
  - Does the physical pendulum fall over or does it stay up while the system is energized?
  - See if you can improve the system response by redesigning the state feedback by adjusting the desired closed loop poles. Explain your procedure.

#### Expected Experimental Vs. Simulation Results (For Instructors)

Figures 7.47 - 7.49 show the pendulum angle and angular velocity, base angle and angular velocity, and the input voltage experimental results, respectively. Since each plot has a lot of varying data, it is really only practical to look at small sections of the

data to see if it is behaving as expected. Compare, for instance, the input voltage plot in Figure 7.49 from  $t=0$  to the first peak, and the corresponding pendulum angle result in Figure 7.47. As the pendulum angle becomes negative from time zero to the first valley, the input voltage becomes positive to overcome the negative pendulum angle. In fact, the largest gain for this controller corresponds to the pendulum angle. This is because the pendulum angle is inherently unstable, thus the gain has increased the weight of the error on the pendulum, so the motor will correct the error for small variations in the pendulum angle. Additionally, the input voltage never goes over 10 volts, indicating that the angles of the pendulum never become greater than  $\frac{\pi}{40}$ . The physical system is able to keep the pendulum upright. These responses in the figures below were found using the gains indicated in the **Expected State Feedback Results (For Instructors)** subsection. Using pole positioning, students should vary their closed loop poles to achieve the quickest, most stable response, while keeping the input voltage from exceeding the supply voltage (or rated motor voltage). Additionally, students should expect to have a larger gain for the pendulum angle than any other gain, since it is the most important state to control.

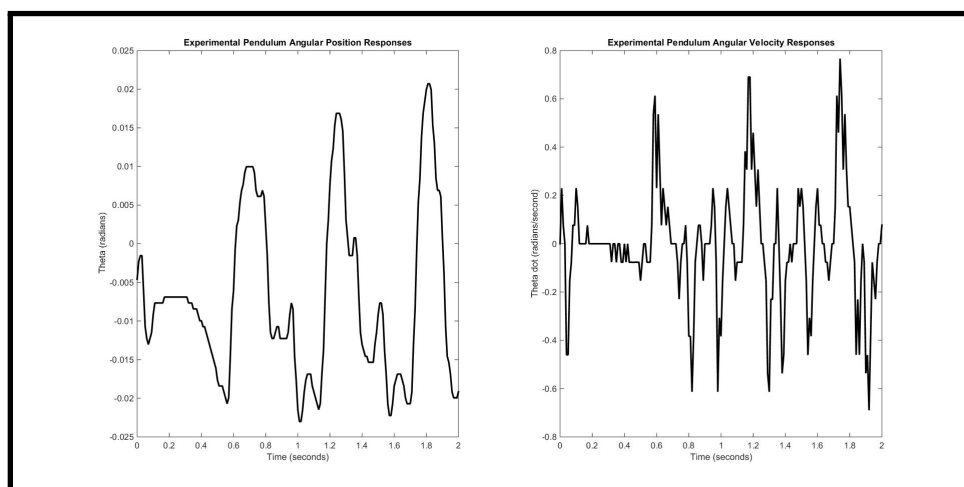


Figure 7.47: Experimental Results (Pendulum Angle and Angular Velocity)



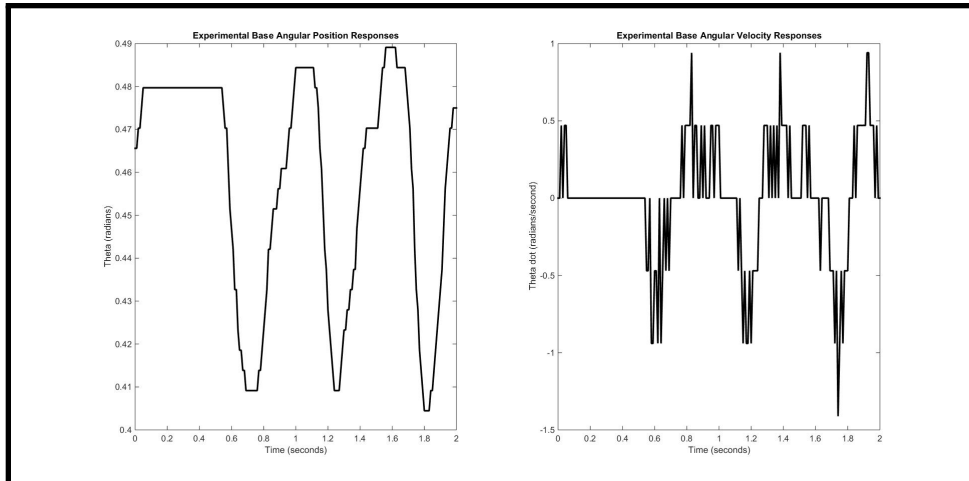


Figure 7.48: Experimental Results (Base Angle and Angular Velocity)

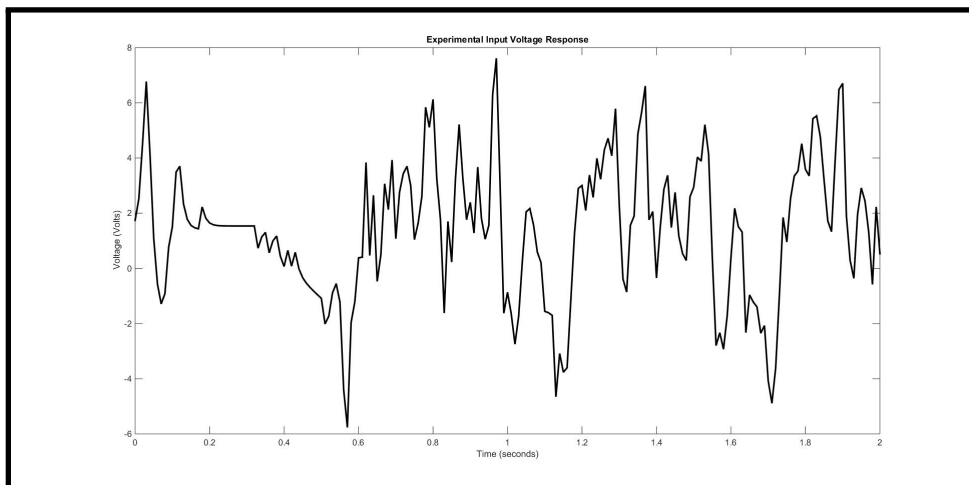


Figure 7.49: Experimental Results (Input Voltage)

## 7.4 Conclusion

This chapter provided an experiment that explains the process of building, simulating, and running a Furuta Pendulum control system. Students were tasked with finding the open loop poles of the system and then placing the closed loop poles in locations that make the system stable. Simulink simulation and experimental files were created and their results were compared and contrasted.

## CHAPTER 8

### CONCLUSIONS

#### 8.1 Summary

The objective of this thesis was to develop inexpensive experiments that can be performed at home, without the need of university resources, including lab space and TAs. This was achieved by creating the Take Home Labs website (<http://thl.okstate.edu>), which provides links to experiment handouts and to materials needed to complete experiments. The software and hardware were kept inexpensive by using university resources such as MATLAB and Simulink and 3D printing, which are typically free to students at many universities. Additionally, lists of inexpensive, off the shelf hardware kept the cost of each experiment down. Experiment handouts were carefully designed, so that students can perform the experiments on their own, without the help of TAs. Background material required of the students are theoretical concepts and not technicalities associated with using the equipment.

#### 8.2 Contributions

This type of lab concept has not been done before. All of the materials that make up entire courses cost less than the price of a text book. Additionally, the labs are infinitely expandable, in terms of the number of experiments that can be included, and are “Open Source” meaning that anyone can contribute new labs and courses with the handout templates available on the website. The labs are not limited to control systems. Subjects such as signal processing, circuits, vibrations, instrumentation, etc.

could be included on the website.

The software contributions included in this thesis are the Dual Encoder Block created in Simulink, and all of the simulation and experimental Simulink and Matlab files created in the Simple DC Motor, Open Loop Step Response, Closed Loop Step Response, and Pole Positioning experiments. Hardware contributions consist of the 3D printed motor load from the Open Loop Step Response and Closed Loop Step Response experiments, and the 3D printed hardware included in the Furuta Pendulum experiment. Additionally, the experiment handouts and instructor guides for each experiment were also developed, so that students may be able to perform the experiments at home without a TA.

While many different software and hardware components were used to create the experiments in this thesis, there were additional topics that were considered, but not implemented. The next subsection will provide ideas for future Take Home Labs contributions.

### **8.3 Future Work**

There were some topics visited while creating the Take Home Labs experiments that will require further attention. For instance, additional software that could be included in experiments are the following:

- National Instruments Labview
- MAC OSX support
- Application support for both Android (Java) and Apple (Xcode) products
- Later versions of MATLAB/Simulink (after R2014b)
- 3D modeling using AutoCAD or Solidworks.

This software was not included in existing experiments due to the time and cost it would take. However, it is possible for external contributions to add these topics to the website by using the handout templates. Additionally, other types of hardware that could be added to future experiments include:

- Microcontrollers compatible with Simulink, including Raspberry Pi and Beagle Board.
- Sensors, including but not limited to, solenoids for fluid or air systems, temperature sensors (such as a thermocouple), inertial measurement units (IMU), gyroscopes, and accelerometers.
- Linear actuators, including pneumatic, mechanical, and hydraulic
- Other rotational actuators, including additional brushed motors, stepper motors, etc.

The experiments that have been developed so far have used three types of physical systems: DC motor with load, Furuta pendulum, and the ball on beam (see Ref. [8]). There are many other interesting systems that could be added: magnetic levitation, cart and pole, Segway, liquid level, flexible beam, etc. Also, although most of the experiments covered here are related to dynamic systems and control, the Take Home Labs concept is suitable for many other topics, including digital signal processing, circuit analysis, vibrations, instrumentation, fluid mechanics, electronics, mechatronics, etc.

## BIBLIOGRAPHY

- [1] M. K. Jouaneh and W. J. I. Palm, “Control systems take-home experiments,” *Control Systems, IEEE*, vol. 33, pp. 44–53, 2013.
- [2] “Take home labs.” <http://th1.okstate.edu>. Accessed: 2015-08-16.
- [3] M. Hagan and C. Latino, *A Modular Control Systems Laboratory*, vol. 3. Comput. Appl. Eng. Education, 1995.
- [4] M. Hagan, “A control systems laboratory with microcomputer supervision,” *Control Systems Magazine, IEEE*, pp. 15–19, 1984.
- [5] P. Horacek, “Laboratory experiments for control theory courses: A survey,” *Annual Reviews in Control*, vol. 24, pp. 151–162, 2000.
- [6] “Quanser.” <http://www.quanser.com/>. Accessed: 2015-08-16.
- [7] Minseg.com, “The miniature balancing robot: A low-cost mobile lab experiment kit for education.” <http://minseg.com/>, 2000. Accessed: 2015-08-16.
- [8] C. Pelton, “Take home labs and the ball and beam,” Master’s thesis, Oklahoma State University, 7 2015.

## VITA

Sean Hendrix

Candidate for the Degree of  
Master of Science

Thesis: TAKE HOME LABS AND THE FURUTA PENDULUM

Major Field: Control Systems

Biographical:

Personal Data: Born in Stillwater, Oklahoma, United States of America on May 2, 1989.

Education:

Received the B.S. degree from Oklahoma State University, Stillwater, Oklahoma, United States of America, 2012, in Electrical Engineering

Completed the requirements for the degree of Master of Science with a major in Electrical Engineering with a Control Systems option at Oklahoma State University in December, 2015.

Experience:

Worked for Tinker Air Force Base, Del City, Oklahoma as electrical engineering intern in 2010.

Worked for Sandia National Laboratories, Albuquerque, New Mexico as robotics intern during the summer of 2011.

Worked for Sandia National Laboratories, Albuquerque, New Mexico as robotics intern during the summer of 2012.

Worked for Sandia National Laboratories, Albuquerque, New Mexico in Masters Fellowship Program from February 2013 - May 2015.

Works for Sandia National Laboratories, Albuquerque, New Mexico as control systems engineer from June 2015 - present.