TAKE HOME LABS AND THE BALL AND BEAM

By

CARION PELTON

Bachelor of Science in Electrical Engineering

Oklahoma State University

Stillwater, Oklahoma

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 BALL AND BEAM

Thesis Approved:

Dr. Martin Hagan
Professor
Department of Electrical and Computer Engineering
Thesis Advisor

Dr. George Scheets
Associate Professor
Department of Electrical and Computer Engineering

Dr. Keith Teague
Professor
Department of Electrical and Computer Engineering

ACKNOWLEDGMENTS

I would like to express my sincerest gratitude to my thesis advisor, Dr. Martin Hagan for his continuous support throughout my research. His assistance, patience, and immense knowledge over the past few years was insurmountable. I cannot imagine being advised by anyone else, and I am truly thankful that I had the opportunity to work with him.

I would also like to thank my fellow lab-mates Sean Hendrix, Amir Jafari, and Mesfin Taye. Sean and I worked jointly on the Take Home Labs, and the encouragement we provided each other made the stressful days much easier. Also, our friendship enabled us to work well together, which is one of the reasons our research was a success. Amir's knowledge of many subjects, his willingness to help in anyway possible, and his advice throughout my research meant a lot and was very helpful. Mesfin's willingness to test experiments, his encouragement, and his kindness was also helpful.

Lastly, I would like to thank my family, my friends, and my girlfriend for their constant encouragement and belief in me. Without their support I would not have been able to stay focused and persevere through the tough times.

Acknowledgements reflect the views of the author and are not endorsed by committee members or Oklahoma State University.

Name: Carion Pelton

Date of Degree: December, 2015

Title of Study: TAKE HOME LABS AND THE BALL AND BEAM

Major Field: Electrical Engineering

Abstract: The work in this thesis discusses the creation of affordable, simple, and high quality engineering laboratory experiments that can be done by students at home. The experiments will enhance the student's understanding of theoretical concepts within various engineering courses by providing a hands on learning experience. These experiments use MATLAB/Simulink in conjunction with an Arduino. The Arduino is an affordable open source microcontroller which has support within MATLAB/Simulink, thus allowing students to easily access the functionality of the Arduino without needing to have significant programming skills. Five experiments are created and presented, and for each experiment, additional hardware components are either purchased or 3-D printed. The first experiment is an introductory experiment for setting up the main hardware. The second experiment is another introductory experiment that uses a DC motor with encoder, a motor driver, and a 3-D printed load to help teach the concept of Sampling and data acquisition with the Arduino. The third and fourth experiments use essentially the same hardware as the Sampling and data acquisition experiment, and are created to help illustrate the importance of frequency response by performing the open loop and closed loop frequency response of the DC motor with an attached load. The final experiment helps teach the concept of optimal state feedback control of a Ball and Beam system. The experiments are provided as handouts to the be followed by the students. The handouts, hardware lists, software files, and 3-D printer files are provided on our website. The website makes the labs easily accessible to anyone interested, and keeps the experiments organized. The combination of the experiments, the hardware and software, using a 3-D printer, and the website results in an affordable, simple, and high quality lab kit with potential to be expanding into multiple engineering courses. Sean Hendrix and I jointly developed the Take Home Labs concept.

TABLE OF CONTENTS

Chapter Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

CHAPTER 1

INTRODUCTION

1.1 Problem Statement

In engineering courses, laboratory experiments provide students with a hands on learning experience to go along with the theoretical concepts being taught in the course. However, the number of courses that have dedicated labs is limited. Laboratory experiments usually require extra space for students to work, expensive equipment, teaching assistants, or limited lab time. Typical electrical engineering courses that have dedicated labs are circuits courses, embedded programming courses, and digital logic courses. Other courses usually rely on simulated homework and projects to help teach the theoretical concepts of the course. Examples of these types of courses are control systems and signal processing courses.

With the emergence of small, powerful, and affordable open source microcontrollers, the potential for expanding the use of laboratory experiments into other engineering courses is more feasible. This thesis discusses the creation of affordable, simple, and high quality engineering laboratory experiments that can be done at home.

1.2 Literature Review

This section discusses the current options for laboratory experiments, mainly related to control systems courses.

A control systems laboratory using a microcomputer is described in [2]. The laboratory is its own dedicated course that is designed to be taken concurrently with an introductory control systems course. The dedicated hardware is a lab station, and the main component is a DC motor. In the addition to the DC motor, the lab station includes; a computer for controlling the execution of each experiment, a digital to analog (D/A) converter for interfacing the motor with the computer, a power amplifier for amplifying the output current from the D/A converter, a sensor for capturing the motor's motion, and a counter for counting the pulses from the sensor to record the motor's position and velocity. The lab station is to be used to teach the concepts of open and closed loop step response, open and closed loop frequency response, stability, and root locus design. The lab station does not require any setting up by the students, which ensures their time can be spent on understanding the concepts. At the time of the paper, the lab had been run for one semester, and the students showed a noticeable improvement in their understanding of the theoretical concepts. The lab station was relatively large, limited to a DC motor, and focuses only on control systems.

In [1], a modular controls laboratory is created to help students in senior level controls courses visualize and evaluate the performance of simulated results of a system, when applied to the actual physical system. Multiple systems are designed and used to provide a range of real systems for teaching controls. In addition to the controls courses, the modular controls lab was able to add to the capstone design course by allowing student teams to design new systems to be added to the overall controls lab. In the controls course, the students learned theoretical control concepts and also learned about real-time programming, effects of sampling rate, use of interrupts, importance of maintaining correct units, effects of nonlinearities such as friction, and the iterative process of design. These additional concepts are not easily learned from simulations. Six experiments were completed; the Ball on Beam, the Magnetically Levitated Sphere, the Cart and Pole, the Wedge Balancer, the Two-Link Pendulum, and the Inverted "T". The experiments were designed to be very modular, making it easy to upgrade the labs as new technology emerged. The types of systems designed and proposed are shown in Figure 1.1. The systems were fairly large and mainly related to control systems.

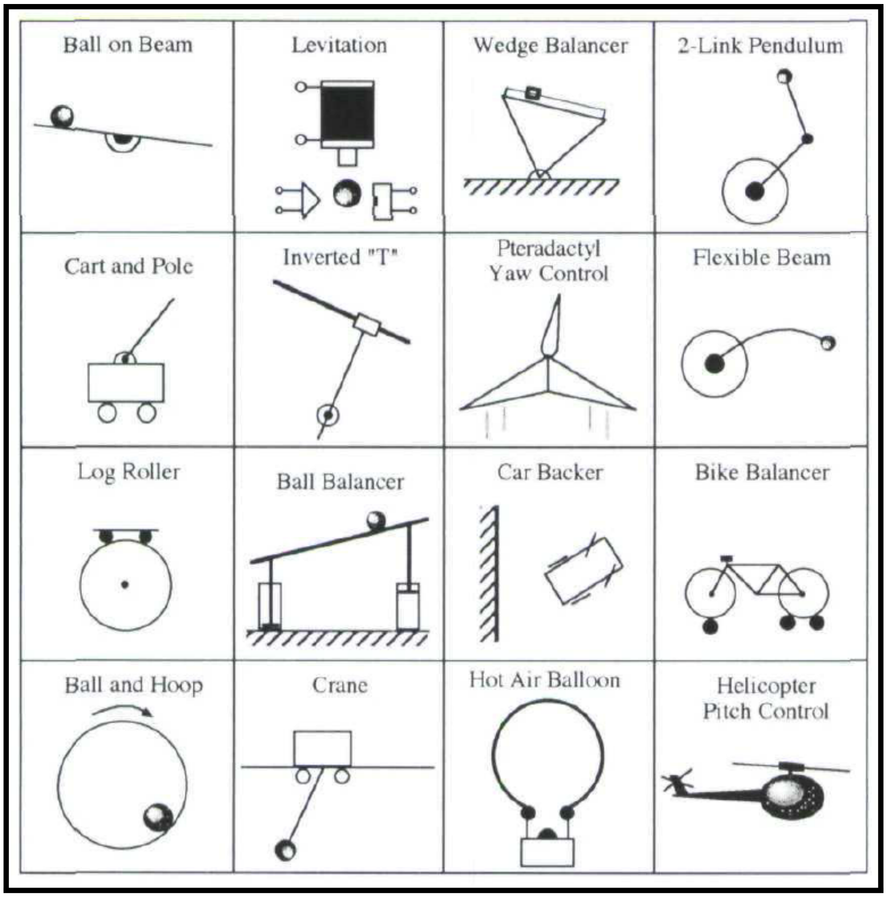


Figure 1.1: Exerimental Systems [1]

In [3], a fourth-order, linear mass-spring-damper system and an analog filtering system were designed and packaged with a controller board for less than \$100, and provided for students as at-home system kits. A take home hardware kit for a Digital Design Lab was developed for students in [4]. Two experiments that involve a DC motor/tachometer system and a heater/temperature sensor system are provided in [5]. The low cost hardware experiments were designed and used in several mechanical engineering courses, and proved to be effective and enjoyable via student feedback. A low cost lab kit platform based on an Arduino microcontroller is designed for teaching technology of modern display systems is discussed in [6]. Student surveys showed high praise for the overall lab kit, and specifically the Arduino platform. A low-cost rapid control prototyping system using MATLAB/Simulink and an Arduino is discussed in [7]. The Arduino and MATLAB are also used in [8] as part of take home lab kit designed by six mechanical engineering senior design students at the University of Minnesota.

Experiment systems can also be purchased online for learning control design. This makes the take home systems available to anyone, and not just the students in a particular course. Quanser designs and manufactures advanced systems for real-time control design and implementation [9]. Quanser has a large catalog of systems. Each system is high quality, and comes with courseware developed by Quanser. Each system is also compatible with MATLAB/Simulink and LabVIEW. The high quality of these systems results in a high cost. MinSeg however, offers versions of a miniature balancing robot for an affordable cost [10]. The kits can be purchased at various prices depending on the board that comes with the system. The options are an Arduino compatible board, an actual Arduino, and a Raspberry Pi. The labs are designed to work with Simulink, which has support for each board. A series of experiments and a Simulink support library is available to be used with the kit, and the experiments are designed for use in a mechatronics course.

The existing literature and various options express the importance, benefits, and interest in developing lab kits for use in engineering courses. There have been multiple laboratory experiments introduced, but they are either too expensive, too large, confined to a lab space, not easily accessible for everyone, or not very adaptable. The following section discusses the approach for making a take home lab kit that does not experience any of these problems.

1.3 Summary of Approach

The overall concept that is discussed in this report is the design of affordable, simple, adaptable, and easily accessible take home labs that can be used to provide a hands on learning experience for students within various engineering courses.

The Take Home Labs are designed to be small, so lab space is not needed. Students can work on the experiments at home, at their own pace. To keep the labs affordable, low cost open source hardware, 3-D printing, software that is widely available to students, and low cost components are used. The overall cost of the system is similar to the price of a textbook. To make the labs easily accessible, a website is designed to organize and distribute the experiments to anyone interested. The experiments designed are introductory, undergraduate, and graduate level, and are available on the website as handouts that are made to be simple, easy to follow, and used in conjunction with specific courses.

The main hardware component is an Arduino, which is an open source microcontroller that has support in MATLAB/Simulink. The experiments are therefore designed heavily around MATLAB/Simulink, which does not require that the students have significant programming skills. The Arduino is a powerful microcontroller, so experiments for multiple engineering courses can potentially be designed, adding to the flexibility of the overall Take Home Labs. 3-D printing portions of the experiments allows for quick prototyping and various design possibilities. The 3-D printed systems are designed so that they can be used for multiple experiments, and multiple courses. Having a website available not only organizes the experiments, it is used to encourage participation from anyone by submitting comments, questions, or their own labs, thus keeping the experiments up to date, and potentially adding new experiments. Combining each of these elements enables an affordable, simple, small, and adaptable take home lab kit to be created. The Take Home Labs concept was developed jointly in this thesis and in [11]. Five experiments are described in this thesis, and five complementary experiments are described in [11]. Together, these documents represent the current state of the Take Home Labs concept.

1.4 Chapter Outline

This chapter has introduced the problem of limited laboratory experiments for engineering courses, and the approach for overcoming this issue. Chapter 2 further discusses the overall concept, and introduces the overall system design of the Take Home Labs. Chapter 3 is an introductory experiment for setting up the main hardware and software that is used for most of the other experiments. Chapter 4 is also an introductory experiment that will review the concept of sampling, show how to acquire data from the Arduino in real-time, at multiple sampling rates, using Simulink, and provide insight into the differences between Normal and External Mode for sampling and acquiring data. Chapter 5 is a Systems Dynamics dedicated experiment that will explore the concept of frequency response by finding the open loop frequency response of a DC motor with an attached load. Chapter 6 is also a Systems Dynamics dedicated experiment that will further explore the concept of frequency response by using the results from the open loop frequency response, and finding the closed loop frequency response of the DC motor with an attached load. Chapter 7 is an experiment designed for a graduate Optimal Control course. The experiment requires that the student design an Optimal State Feedback Controller for a Ball and Beam system. The decision for the software, hardware, and the overall framework of the Take Home Labs is discussed in the next section.

CHAPTER 2

SYSTEM OVERVIEW

The purpose of the Take Home Labs concept is to provide access to a set of inexpensive laboratory experiments that can be performed in a dorm room. The two major components of the overall system design are the software and hardware. The hardware and software that are used for the experiments are chosen based on price, simplicity, and size. Making the entire system affordable is the top priority, as it is targeted for college students. Affordable in this case means keeping the system cost below $150, which is around the normal cost of a college textbook. To also ensure that the experiments developed for the system can be done at a student's home, the size of the entire system should not be very large. There will likely not be a teaching assistant to help students with the experiments, so the software chosen is easy to use, the lab handouts are easy to follow and easily accessible, and the hardware components are easily interfaced with the PC and easily built. This chapter discusses the different elements that make up the overall framework for the Take Home Labs, and their significance.

2.1 Software

2.1.1 Operating System

All of the experiments require using a personal computer (PC) for developing the experiments. Therefore, designing the experiments to be compatible with an operating system that most students use is important. The program for running the experiments, and the hardware components, need to be compatible with the chosen operating system. Engineering students at Oklahoma State University (OSU) are recommended to have a laptop with Windows installed on it, as Windows is required for some applications [12]. At other universities, it is likely that most students will have a PC that runs Windows, thus Windows is the likely the best platform for running the experiments. Another popular operating system is Macintosh (Mac), and students with a Mac have the ability to install Windows on their PC. The potential need for

Mac users to install Windows on their PC may be cumbersome, so the experiments are also being designed to work solely on Macs. The main software used for designing the experiments is compatible with both Mac and Windows.

2.1.2 Main Software Program

The software that is used for running and designing the experiments is MATLAB. In MATLAB, the experiments are mostly designed using Simulink, which is a program integrated into MATLAB. Simulink is a great choice because it is a graphical programming environment that can be used to model, simulate, and analyze systems, in a block diagram form [13]. Electrical and Computer Engineering courses at OSU use MATLAB extensively for homework and/or projects. Simulink is a great tool for designing and simulating control systems, which is one of the main focus areas of the Take Home Labs. All engineering students enrolled at OSU can download MATLAB for free through the College of Engineering, Architecture, & Technology (CEAT). Most other universities also use MATLAB, and provide their students with free access. For students attending universities that do not provide access, a student version of MATLAB can be purchased for $99. That fact that MATLAB is widely available to many college students for a reasonable price, or even for free, makes it very suitable for implementing the experiments. Another reason Simulink is a great choice is the built in support it has for many embedded hardware platforms. Without Simulink, most of the hardware platforms require writing code to use its functions. Simulink eliminates the need for writing embedded code, because many of the embedded hardware functions are represented as Simulink blocks. Custom Simulink blocks called S-Functions can also be created. This makes dealing with other hardware elements easier, because the code needed to interface the sensors to the embedded hardware can be written into a custom S-Function block, and added to the Simulink library. This helps to keep the experiments simple and organized. To prevent students from needing to make their own S-Function blocks, a custom Simulink library containing S-Functions and other helpful blocks will be provided. In the *Sampling and Data Acquisition* experiment, the students will add the custom library, THLlib, to their Simulink library. The library is an altered version of the MinSeg RASPlib. The MinSeg library already has S-Functions for reading one quadrature encoder, reading two quadrature encoders, and reading an ultrasonic sensor. In addition to the S-Functions, the RASPlib has a click function along with custom Simulink blocks to read different types of data sent to the serial port. The RASPlib is very helpful in learning how to create custom S-Functions, and interface with different sensors. The S-Functions and custom Simulink blocks within the THLlib are modified versions of the RASPlib blocks, as well as new blocks created by myself and fellow research students that are contributing the the Take Home Labs concept. To modify and create new S-Functions, the embedded code used to program the embedded microcontroller had to be modified or created within the S-Function block.

With Simulink, the experiments can be designed and simulated easily. The simulation models and experimental models are similar, making it easy to convert the simulation models quickly to run on the hardware. Running the Simulink model on the hardware just requires preparing the Simulink model to run on the desired hardware in the settings, and then downloading the model to the board. Simulink automatically converts the model into the embedded code that runs on the hardware. Simulink has more built in functionally for some hardware than others. Therefore, a major factor in choosing the experiment's embedded microcontroller is its compatibility with Simulink.

2.2 Hardware

2.2.1 Main Hardware

In order to help keep the cost of the system down, the main hardware components are chosen so that they can be used in multiple experiments. The connection between the software and hardware for the experiments is handled by an Arduino. An Arduino is an affordable open-source microcontroller, and is the most important element of the system. The experiments can not run without the microcontroller, and the low cost, small size, and power of the Arduino help make the take home labs practical for college students. Arduino has many different models of microcontrollers available. The Arduino MEGA 2560 R3 is the model recommended for running the experiments. The Arduino Uno is a cheaper alternative that can also be used to run the experiments, but it is not as powerful as the Arduino MEGA 2560 R3. The Arduino MEGA 2560 R3 typically costs around $35 - $50, and the Arduino Uno typically costs around $20 - $30. Simulink supports the Arduino boards, and accessing the functionality of the boards using Simulink will be much easier than having to write a lot of code. Figure 2.1 shows an example of some of the Arduino functions available in Simulink.

Another reason the Arduino is the chosen microcontroller is its popularity. It has been around since 2005, and the support for it has consistently been growing. There are many projects, tutorials, and accessories available for Arduino, thus making it easy to find examples for interfacing it with many different hardware components.

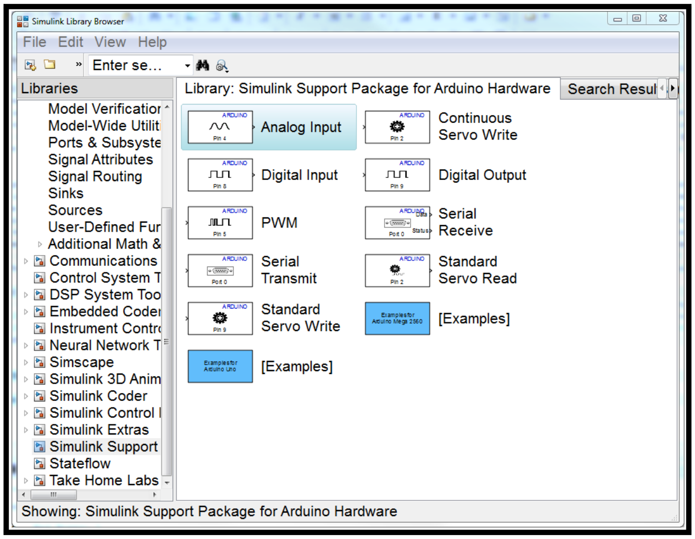


Figure 2.1: Simulink Arduino Library

Another main hardware component that is used in almost every experiment is a DC motor. There are many experiments that can be developed around it, and it can be used to help students learn many different topics within electrical and mechanical engineering. The DC motor used is small, since the the experiments are designed to be small, and will help keep the system cost down, since smaller motors are usually cheaper. However, for most practical applications, a measurement of the motor's position is needed. Encoders are good options for monitoring motor position, as they have a high resolution and can mount easily to the motor's shaft. Encoders are available in a wide variety, so it is not hard to find one suitable for most experiments. However, encoders are usually fairly expensive, and if buying them separate from the motor, they will need to be attached manually. With the motor being one of the main components, it needs to be reliable. The reliability, simplicity, and usefulness of the experiments may be affected if students have to attach encoders to motors themselves. Therefore, finding a DC motor with an encoder already attached is the best option. For the encoder, the higher the pulses per revolution (ppr) it has, the better the resolution will be for the motor's position. One of the main experiments that would have the largest load attached to the motor is the Ball and Beam. Before trying to find a motor, the Ball and Beam system was simulated for various sizes and masses.

The simulations for various sized beams provided an idea of how much current would be drawn by the motor, the voltage that would be needed, and the amount of torque needed. Having an idea of the motor characteristics that are needed helps narrow down the search for a motor, but finding an affordable DC motor with an encoder that meets the needs for every experiment is challenging. On `robotkitsworld.com`, there is a Mitsumi DC motor with a 1336 ppr quadrature encoder listed at $26.31. This is a really good price for small motor with sufficient encoder ppr. This motor encoder combination is also listed on `www.aliexpress.com` and ebay for around $6. The products ordered from the robokits website ship from India, and the motors found on aliexpress and ebay are shipped from China. The shipping rates vary, but the overall cost of the motor still ends up being the best option in comparison to anything else.

For each experiment that uses the motor, a motor driver and an external power supply need to be used as well. The external power supply is used because the Arduino can not supply the amount of current that the DC motor needs. The Arduino itself should be powered separately from the DC motor to help keep any voltage spikes or motor glitches from possibly interfering with the power to the processor. The motor driver controls the speed and direction of the motor. Motor drivers are common and easy to obtain. The motor driver has to be able to handle the amount of current that will be drawn by the motor during experiments, and has to be able to operate at the voltage of the external power supply. The DC motor can operate up to 34 volts, and has an armature resistance of 13.5 ohms. A 12 volt supply that can supply up to 3 amps of current is chosen as the external power for the motor. Since the motor's resistance is 13.5 ohms, at the maximum voltage, 888.9 milliamps will be drawn by the motor. The 3 amp maximum of the voltage supply is more than enough. The motor driver also needs to be rated for at least 888.0 milliamps and 12 volts.

There are numerous motor driver options that would meet these requirements. Some motor drivers come just as the chip itself, and may need to be soldered to a printed circuit board (PCB), or used with a breadboard. These separate motor driver chips are usually cheap, but have a suggested application circuit that would need to be constructed to use the driver properly. Other motor drivers may already come on a PCB with the required application circuit elements. Using a driver that is completely constructed or mostly constructed would require less work for the students. However, these motor drivers are more expensive than the motor driver by itself. After experimenting with multiple motor drivers, the DFRobot Arduino Compatible Motor Shield (2A) is chosen because it is only $16, does not require much alteration by the students, and fits directly on the Arduino Mega 2560 and Arduino Uno. Choosing a motor driver that fits directly on the Arduino helps keep the overall system design clean and simple. Choosing a motor driver that does not fit directly on the Arduino is harder to assemble and keep neat since multiple wires have to be used to connect the two. The DFRobot motor driver can drive up to two motors with maximum current of 2 amps. The external power supply has a 2.1mm barrel jack connector, so in order to connect the power supply to the motor driver, a 2.1mm female barrel jack to a terminal is used. Two wires are added into the terminal output of the female barrel jack adapter and input into the motor driver. The additional hardware that is used in every experiment is a USB A to B cable for connecting the Arduino to the PC, and wires for connecting the motor to the motor driver.

With the hardware introduced so far, introductory experiments for setting up the Arduino and interfacing with the DC Motor can be performed. In order to expand the experiments further, additional hardware needs to be purchased, or built. Specifically, some type of load needs to be added to the motor's shaft, and the motor needs to be mounted. Making these type of parts helps keep the cost down, and allows for various design capabilities. Therefore, the remaining parts are 3-D printed, or purchased.

2.2.2 3-D Printer

The ability to design and make your own parts has greatly been improved by the increased popularity and support for 3-D printing. 3-D printing is not new by any means, but previously 3-D printing was very time consuming and expensive. This is no longer the case, as there are numerous 3-D printers now available for less than $1000, and there is a wide variety of affordable printing materials that can be used. With the popularity increasing, more companies, people, and universities are exploring the capabilities of 3-D printing, thus creating multiple options for 3-D printing parts. At OSU, 3-D printers have been available for use by all CEAT students since Fall 2014. The 3-D printers available are the Solidoodle 4, and as long as you complete the training, you can use any of the 3-D printers for free during lab hours. The print size for parts with the Solidoodle 4 are 8"x8"x8". With 3-D printers being so popular, and decreasing in price, it is likely that other universities have student options for 3-D printing. There may not be multiple 3-D printers open for students to use, but there may be a department or lab in which they can get the parts printed for them for free, or at a low cost. There are also 3-D printing services available online that will print the parts and mail them to you. Expanding and creating multiple experiments is made easier using the 3-D printers. To integrate the main hardware, a 3-D printed base platform is designed so that the Arduino, the motor shield, the DC motor, and the other smaller components can be assembled together. By printing the base platform for the main hardware, and connecting all of the main hardware together, the students do not need to change the core design when working on different experiments. The base platform and main hardware components are shown in Figure 2.2 and Figure 2.3. Various loads and motor attachments can be designed for specific experiments, and attached to the motor. This saves time when moving from experiment to experiment, by requiring only the additional hardware for a given experiment be printed, as opposed to needing to print and connect the entire hardware system each time.

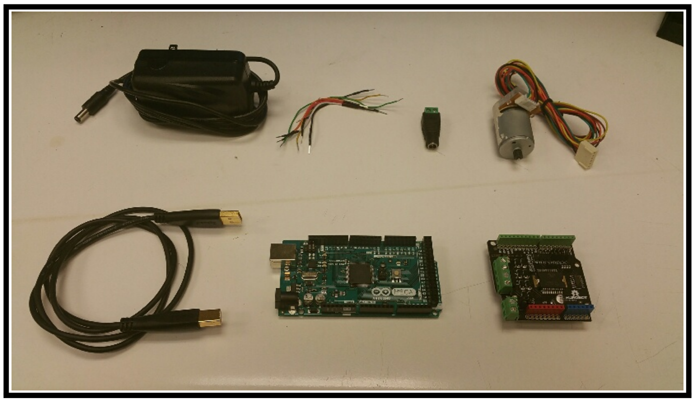


Figure 2.2: Main Hardware (1)

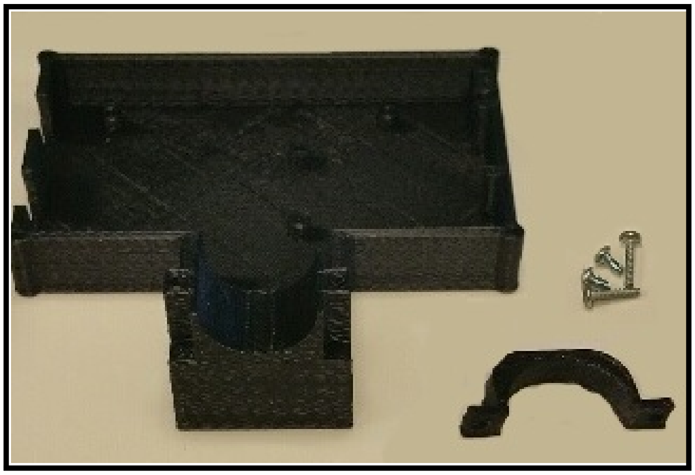


Figure 2.3: Main Hardware (2)

2.2.3 Experiment Specific Hardware

Adjustable Motor Load

To make the experiments more practical for use in a course, easily adding variation to the systems can help make each student's experiment unique without changing the main concept. Four of the five experiments involve the DC motor, and thus require a load attached to the motor's shaft. The load for the *Sampling and Data Acquistion*, *Open Loop Frequency Response*, and the *Closed Loop Frequency Response* experiments can be easily be made adjustable by using the 3-D printer. The loads shown in Figure 2.4 are two loads used in the experiments. The larger load on the left of the figure was created by another student, Sean Hendrix. The smaller load on the right is a modified version of his load, that I created. The load has circular slots along each arm so that up to four pennies can be added in each slot to vary the mass of the load. In the center of the load is a 3-D printed insert that can securely attach to the gear on the motor's shaft. The insert needs to be printed separately because it has to be printed at a higher resolution (0.1mm) in order for the small gear on the motor to fit inside of it. Printing at higher resolutions takes much more time, so only the small gear insert uses this resolution. Within a course, various combinations of pennies can be assigned to different students so that the experimental results are different from one another, ensuring each student does his/her own work. In addition to the three experiments for the loads shown in Figure 2.4, two other experiments, *Open Loop Step Response* and *Closed Loop Step Response*, designed by Sean Hendrix can use the loads as well. The ability to create multiple experiments around these loads alone shows the potential of 3-D printing for designing real experiments. Further proof of the 3-D printers usefulness can be shown by designing more advanced systems.

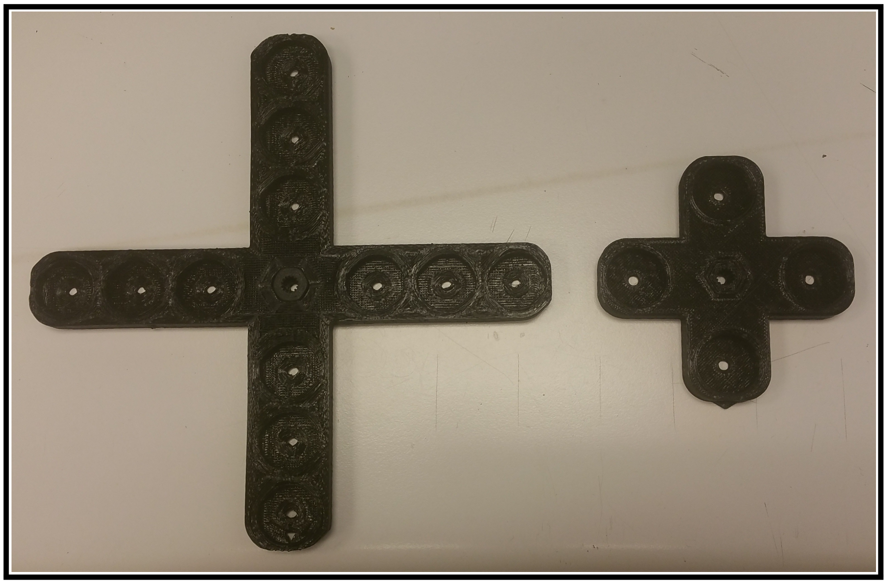


Figure 2.4: Adjustable Load Comparison

Ball and Beam

The Ball and Beam is a more advanced system design that can be used to teach multiple control theories for undergraduate and graduate courses. The Ball and Beam is a nonlinear unstable system with two degrees of freedom, the ball moving along the beam and the beam rotation. There are two potential models that can be used for the Ball and Beam; controlling the beam angle at one the end of the beam, or controlling the beam angle at the center of mass of the beam. The model for controlling the beam angle at the beam's center of mass is simpler than the model for controlling the beam angle at the end of the beam. Also, the design of the Ball and Beam with the beam angle controlled at the beam's center of mass is simpler. Therefore, this is the best model for making the Ball and Beam system. Keeping the design simple will ensure that the 3-D printer can print it at a high quality, and it will be easier for students to replicate and assemble. A theoretical model of the Ball and Beam system with the beam angle controlled at the beam's center of mass is shown in Figure 2.5.

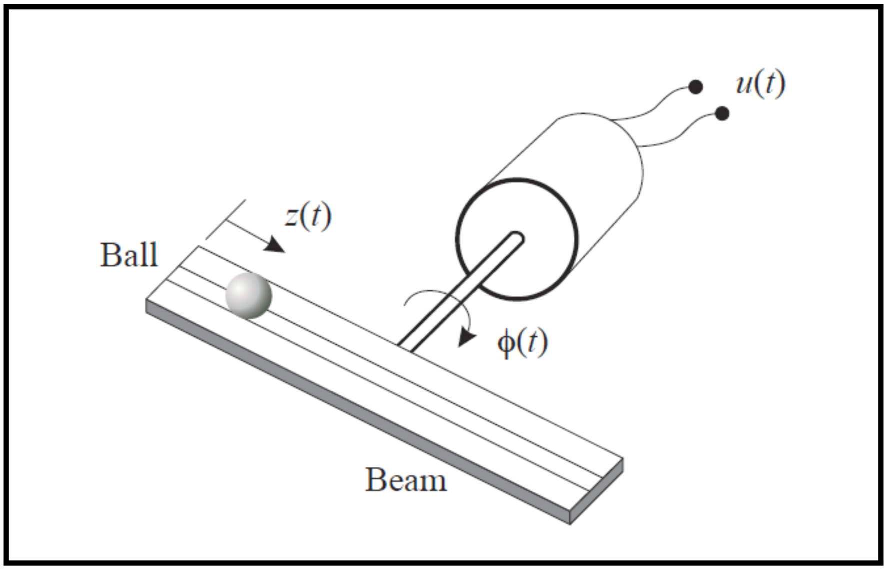


Figure 2.5: Ball and Beam Model

The equations of motion of the system are provided below [14].

$$\dot{x_1} = \frac{\tau - mgcos(x_2)x_4 - 2mx_1x_3x_4}{J_{beam} + J_{ball} + mx_4^2} \tag{2.1}$$

$$\tau = \frac{K_t}{R_a}(u - K_b x_1) \tag{2.2}$$

$$\dot{x_2} = x_1 \tag{2.3}$$

$$\dot{x_3} = \frac{mx_1^2x_4 - mgsin(x_2)}{\frac{J_{ball}}{r^2} + m} \tag{2.4}$$

$$\dot{x_4} = x_3 \tag{2.5}$$

where

$$x_1 = \text{Beam Velocity}$$
$$x_2 = \text{Beam Position}(\phi)$$
$$x_3 = \text{Ball Velocity}$$
$$x_4 = \text{Ball Position}(z)$$
$$u = \text{Motor Voltage}$$
$$\tau = \text{Torque}$$

The constants are provided in the *Optimal State Feedback Control (Ball and Beam)* experiment.

For the actual Ball and Beam system, there needs to be a way to measure the beam position and ball position. Therefore, sensors are used to measure the positions, which affect the actual design of the beam. The DC motor needs to be attached to the beam, since the encoder on the motor measures the beam angle. There are multiple possibilities for measuring the ball position; conductive plastic, resistive wire, an ultrasonic range finder, or a vision sensor [15]. Conductive plastic is not easily found, and does not seem like a reliable option. The resistive wire requires a metal ball be used, thus potentially making the ball and beam too heavy for the small motor. In addition to the mass, the resistive wire requires a good deal of maintenance and attention to detail to assemble. The vision sensor is too expensive for this application so it is not a good choice.

Ultrasonic sensors are cheap, but are sometimes noisy. The HC-SR04 costs less than $3. The low cost makes it the best choice of the sensors mentioned thus far. The HC-SR04 ultrasonic sensor has a range of 2 cm to 400 cm, with an accuracy of 3 mm. Testing the experiment using the ultrasonic on one end of the beam, along with a ping pong ball, the noise of the ultrasonic proved to be a big issue. The Ball and Beam was able to balance, proving the controller worked, but the ultrasonic sensor would experience random large jumps while the ball was not moving at all. The large jumps caused the controller to think the ball position and velocity changed, which made the motor apply a voltage to attempt to correct the ball position. To determine if the source of the problem was the ping pong ball, multiple balls were used. After using a foam golf ball, a rubber ball, and a soft mini baseball, the ping pong worked the best. The ultrasonic waves do not reflect well for certain materials, and the reason for the large jumps with the ping pong ball were its size and shape. The ultrasonic wave did not reflect well because of the edges of the ball, which resulted in frequently receiving incorrect readings. The ultrasonic works best with large objects, or with a large flat surface in which the ultrasonic wave can easily be reflected. Smoothing the output of the ultrasonic helped, but not enough.

In order to make sure the experiment is reliable, a different ball position sensor is used. The best alterative to the ultrasonic sensor is the Sharp GP2Y0A41SK0F IR (infrared) sensor. The function of the Sharp IR sensor is similar to the ultrasonic sensor, but is more expensive. The Sharp IR sensor is an analog distance sensor with a detection range of 4 cm to 30 cm (1.5” to 12”), and costs around $10. The sensor outputs an analog output voltage for corresponding distances, but this relationship is nonlinear. To read the sensor correctly, a look-up table block is used in Simulink to generate the nonlinear curve that matches the sensor’s output voltage versus distance curve. The analog voltages corresponding to 3 cm and 40 cm are 3.025 volts and 0.3 volts. The description of the sensor says the lowest distance the sensor can read is 4 cm, but from the datasheet, it shows it can actually read as low as 3 cm.

To read the analog output voltage from the IR sensor, an analog input pin on the Arduino is used. The Arduino’s analog input represents the input voltage as a digital value ranging from 0-1023. The maximum voltage is determined by the analog input reference voltage, which by default is 5 volts. This means that if the input to the analog input pin is 5 volts, the output of the block will be 1023, and if the input to the pin is 0 volts, the output of the block will be 0. The maximum reference voltage can be changed from 5 volts to 1.1 V, 2.56 V, or an external voltage in Simulink’s configuration settings. To set the maximum reference voltage externally, the voltage desired has to be input into the Arduino’s analog reference pin. Since the maximum output voltage of the IR sensor is 3.025 volts, the analog input reference voltage is changed to 3.3 volts to provide a better resolution. The resolution of the analog pin will be 3.2 millivolts, which makes the resolution of the IR sensor 0.32 mm per count. This is a much better resolution than the 3 mm resolution for the ultrasonic sensor. Another benefit of choosing the Sharp IR sensor as the ultrasonic sensors replacement is that the beam design does not have to be altered much since the IR sensor functions similarly to the ultrasonic sensor.

There are multiple possibilities for the actual beam that is 3-D printed. The initial design was similar to the theoretical picture. The center of the beam was lower than the outer edges. The only difference was the ends of the beam were a little higher so that the ball would not fall off of the end. The beam was 12 inches, and since the 3-D printer could not print the beam this large, it had to be broken up into two parts. The problem with this design is the ball may fall off the sides, since the ball just rolls along the top of the beam. Also, the need to connect two beams together may prevent the beam’s surface from being level. Other designs were created, and ultimately the beam design shown in Figure 2.6 was created in Solidworks. This final design uses the Sharp IR sensor, and the design is very similar to the design used with the ultrasonic sensor. One end of the beam is elevated substantially, and the inside of the beam is cut out in the shape of a ball, helping prevent the ball from falling off or out of the beam. The other end of the beam is not elevated as high, because that is the end the IR sensor is mounted to. The sensor itself is mounted on a separate smaller attachment that offsets it from the end of the beam. The reason for the offset is that when the ball rolls to the end of the beam closest to the sensor, it is still at a distance greater than the smallest distance the IR sensor can read. Adding the IR sensor on the end of the beam shifts the beam's center of mass. Depending on the 3-D printer used, and the settings of the printer, the 3-D printed parts that the students make may not be exactly the same as those used to test the experiments. Therefore, in the general area of the beam's center of mass, the beam extends outward on both sides and has multiple holes for connecting the motor and beam attachment. The multiple holes allow the center of mass to be adjusted easily without needing to reprint the beam. The motor and beam attachment is divided into three smaller parts; the top attachment that connects to the beam, the bottom attachment that connects into the top attachment, and the gear insert that is inserted in the the bottom attachment. The reason for three separate parts is the limitations of the available 3-D printers. The final Ball and Beam system is shown in Figure 2.7.

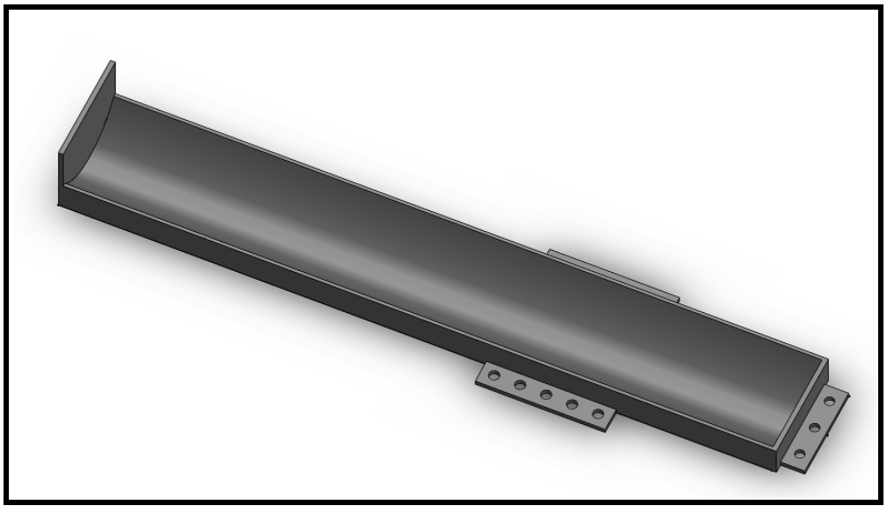


Figure 2.6: Beam Model - Solidworks

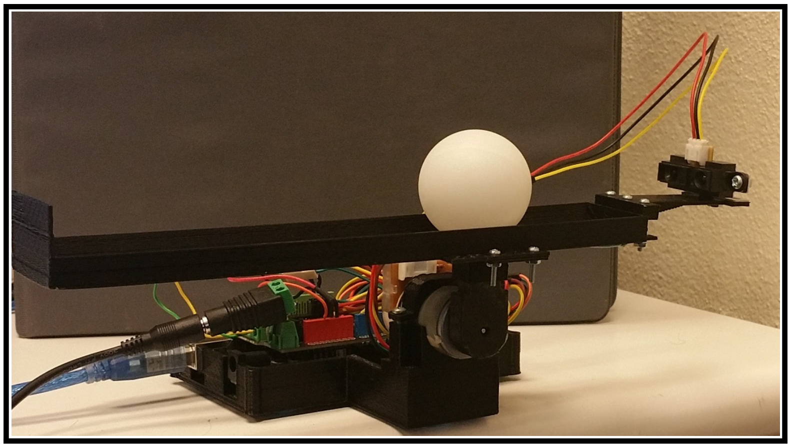


Figure 2.7: Ball and Beam Final Design

System Cost

Most of the hardware used in the first four experiments can be used in the Ball and Beam experiment as well. However, in addition to some of those components, others need to be purchased. The entire system cost is shown in 2.1. The total costs are assuming MATLAB/Simulink and 3-D printing are available for free. There are two final costs, \$119.58 and \$139.27, and they are the estimated worse cases. The two different costs are dependent on where the motor is purchased. Both costs meet the goal of designing the system to be less than \$150. Also, some of the hardware components listed may not need to be purchased, as students may have them already, or have access to them. This would bring the cost of the overall system down substantially. The hardware needed for the first four experiments would not include all of the parts listed in the table, resulting in a much lower price as well. Another factor for the cost is that all of the hardware would not necessarily need to be purchased all at once. As the experiments progress, new components are needed, but some of the main components will have been purchased already. For each experiment, the required hardware will be listed in the required materials section of the handout they will follow.

Total System Cost		
Item	**Cost**	**Location**
Arduino Mega 2560 Rev3	$37.39	Robotshop.com
Mitsumi DC Motor with Encoder	$6.62 or $26.31	Ebay or Robokitsworld.com
DFRobot Arduino Compatible Shield (2A)	$16.00	Robotshop.com
12VDC 3A Wall Adapter Power Supply	$9.95	Robotshop.com
Sharp GP2Y0A41SK0F IR Range Sensor - 4 to 30cm	$9.95	Robotshop.com
SIRC-01 Sharp GP2 IR Sensor Cable - 8”	$1.95	Robotshop.com
Bracket Pair for Sharp IR - Parallel	$3.49	Pololu.com
Ping Pong Ball	$1.56	Walmart
USB Cable	$1.99	Robotshop.com
5V Cell Phone USB Wall Adapter	$6.95	Robotshop.com
Female Barrel Jack	$2.25	Robotshop.com
Wires	$4.98	Robotshop.com
Screwdriver Set	$5.98	Amazon.com
Sticky Tack	$3.29	Walmart
2-56 Screws (1/4”)	$0.99	Pololu.com
2-56 Screws (5/16”)	$0.99	Pololu.com
2-56 Screws (7/16”)	$0.99	Pololu.com
4-40 Screws (1/4”)	$0.99	Pololu.com
4-40 Screws (1/2”)	$0.99	Pololu.com
2-56 Hex Nut	$0.99	Pololu.com
2-56 Aluminum Standoffs (1/4”)	$1.29	Pololu.com
Total Cost	**$119.58 or $139.27**	

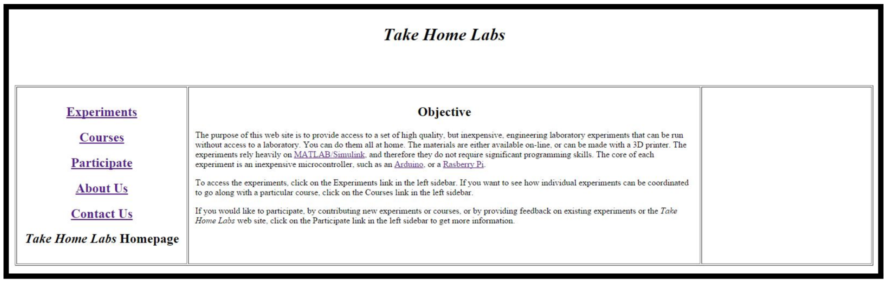
Table 2.1: Estimated System Cost

2.3 Student Handouts

The experiments will be distributed via student handouts. The student handout is essentially a lab manual that the student will follow for each experiment. The handouts are designed to be simple and easy to follow, since there will likely not be a TA to help them. Each handout will begin with an objective section in which the overall idea and framework of the experiment is discussed. The setup section is next, in which the required hardware and software materials will be listed, any prerequisite experiments will be provided, and the actual hardware and software setup steps will be listed for the student to follow. The experimental procedures section follows the setup section. The student will follow the all of the steps for completing the lab. Throughout the experimental steps, the student will be asked questions about important topics. The questions are intended to help keep the student thinking, and to relate the experimental portions to the theoretical topics of the experiment. The final section of the student handout is the conclusion section which reiterates the main topics and things learned in the lab. This overall structure is used for every experiment, which helps keep things consistent and organized. To make the student handouts easily accessible to anyone who wishes to try them, they are provided on our Take Home Labs (THL) website `www.thl.okstate.edu`.

2.4 Take Home Labs Website

The THL website is created to organize and distribute the experiments. With a website, anyone can access the experiments at any time. The homepage of the website gives an introduction about the overall idea of the Take Home Labs. When someone is viewing the homepage, in the left most column will be sections of the website they can navigate to easily. The different sections are *Experiments*, *Courses*, *Participate*, *About Us*, *Contact Us*, and *Take Home Labs*. These sections will appear on the left of any page, providing easy access back and forth between the main sections of the website.



Objective

The purpose of this web site is to provide access to a set of high quality, but inexpensive, engineering laboratory experiments that can be run without access to a laboratory. You can do them all at home. The materials are either available on-line, or can be made with a 3D printer. The experiments rely heavily on MATLAB/Simulink, and therefore they do not require significant programming skills. The core of each experiment is an inexpensive microcontroller, such as an Arduino, or a Rasberry Pi.

To access the experiments, click on the Experiments link in the left sidebar. If you want to see how individual experiments can be coordinated to go along with a particular course, click on the Courses link in the left sidebar.

If you would like to participate, by contributing new experiments or courses, or by providing feedback on existing experiments or the *Take Home Labs* web site, click on the Participate link in the left sidebar to get more information.

Figure 2.8: Take Home Labs Homepage

2.4.1 Experiments

The Experiments page shows a list of categories for each experiment. Pressing one of the categories jumps to a new page with a list of the experiments that are related to the category. This is useful when someone is looking for a particular experiment based on an idea they would like to learn. Pressing any of the experiment links will open the main page for the actual experiment. In the middle of the actual experiment page, there will be an introduction or objective about the experiment. The right section of the page will list important links for the experiment. The links are *Handout*, *Hardware*, *Software*. The handout link will open or download the PDF of the student handout. The Hardware link will open a new page that lists the required hardware for the experiment. The listed hardware components are links that will navigate to pages where parts can be purchased. It is mentioned on the Hardware page that the links are just for reference, the components can be purchased anywhere. The Software link will open a new page that lists all of the experiments that have software files. Pressing each link will download the required software and 3-D printer files for that particular experiment.

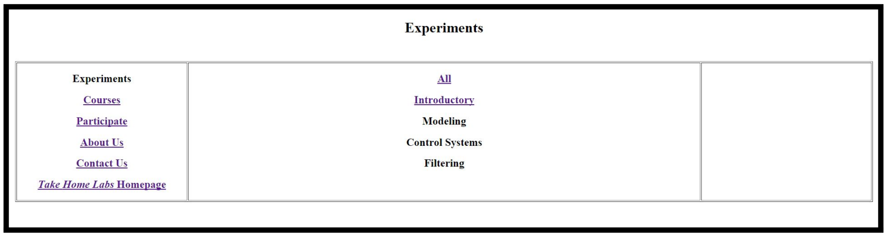
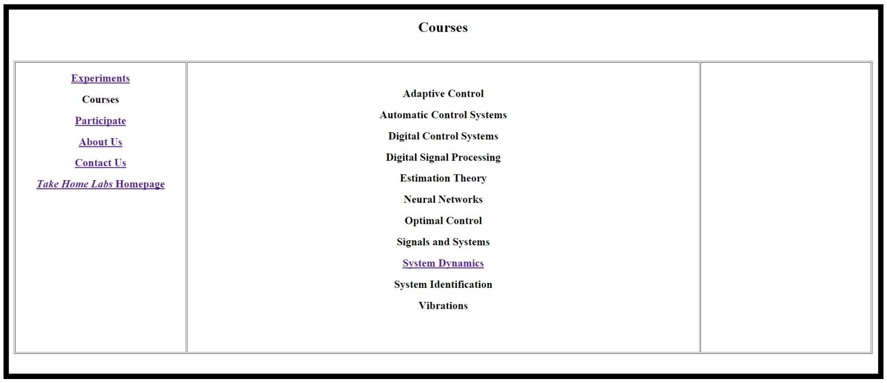


Figure 2.9: Take Home Labs Experiments Page

2.4.2 Courses

On the Courses page, a list of courses that the experiments pertain to are shown. Pressing one of the courses opens a new page with a list of the experiments for that particular course. This is another method for finding experiments, but based on specific courses. Pressing any of the experiment links will open the main page of the actual experiment the same way it does in the Experiments subsection.

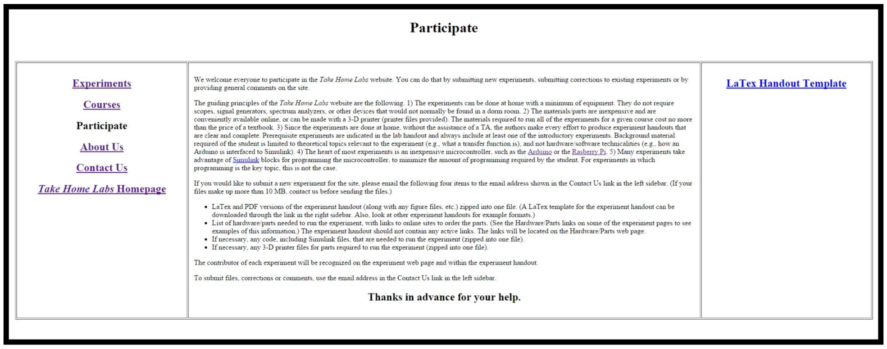


Adaptive Control
Automatic Control Systems
Digital Control Systems
Digital Signal Processing
Estimation Theory
Neural Networks
Optimal Control
Signals and Systems
System Dynamics
System Identification
Vibrations

Figure 2.10: Take Home Labs Courses Page

2.4.3 Participate

On the Participate page, information about the overall idea of the take home labs is provided for newcomers to the site. The structure for the student handouts is listed, with examples for anyone who wishes to add experiments to the site. This helps promote other enthusiasts to participate, and gives them the same guideline and structure the current experiments follow. Therefore, any new experiments that are added will maintain the desired simplicity and low cost. In addition to submitting new experiments, people can provide general comments about the existing experiments. Feedback is helpful for keeping experiments and ideas up to date.



Participate

We welcome everyone to participate in the *Take Home Labs* website. You can do that by submitting new experiments, submitting corrections to existing experiments or by providing general comments on the site.

The guiding principles of the *Take Home Labs* 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 Arduino or the Rasberry Pi. 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.

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.)

- 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.)
- 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.
- If necessary, any code, including Simulink files, that are needed to run the experiment (zipped into one file).
- If necessary, any 3-D printer files for parts required to run the experiment (zipped into one file).

The contributor of each experiment will be recognized on the experiment web page and within the experiment handout.

To submit files, corrections or comments, use the email address in the Contact Us link in the left sidebar.

Thanks in advance for your help.

Figure 2.11: Take Home Labs Participate Page

2.4.4 About Us/Contact Us

The About Us page just lists information about the people who have contributed to the Take Home Labs website. The Contact Us page lists contact information that can be used to submit any questions or concerns. This is the primary access point people will use when they wish to participate.

2.5 Additions and Alternatives

There are other software and hardware that can be used for the experiments. Additions and alternatives help keep the experiments flexible and open, making them more attractive to different people. Some people may be more accustomed to other software and hardware than those used currently. One popular addition for a software program would be Labview. Some students may be more accustomed to Labview, and some universities may have access to Labview instead of MATLAB. Additional microcontrollers that could be added to work with the experiments are the Raspberry Pi and Beagleboard. Both have support in Simulink. Like the Arduino, the Raspberry Pi has numerous custom blocks in Simulink for using the board's functionality. The Raspberry Pi is more powerful than the Arduino, but is not as easy to use and set up. There are multiple motor drivers that could be used, as long as the motor driver is rated for 12 volts and can handle at least 2 amps. The same thing goes for the power supply. Any 12 volt power supply can be used if it can supply at least 2 amps. Other voltage power supplies will work as well, but some of the experiment models will need to be adjusted. An alternative motor is more difficult to than other hardware components since the current motor is small and available for a low price. Choosing a different motor would also change some of the 3-D printed hardware designs. However, the experiments will work for different motor encoder combinations if desired, but the experiments would have to be adjusted. Other 3-D printed designs can be created, and as 3-D printers continue to progress, better printers will be available. The better printers will likely not impose as many design constraints as the current 3-D printers. This is where having the website will be very beneficial, as it will allow others to participate with adding additional elements to the overall system framework. The overall purpose of the the Take Home Labs is expressed further by the following chapters. The chapters are example experiments that are available on the website for students to download and attempt. The first experiment is a simple introductory experiment. The next three experiments are example experiments that are intended to be used in an undergraduate System Dynamics course, and the final experiment is an advanced experiment intended for students in an Optimal Control course.

CHAPTER 3

BALL AND BEAM SYSTEM

Third chapter text goes here. Table 3.1 is another sample table.

Table 3.1: Sample multicolumn table

Parameter	Column 1		Column 2	
	Subcolumn 1	Subcolumn 2	Subcolumn 1	Subcolumn 2
Parameter 1	Element 11	Element 12	Element 13	Element 14
Parameter 2	Element 21	Element 22	Element 23	Element 24
Parameter 3	Element 31	Element 32	Element 33	Element 34

CHAPTER 4

BLINKING LED

4.1 Objective

Experiment one is an introductory lab designed to help the students setup and become familiar with the hardware and software that is required for each of the controls labs. In order to achieve this, the student will follow the steps provided to install support for the Arduino in Simulink, and install the Arduino drivers on their computer. Successfully installing the support needed for the hardware is crucial, thus the student will need to verify that everything is installed correctly by using the hardware and software to successfully blink an LED using two different methods for running models on the Arduino with Simulink: Normal Mode and External Mode.

4.2 Setup

Before each experiment is run, there will be necessary hardware and software setup steps to follow. These steps will allow the student to properly construct and interface any of the hardware components that will be used in that particular lab. The necessary software models will be constructed as well. The students should be more focused on understanding the actual experimental steps, not the setup portion. Therefore the setup section will be as detailed as needed to reduce the probability of the student making a mistake. Before the setup steps are given, the relevant hardware and software materials will be listed. These things could include, the Arduino, a motor driver, DC motor, 3-D printed parts, 3-D printed files, Simulink files, etc. Since this is an introductory lab, the hardware and software setup portion of this lab will be apart of the actual experimental steps. Therefore, this section will only contain the hardware and software materials that are needed.

4.2.1 Required Materials

Hardware

- Arduino Mega 2560 R3

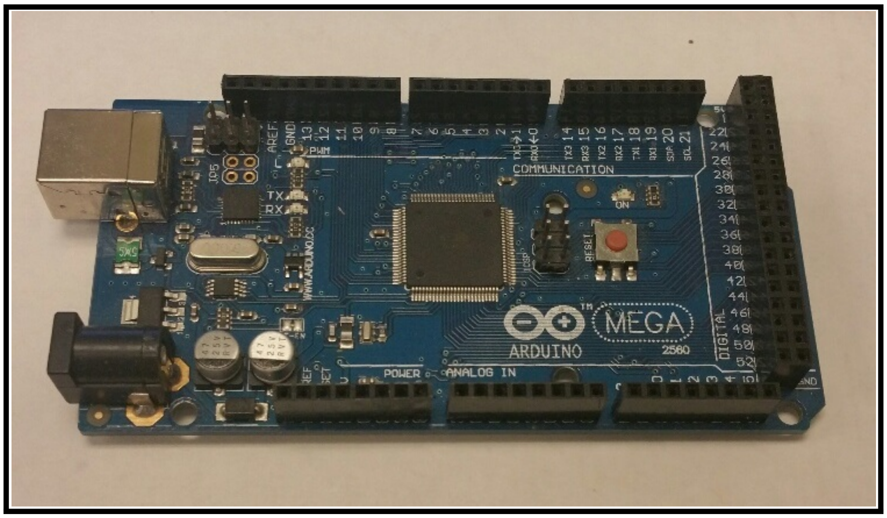


Figure 4.1: Arduino Mega 2560

- USB cable (Standard A to B plug)

Figure 4.2: USB Cable

Software

- MATLAB/Simulink R2013a or later

  ⋆ *The steps and images related to MATLAB/Simulink for this experiment were created using MATLAB/Simulink 2013a. Therefore some steps and images may be a little different if you are not using this version.*

- Personal Computer with administrator access

Prerequisite Experiments

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

4.3 Experimental Procedures

The main concept of each lab will be contained in this section. These will be the steps the student follows to further learn about different topics within control systems. In this thesis, all of the relevant details and ideas will be given, but the lab handout the student receives will not contain all of the information. The student will have to use what they have learned to successfully complete the lab. Checkpoints and questions may be inserted throughout the steps in order to give the student a chance to stop, so that they can reflect on and verify some of the things they just completed, especially when an important topic has been discussed. Since the students will have to rely on their own knowledge and skills, there is a higher chance they may encounter difficulties unrelated to the experiment topics. Therefore, troubleshooting sections will be provided to hopefully eliminate any common issues that may be encountered.

4.3.1 Exercise 1: Installing Arduino IDE and Drivers

The first task is installing the Arduino drivers on the PC. The Arduino and the USB cable will be needed at this time. These steps are based on the installation method provided on the Arduino website. General information about getting started with the Arduino can be found at `www.arduino.cc/en/Guide/HomePage`. This is a good place to explore if you experience any difficulties with the following installation process.

1. In your web browser, navigate to `www.arduino.cc`.

2. At the top of the home page, select “Download”.

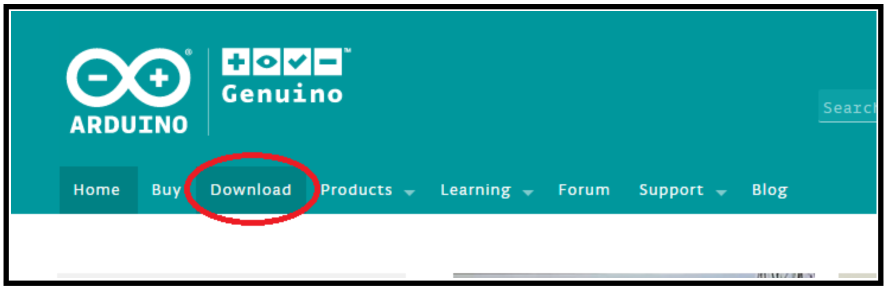


Figure 4.3: Arduino Download Page

3. On the right side of the download page, select "Windows Installer" as shown in Figure 4.4.

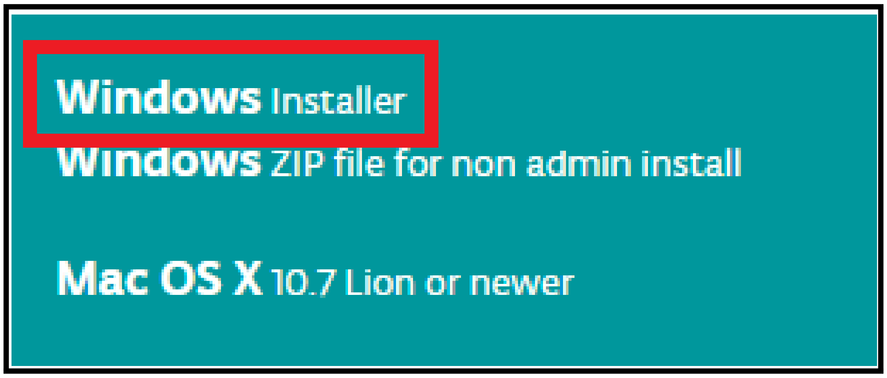

Windows Installer
Windows ZIP file for non admin install
Mac OS X 10.7 Lion or newer

Figure 4.4: Windows Installer

4. A page asking you to contribute should show, select "Just Download", unless you would like to contribute to the Arduino software.

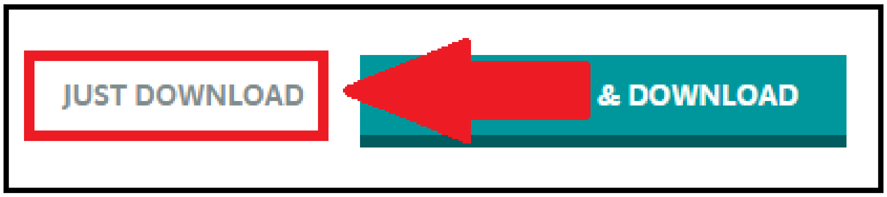


Figure 4.5: Contribute Page

5. An executable file will start to download. Once the download finishes, select the file and install the Arduino IDE. You will need administrator access on your PC to install the IDE.

6. When the install finishes, make sure the Arduino is not sitting on a metal or conductive surface, as it may cause the pins on the bottom of the board to short

together. Connect the USB cable from the Arduino to the computer. The green LED on the Arduino should turn on, indicating the board is being powered.

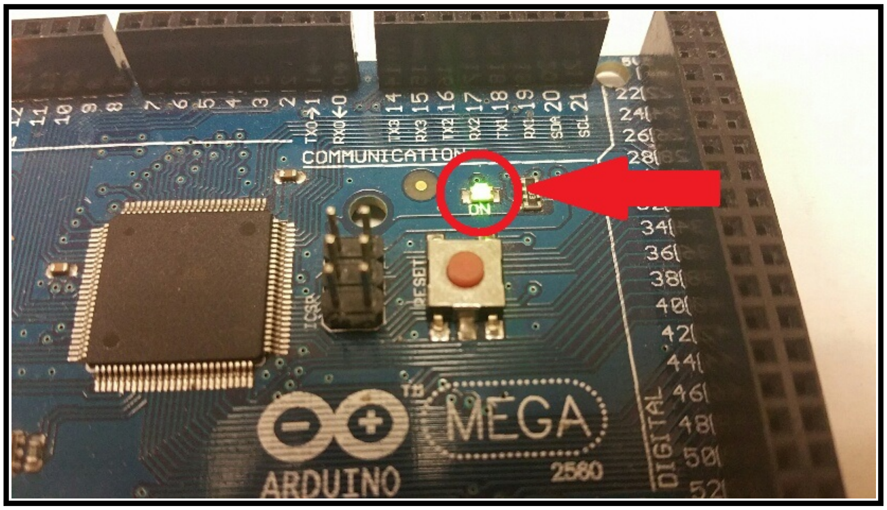
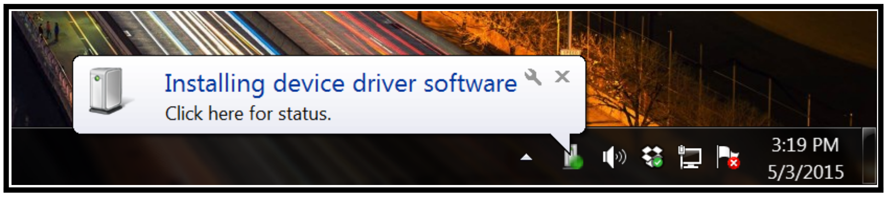


Figure 4.6: Arduino Power ON LED

7. Wait for the computer to search for the drivers online.



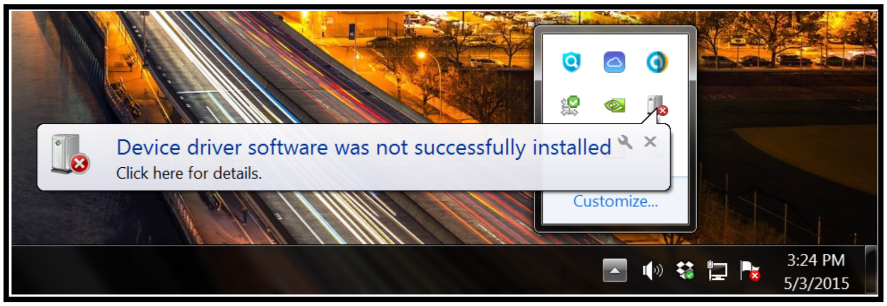
Installing device driver software

Click here for status.

3:19 PM
5/3/2015

Figure 4.7: Driver Search

8. If the computer fails to locate the drivers automatically, as shown in Figure 4.8, follow the steps below in red, otherwise skip ahead to step 9.



Device driver software was not successfully installed

Click here for details.

3:24 PM
5/3/2015

Figure 4.8: Failed install

(a) To manually install the Arduino drivers onto the computer, navigate to Device Manager, and look for the Arduino Mega board. The board should be under “Ports(COM & LPT)”, or “Other Devices”.

- If the Arduino is under “Other Devices”, it may be listed as “Unknown Device”. To verify that this is the Arduino, unplug the USB from the Arduino and see that the device is removed from the list. Now reconnect the USB to the Arduino.

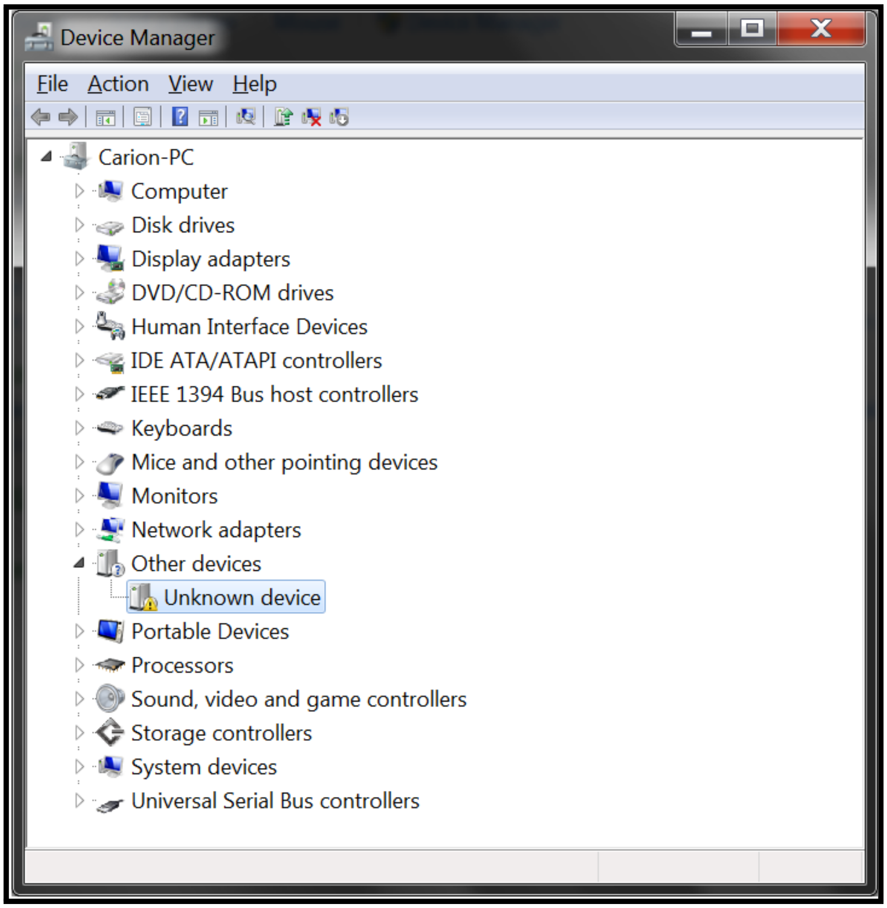
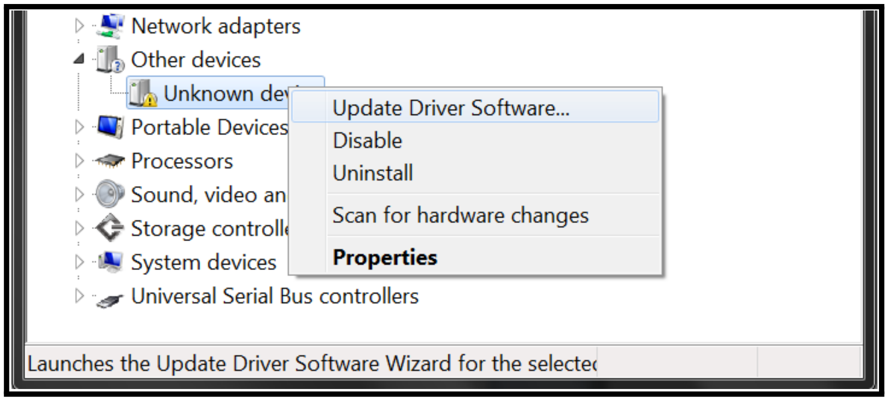


Figure 4.9: Unknown Device

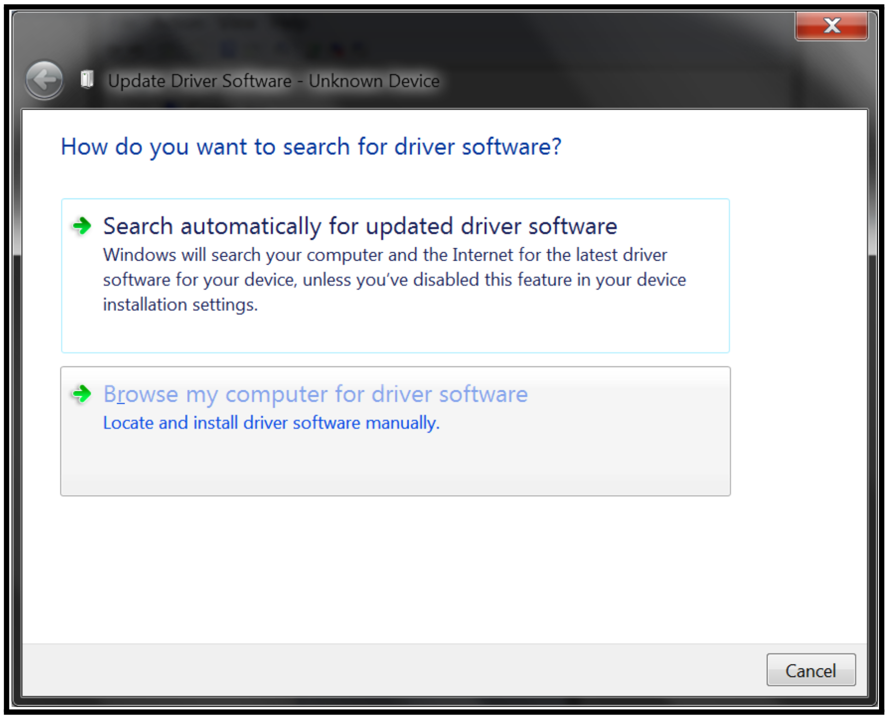
(b) Right-click on the Arduino Mega or the “Unknown Device”, and select “Update Driver Software...”.



Launches the Update Driver Software Wizard for the selectec

Figure 4.10: Driver Update

(c) Select “Browse my computer for driver software”


Update Driver Software - Unknown Device

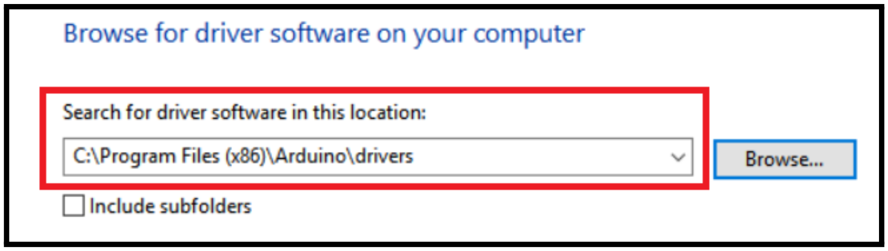
How do you want to search for driver software?

Search automatically for updated driver software
Windows will search your computer and the Internet for the latest driver software for your device, unless you've disabled this feature in your device installation settings.

Browse my computer for driver software
Locate and install driver software manually.

Figure 4.11: Browse for Driver

(d) Set the location to the “drivers” folder that is within the folder where the Arduino software was downloaded, similarly to the location shown in Figure 4.12. Deselect the box next to Include subfolders.



Browse for driver software on your computer

Search for driver software in this location:

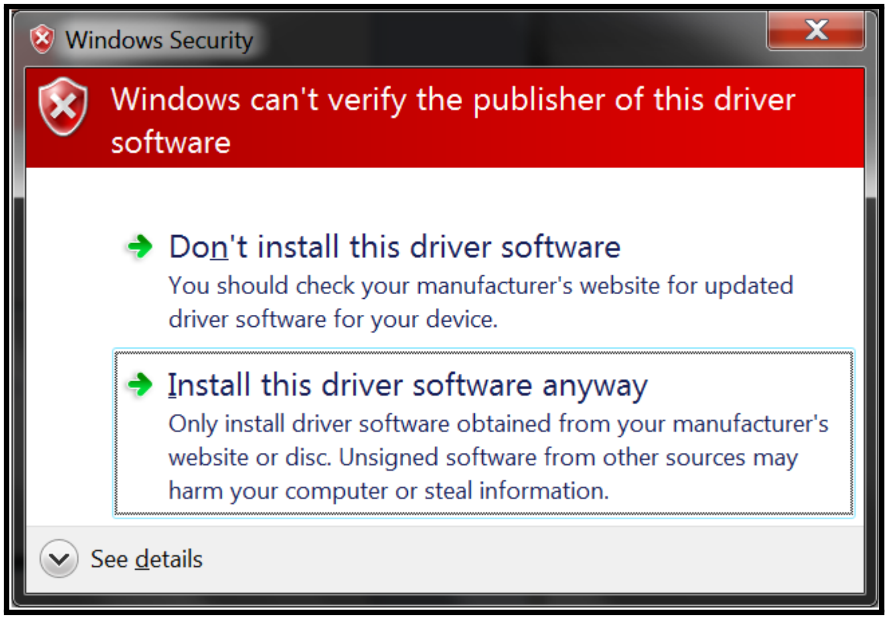
C:\Program Files (x86)\Arduino\drivers

Browse...

□ Include subfolders

Figure 4.12: Driver Path

(e) There will be a prompt stating that Windows cannot verify the publisher of the driver, select "Install this driver software anyway".



Windows Security

Windows can't verify the publisher of this driver software

Don't install this driver software

You should check your manufacturer's website for updated driver software for your device.

Install this driver software anyway

Only install driver software obtained from your manufacturer's website or disc. Unsigned software from other sources may harm your computer or steal information.

See details

Figure 4.13: Windows Security

(f) Once Windows finishes installing the drivers, a message will appear stating the driver software was updated successfully. Close this window.

9. Navigate to the Device Manager on the computer, and check that the Arduino shows under "Ports(COM & LPT)". The Arduino Mega should be listed as "Arduino Mega 2560 (COMx)" where x is the communication port number being used. Write down the communication port number as it may be useful later on.

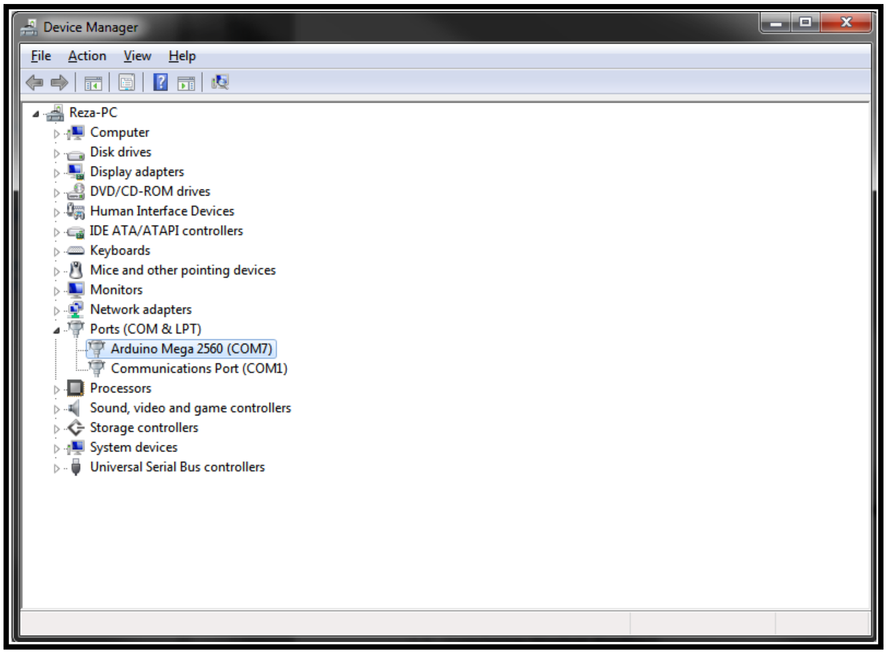


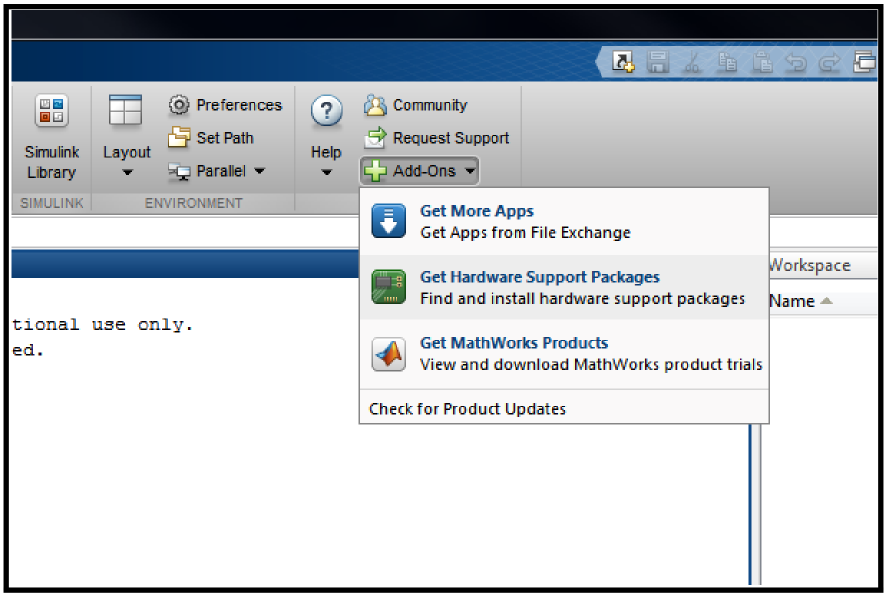
Figure 4.14: Driver Check

4.3.2 Exercise 2: Installing Arduino support in Simulink

Now that the Arduino drivers have been successfully installed on the PC, the next step is installing the Arduino support in Simulink.

10. Open MATLAB.

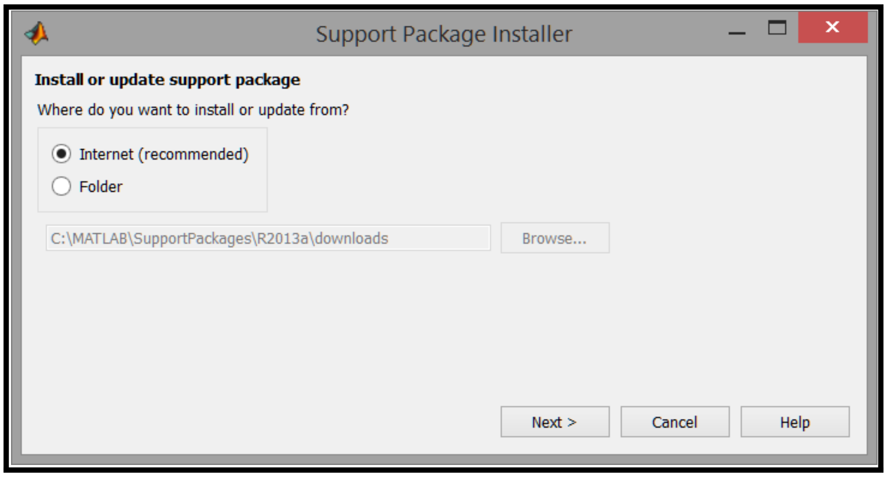
11. On the “Home” toolstrip, navigate to “Resources” and click the “Add-Ons” dropdown menu followed by “Get Hardware Support Packages” as shown in Figure 4.15.


Simulink Library
Layout
Preferences
Set Path
Parallel
SIMULINK
ENVIRONMENT
Help
Community
Request Support
Add-Ons
Get More Apps
Get Apps from File Exchange
Get Hardware Support Packages
Find and install hardware support packages
Get MathWorks Products
View and download MathWorks product trials
Check for Product Updates
Workspace
Name

```
tional use only.
ed.
```

Figure 4.15: Get Support Package

12. Install the support package using the internet and hit “Next”.



Install or update support package

Where do you want to install or update from?

● Internet (recommended)
○ Folder

C:\MATLAB\SupportPackages\R2013a\downloads

Figure 4.16: Install Method

13. A MathWorks account will be needed to proceed, so log in with an existing account or create a new one.

14. Once successfully logged in, check any boxes next to support packages related to the Arduino and hit next.

15. Follow the remainder of the on-screen prompts until the installation is finished, and exit out by clicking “Finish”.

Checkpoint:

This section allows the student to stop and verify that the installation was successful. Verification requires the student to open Simulink and check that the "Simulink Support Package for Arduino Hardware" library shows up in the Simulink Library Browser. The steps the student would follow are shown below:

16. Type *simulink* in the MATLAB Command Window.

17. The Simulink Library Browser should automatically open. Now, in the "Libraries" section, verify that the "Simulink Support Package for Arduino Hardware" is there, as shown in Figure 4.17. For Matlab/Simulink versions released after 2013a, the Arduino blocks will be grouped into categories. The blocks used throughout the labs for these later versions will be under Simulink Support Package for Arduino Hardware → Common.

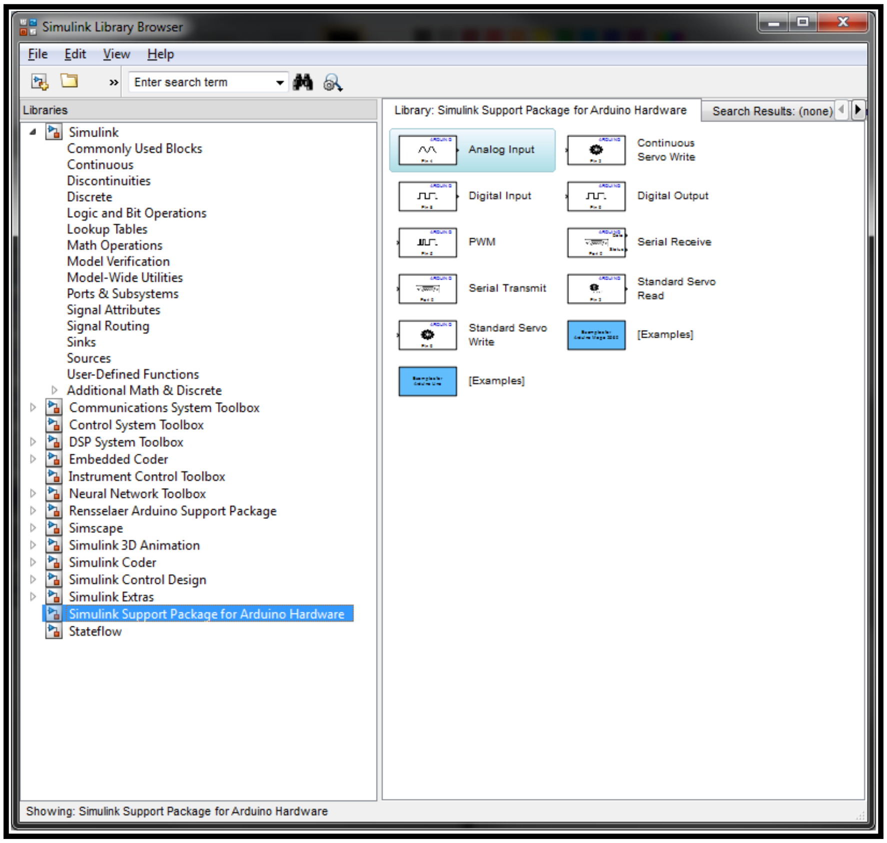


Figure 4.17: Support Package Validation

4.3.3 Exercise 3: Blinking an LED

Now that the Arduino support is installed in Simulink, and the Arduino drivers have been installed on the PC, the next step is verifying the PC can communicate with the board properly. There are two methods that will be used to demonstrate proper communication between the board and the PC: Normal Mode and External Mode. Both methods have their pros and cons, and, depending on the project, one method may prove to be more beneficial than the other. Thus, it is important to introduce both methods now so that the student is familiar with them before starting any of the later labs. A simple experiment to introduce both of these methods is blinking an LED. A more extensive introduction to Normal and External Mode can be found in experiment *Sampling and Data Acquisition*. Besides the Arduino and the USB cable, there is no additional hardware needed for this experiment, because the Arduino has a built-in LED.

Normal Mode

As indicated by its name, Normal Mode is just the normal method used to download programs onto the Arduino. Usually, with an Arduino, the programs are developed and downloaded to the board in the Arduino IDE, which is not needed for these projects, since we are using Simulink. A program downloaded to the board in Normal Mode will stay on the board until another program overwrites it. This means that even when the board is disconnected from power and reconnected later, the program is still loaded onto the board, and will just start executing again. The main disadvantage with Normal Mode is that if any data needs to be recorded, they will be more difficult to acquire than with External Mode. The main advantage with Normal Mode is that it allows programs to run at a much faster rate than External Mode. The following steps show how to use Normal Mode to blink the onboard LED of the Arduino. The final Simulink Model will look like Figure 4.18:

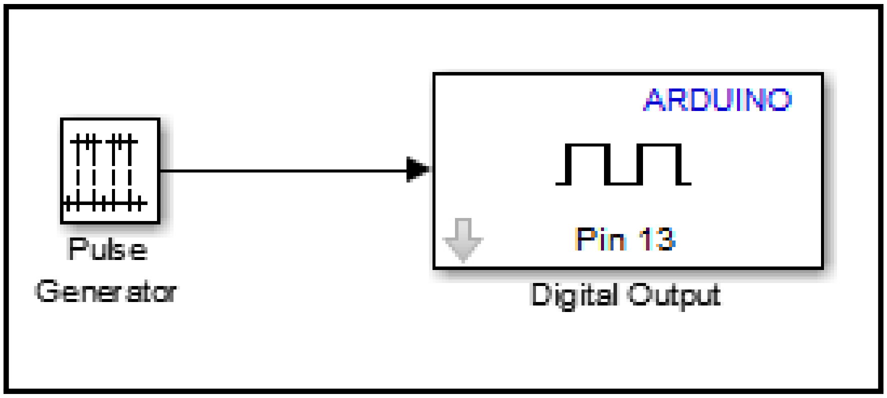


Figure 4.18: Simulink Model - Normal Mode

18. Open MATLAB, then open a new Simulink Model using either of the following methods:

    - In the MATLAB window's "Home" toolstrip, choose File → New → Simulink Model.
    - Type *simulink* in the MATLAB Command Window, then press the New Model icon.

19. Open the Library Browser by pressing the icon, or selecting View → Library Browser.

20. Under the Libraries section add a Pulse Generator block to the model:

    - Simulink → Sources → Pulse Generator

21. Under the Libraries section add an Arduino Digital Output block by selecting:

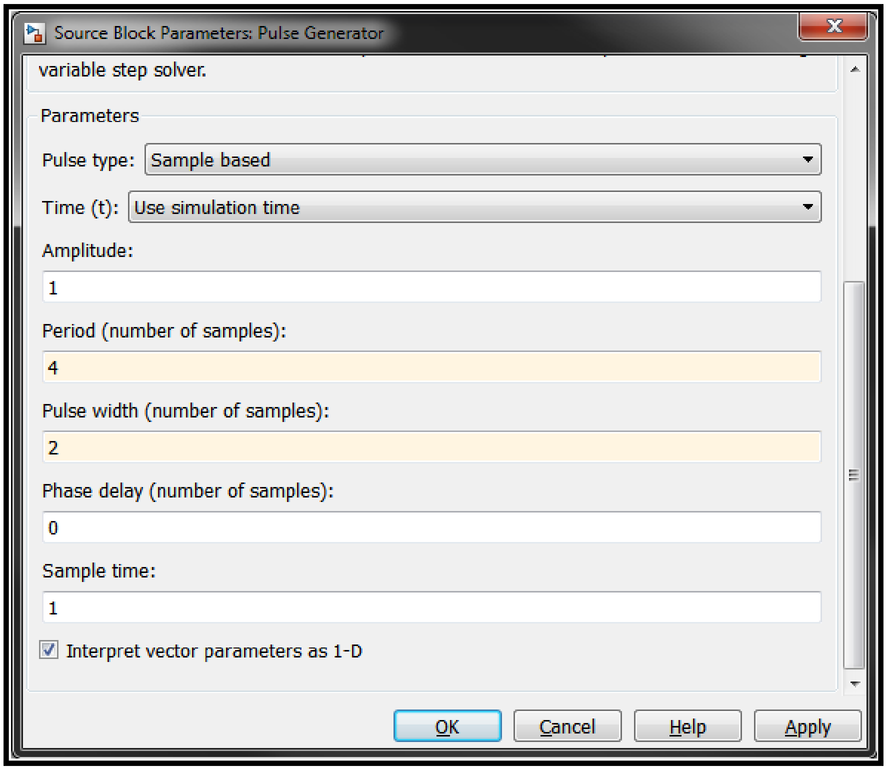
    - MATLAB R2013a: Simulink → Simulink Support Package for Arduino Hardware → Digital Output
    - MATLAB R2013b/R2014a: Simulink → Simulink Support Package for Arduino Hardware → Common → Digital Output

22. Save the Simulink model as any filename.

23. Double-click the Pulse Generator block, make the following changes and then hit OK:

    - Set "Pulse type" to Sample based

- Set “Period” to 4
- Set “Pulse width” to 2
- Set “Sample time” to 1 second



variable step solver.

Parameters

Pulse type: Sample based

Time (t): Use simulation time

Amplitude:

1

Period (number of samples):

4

Pulse width (number of samples):

2

Phase delay (number of samples):

0

Sample time:

1

☑ Interpret vector parameters as 1-D

Figure 4.19: Pulse Generator Parameters

24. Double-click the Arduino Digital Output block, make the following changes and then hit OK:

    - Set Pin number to 13, this is the pin the onboard LED is connected to.

25. Connect the Pulse Generator block to the Digital Output block. The model should look like Figure 4.18.

26. Now select Tools → Run on Target Hardware → Prepare to Run...

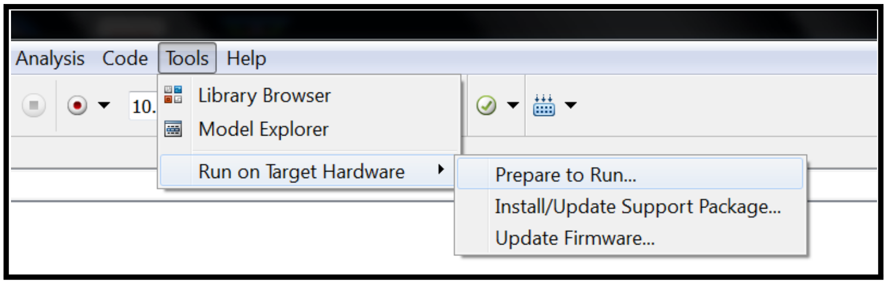
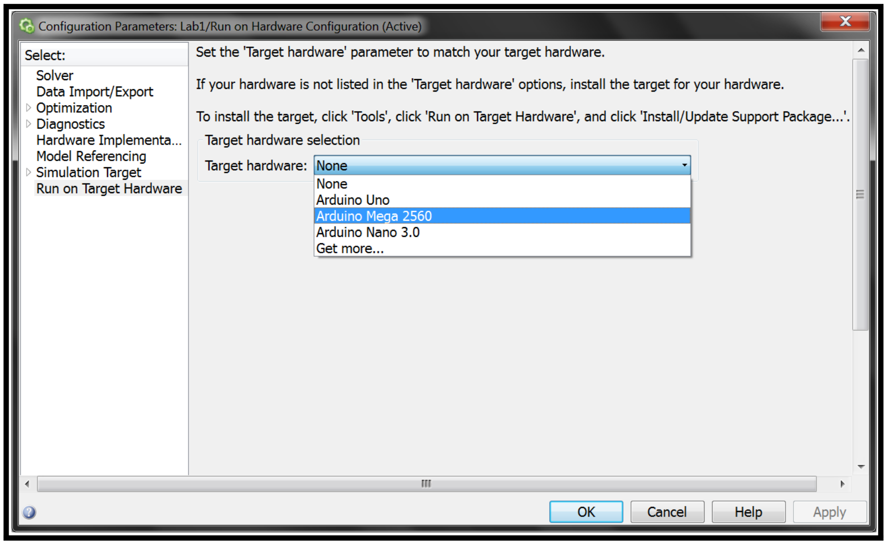


Figure 4.20: Prepare to Run

27. A window should appear as that in Figure 4.21. Set Target hardware to Arduino Mega 2560, and wait a moment for the window to update.


Configuration Parameters: Lab1/Run on Hardware Configuration (Active)

Set the 'Target hardware' parameter to match your target hardware.

If your hardware is not listed in the 'Target hardware' options, install the target for your hardware.

To install the target, click 'Tools', click 'Run on Target Hardware', and click 'Install/Update Support Package...'.

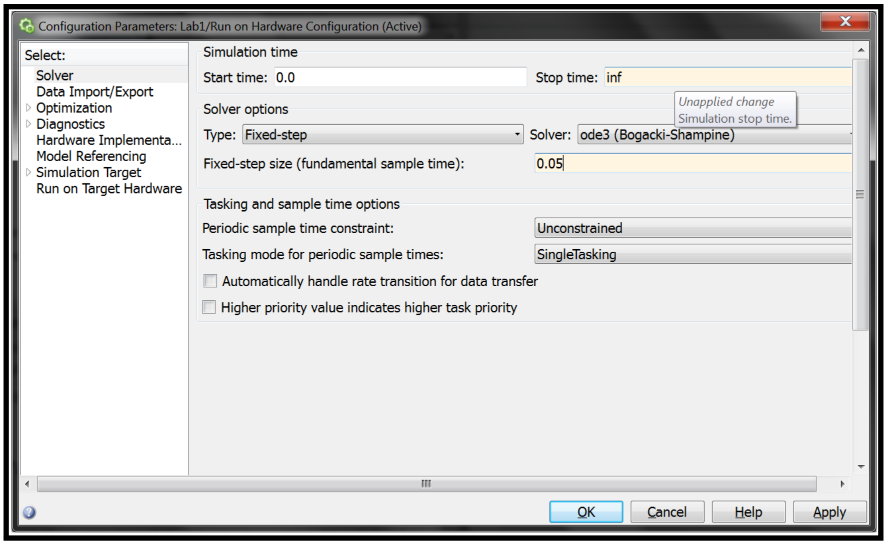
Target hardware selection

Target hardware: None

Figure 4.21: Target Hardware

28. Select Solver on the left hand side of the window, then make the following changes and hit OK.

- Set "Type" to *Fixed-step*
- Set "Fixed-step size" to *.05*



Configuration Parameters: Lab1/Run on Hardware Configuration (Active)

Select:
- Solver
- Data Import/Export
- Optimization
- Diagnostics
- Hardware Implementa...
- Model Referencing
- Simulation Target
- Run on Target Hardware

Simulation time

Start time: 0.0 Stop time: inf

Unapplied change
Simulation stop time.

Solver options

Type: Fixed-step Solver: ode3 (Bogacki-Shampine)

Fixed-step size (fundamental sample time): 0.05

Tasking and sample time options

Periodic sample time constraint: Unconstrained

Tasking mode for periodic sample times: SingleTasking

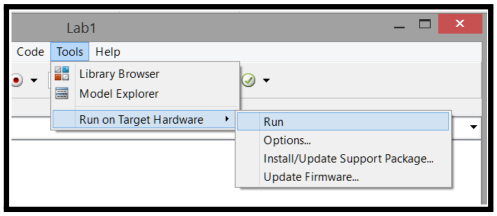
□ Automatically handle rate transition for data transfer

□ Higher priority value indicates higher task priority

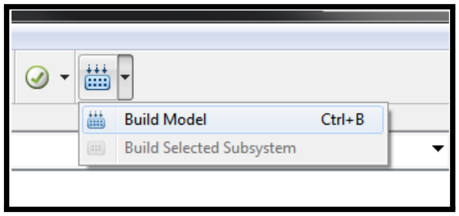
Figure 4.22: Solver Setup

29. Now run the model by selecting:

    - MATLAB R2013a: Tools → Run on Target Hardware → Run
    - MATLAB R2013b/R2014a: Click the "Build Model" button or Ctrl+B



(a) Run: R2013a



(b) Run: R2013b/R2014a

30. Wait until the the model has successfully downloaded to the board, now observe the onboard LED of the Arduino shown in Figure 4.24. Describe what the LED is doing. Estimate and record the timing. Is the timing consistent with the settings you made in Figure 4.19? Explain.

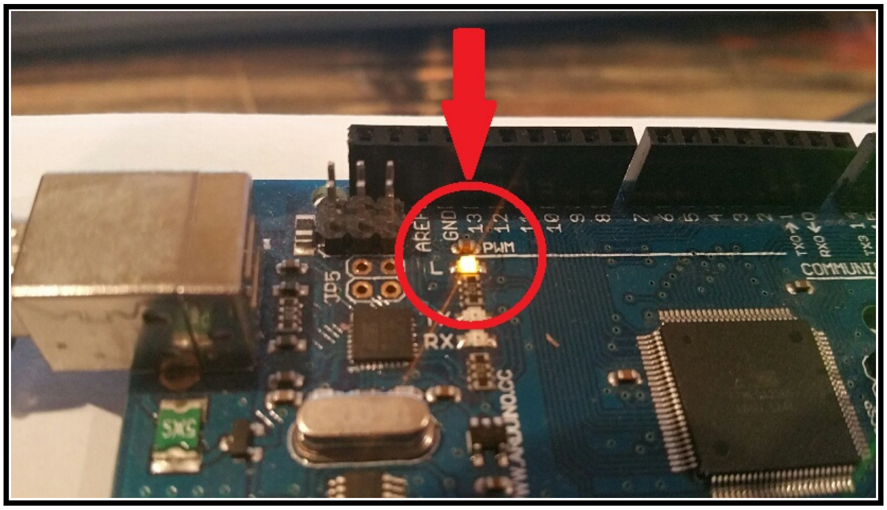


Figure 4.24: Arduino LED

- ⋆ The LED should be on for 2 seconds and off for 2 seconds because we set the Pulse Generator block's period to 4 samples, Pulse width to 2 samples, and Sample time to 1 second. Next, they will adjust these values, re-download them to the board, and observe the changes.

31. Change the Pulse Generator block sample time from 1 second to 0.1 seconds.

32. Download the modified model onto the board. Describe the changes in the LED operation. How has the timing changed? Did it change in the way that you expected?

    - ⋆ Now the student should notice that the LED blinking becomes faster since the sample time was decreased.

33. Change the parameters of the Pulse Generator block so that the LED will be on for 4 seconds and off for 1 second. Record your settings, download the modified model, and describe the resulting operation. If the timing does not produce a 4 second on period and a 1 second off period, make the necessary changes to the parameters of the Pulse Generator and test the operation again.

Troubleshooting: If there are any errors encountered throughout this section, try the following:

- If MATLAB cannot find the board when attempting to run the model, try setting the COM port manually. In Simulink, go to Simulation → Model Configuration Parameters. On the left side choose "Run on Target Hardware", then change "Set host COM port:" from Automatically to Manually. Then change the COM port number to the value written down in the installation section earlier.

- If Simulink cannot connect to the board either: try disconnecting and reconnecting the USB cable, or closing and reopening MATLAB.
- Any other issues encountered may be solved by reading the Arduino troubleshooting page located at `http://www.arduino.cc/en/Guide/Troubleshooting` .

<u>External Mode</u>

In External Mode the user is given the ability to tune parameters, and monitor data in real-time, without having to re-download the model each time parameters are adjusted. Once External mode is started, it uses the serial port to communicate between the board and the PC. Specifically, the serial port being used is port 0, which is used by the USB. Therefore, to run in External mode, the USB should be connected between the board and the PC. This ability to adjust and monitor changes in real-time is what makes External Mode such an enticing option for running these experiments. Students can benefit from being able to see the immediate effects their changes have on the hardware. However, this ability to make changes real-time comes at a cost. To communicate back and forth, the serial port is kept open and has a limited amount of bandwidth. This limitation affects the rate at which the program can run. External Mode's ability to make real-time adjustments will be shown in the following steps, using the Arduino's onboard LED.

34. Open a new Simulink Model, and name it something different from the previous model.

35. Add the following items from the Simulink Library Browser:

    - Two Pulse Generators: Simulink → Sources → Pulse Generator
    - Manual Switch: Simulink → Signal Routing → Manual Switch
    - Arduino Digital Output Block:
      - MATLAB R2013a: Simulink Support Package for Arduino Hardware → Digital Output

  - MATLAB R2013b/R2014a: Simulink Support Package for Arduino Hardware → Common → Digital Output

36. Save the model, and connect the blocks so that they look like Figure 4.25.

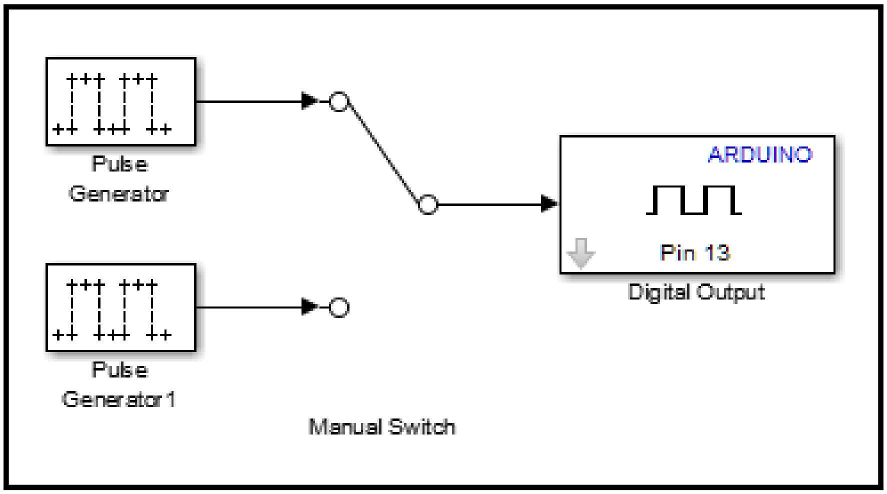


Figure 4.25: Simulink Model - External Mode

37. Double-click the Digital Output block and set the pin number to be 13.

38. Double-click the top Pulse Generator block, make the following changes and hit OK:

    - Set "Pulse type" to Sample based
    - Set "Period" to 4
    - Set "Pulse width" to 2
    - Set "Sample time" to 1 second

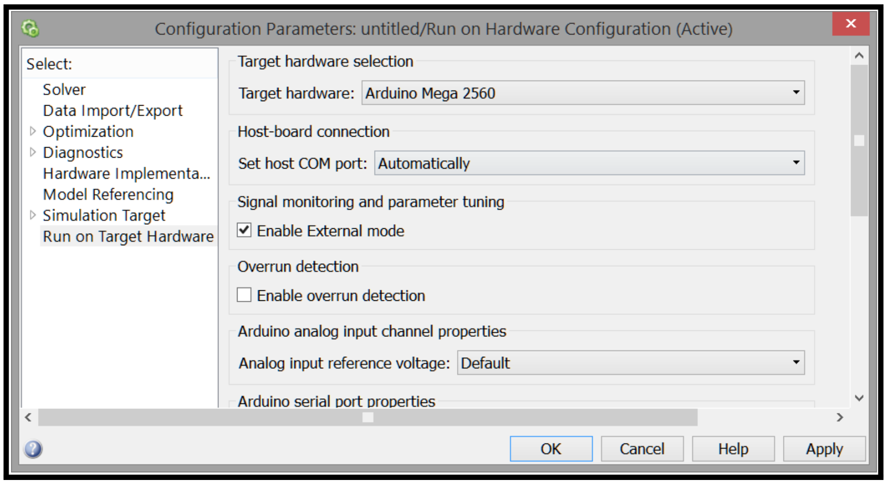
39. Double-click the bottom Pulse Generator block, make the following changes and hit OK:

    - Set "Pulse type" to Sample based
    - Set "Period" to 4
    - Set "Pulse width" to 2
    - Set "Sample time" to 0.1 second

40. Set up the Arduino to run in External Mode. Select Tools → Run on Target Hardware → Prepare to Run...

41. Set Target hardware to Arduino Mega 2560, and wait a moment for the window to update.

42. In MATLAB R2013a make sure to check the box next to "Enable External Mode", as shown in Figure 4.26. In R2013b or later, this step is skipped.



Configuration Parameters: untitled/Run on Hardware Configuration (Active)

Select:
- Solver
- Data Import/Export
- Optimization
- Diagnostics
- Hardware Implementa...
- Model Referencing
- Simulation Target
- Run on Target Hardware

Target hardware selection

Target hardware: Arduino Mega 2560

Host-board connection

Set host COM port: Automatically

Signal monitoring and parameter tuning

☑ Enable External mode

Overrun detection

☐ Enable overrun detection

Arduino analog input channel properties

Analog input reference voltage: Default

Arduino serial port properties

Figure 4.26: External Mode Enable

43. Select Solver on the left hand side of the window, then make the following changes and hit OK.

- Set "Stop time" to *inf*
- Set "Type" to *Fixed-step*
- Set "Fixed-step size" to *.05*

44. Make sure the drop-down menu shown in Figure 4.27 is set to External.

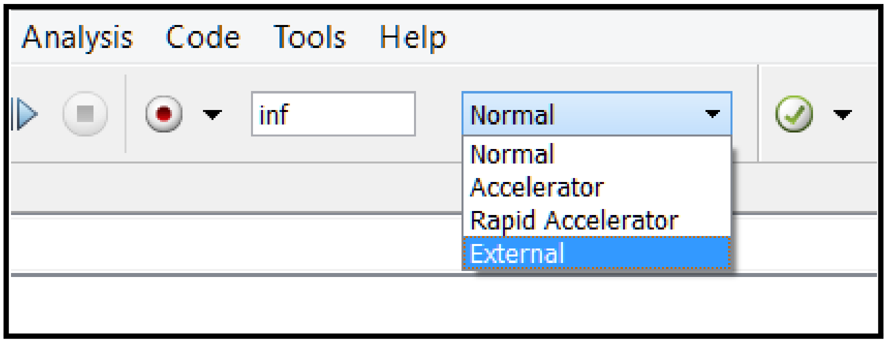


Figure 4.27: External Mode Enable

45. Download the model to the Arduino using the same method used for Normal Mode. The program may not start running automatically so connect or run the model by doing the following:

    - MATLAB R2013a: Connect to the Target Hardware using , as shown in Figure 4.28. If the Run button ▶ is not grayed out after connecting to the hardware, you will need to press it also to starting running the model in External Mode.

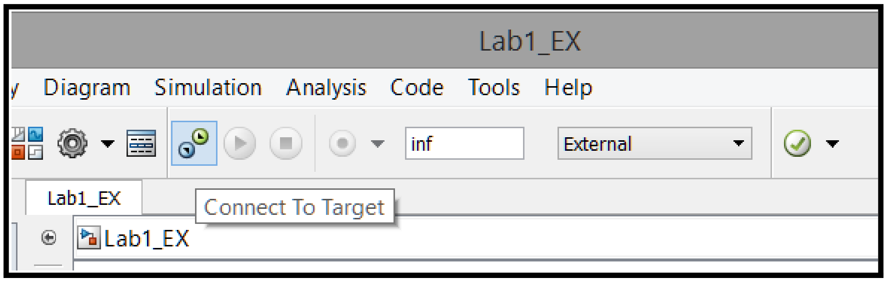


Figure 4.28: External Mode Connect: R2013a

    - MATLAB R2013b or later: Press the Run button ▶, as shown in Figure 4.29.

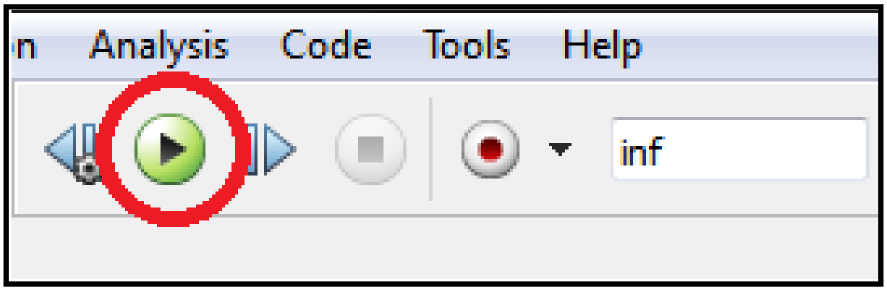


Figure 4.29: External Mode Connect: R2013b or later

46. Record what the Arduino's LED is doing. Estimate timing. Is the timing consistent with the parameter settings?

    ⋆ The LED should be blinking ON for 2 seconds, and off for 2 seconds, like it was in the first part for Normal Mode, since the Pulse Generator has the same parameter values.

47. Double click the Manual Switch in the Simulink Model. The switch should now be connecting the bottom Pulse Generator to the Digital Output.

48. Record what the Arduino's LED is doing now. Describe any timing changes. Explain why the timing changed. How is the process different than Normal Mode? Which Mode is better? Why?

    ⋆ The LED should now be blinking ON for .2 seconds, and off for .2 seconds, just as it was in the second part of the Normal Mode experiment. The student should now be able to see that using External Mode they were able to change the response of the LED without having to make changes and re-download the model to the board.

49. Disconnect from the hardware:

    - MATLAB R2013a: Press the Disconnect from Target button 🔗, as shown in Figure 4.30.

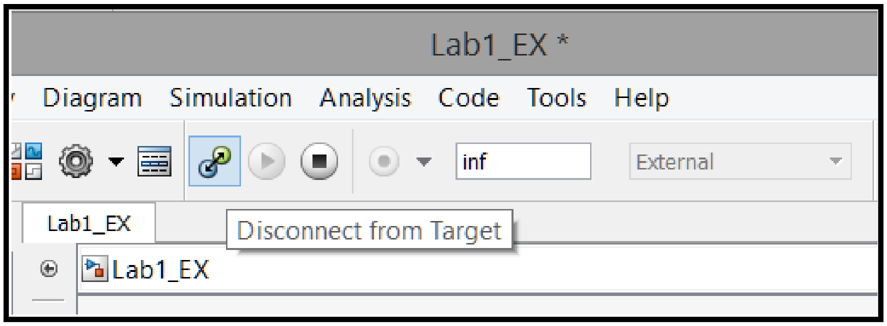


Figure 4.30: External Mode Disconnect: R2013a

    - MATLAB R2013b or later: Press the Stop button ⏹, as shown in Figure 4.31.

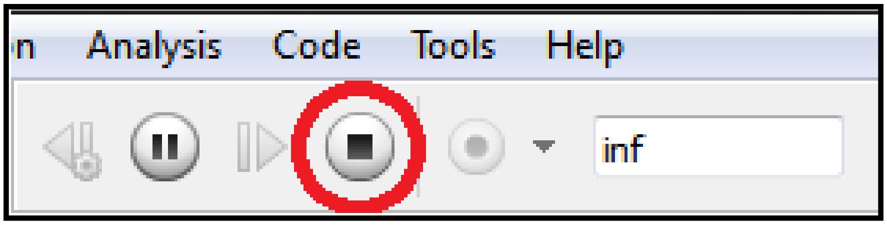


Figure 4.31: External Mode Disconnect: R2013b or later

50. Disconnect the USB from the Arduino and PC.

Troubleshooting:

- The most common error that may be encountered using External Mode is Simulink being unable to connect to the Arduino. This can happen when running in External mode if a previous simulation was not stopped correctly. The serial port may have been left open creating issues when trying to reconnect to the board. Another possible cause is that the model was not rebuilt after making changes. Possible fixes for this error are:
  - Try rebuilding the model using Ctrl+B in MATLAB R2013b or later or Tools → Run on Target Hardware → Run in R2013a.
  - Disconnect the USB from the Arduino and reconnect. (This usually fixes most of the errors).
  - Try hitting the reset button on the Arduino, shown in Figure 4.32.

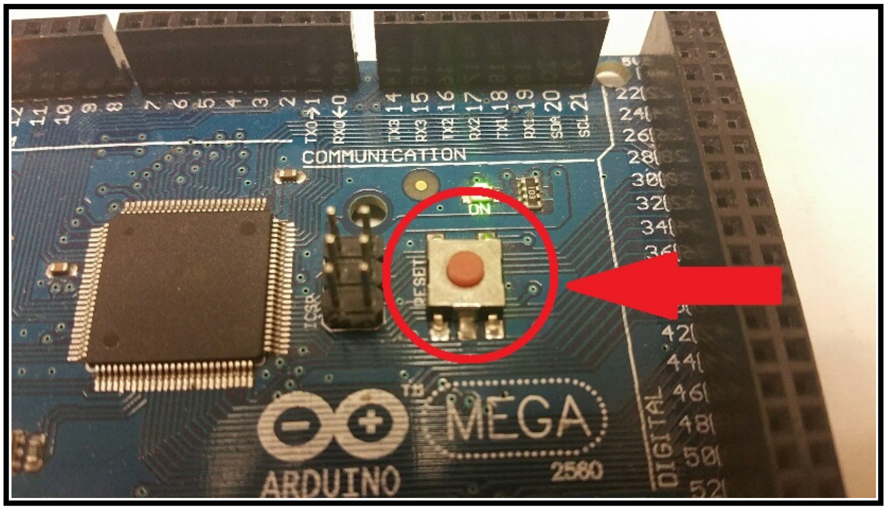


Figure 4.32: Arduino Reset Button

  - Close MATLAB and re-open.
  - Any other issues encountered may be solved by reading the Arduino troubleshooting page located at `http://www.arduino.cc/en/Guide/Troubleshooting`.

4.4 Conclusion/Student Feedback

This section will conclude the experiment and reiterate the important topics. Student feedback will also be provided in this section to show the effectiveness and difficulty of the experiment from the student's perspective. This introductory lab was constructed to show how to install the necessary support for the Arduino in Simulink, and on the PC. Control of the Arduino's onboard LED was used to verify everything was installed correctly. There were two methods used to test this - Normal Mode and External Mode. This lab only provided a glimpse of these two modes. A more in depth comparison of the two modes is provided in the experiment *Sampling and Data Acquisition*.

CHAPTER 5

SAMPLING AND DATA ACQUISITION

5.1 Objective

This experiment is designed to show some of the key differences between using Normal Mode and External Mode to run labs on the Arduino using Simulink. Differences in sampling and data acquisition for the two modes are the main focus. The experiment will review the concept of sampling, and will show how to acquire data at multiple sampling rates for both Normal Mode and External Mode using a DC motor. This will give an idea of how difficult it is to acquire data using both methods, and also provide insight on which mode may be best for running experiments based on sampling needs.

5.2 Setup

5.2.1 Required Materials

Hardware

- DC Motor with Encoder
- Arduino Mega 2560 Rev3
- Motor Shield (DFRobot L298P)
- 12V DC Power Supply (12VDC 3A Wall Adapter Power Supply)
- USB cable (Standard A to B plug)
- Female Barrel Jack (2.1mm x 5.5mm Female CCTV Power Jack Adapter)
- 8 wires
- 3-D Printed Base
- 3-D Printed Motor Clamp

- 3-D Printed Load (from Experiment: Intro to 3-D Printing)
- 2 #4-40 1/2" screws
- 2 #4-40 1/4" screws
- Small Flat screwdriver
- Small Phillips screwdriver
- Sticky Tack
- Sandpaper or Sanding Sponge (Optional)

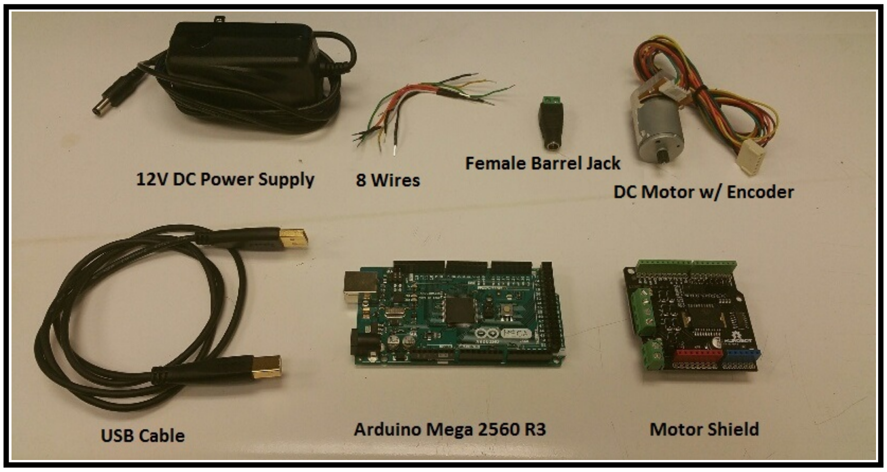


Figure 5.1: Hardware Items(1)

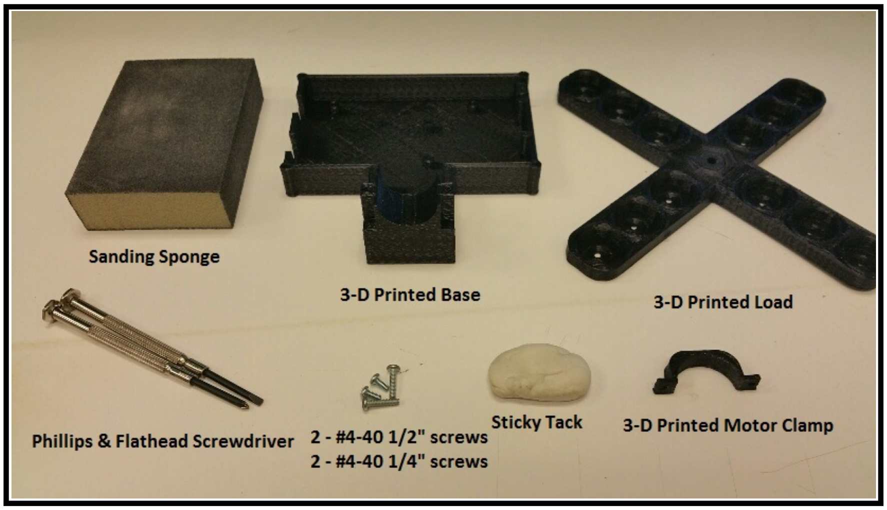


Figure 5.2: Hardware Items(2)

Software

- MATLAB/Simulink 2013a or later

  ⋆ *The steps and images related to MATLAB/Simulink for this experiment were created using MATLAB/Simulink 2013a. Therefore some steps and images may be a little different if you are not using this version. If you are in fact using a different version, make sure you know the steps for running models onto the Arduino for your version of Simulink.*

- MATLAB/Simulink files
- Take Home Labs Arduino Library
- 3-D Printer files

Prerequisite Experiments

The experiments listed below should be completed prior to this one. Steps and hardware from these experiments may be needed or referenced throughout this experiment. It is assumed that you already have the Arduino communicating with the PC, know how to set up and run models in Normal and External Mode, can troubleshoot any of the issues discussed in these three labs, and can 3-D print the required parts.

- Blinking LED
- Simple DC Motor
- Intro to 3-D Printing

5.2.2 Software Setup

The following steps cover the process of downloading and creating the necessary files needed for this experiment. The first section covers downloading the required Matlab/Simulink files from the website, and saving them in the MATLAB directory that will be used. Downloading and adding additional Simulink blocks to the Simulink Library is covered in the second section. The third section requires downloading the 3-D printer files dedicated to this experiment, so that they can be printed.

Matlab/Simulink Files

1. Open your internet browser and navigate to `csl.okstate.edu`

2. On the left side of the Homepage, select “Experiments”

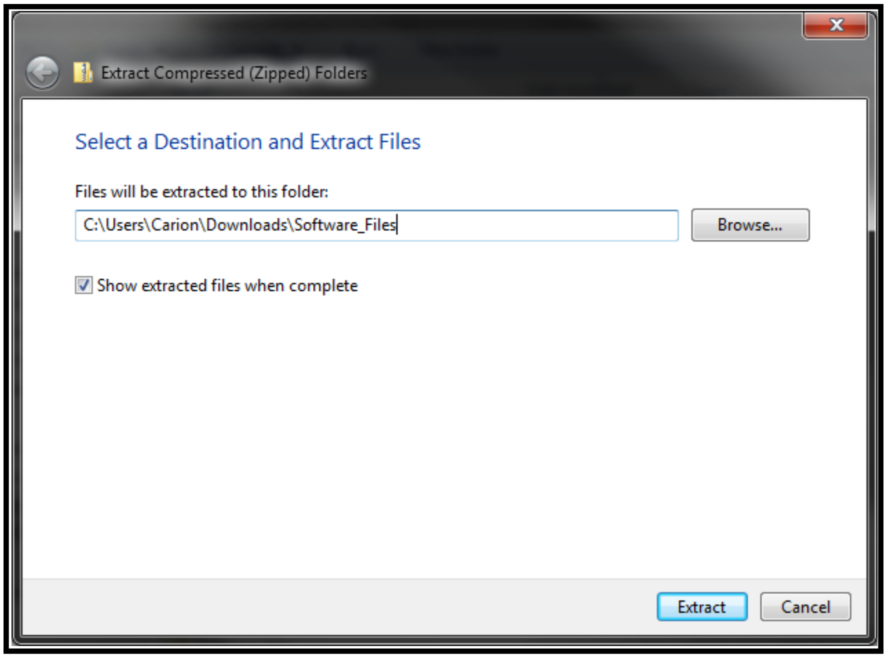
3. In the middle section of the Experiments page select “All”

4. In the middle section of the All Experiments page find and select *Sampling and Data Acquisition*

5. On the Sampling and Data Acquisition page select “Software/Code” in the rightmost section. Download the zipfile named *Software_Files*.

6. Right-click the file and choose “Extract All...”, or use any other method of extracting the files on your PC.

7. You will now need to set the path for these files, and you should extract the folder somewhere such as the Downloads or Documents folder. Make sure the box next to “Show extracted files when complete” is checked, as shown in Figure 5.3. Now select “Extract”



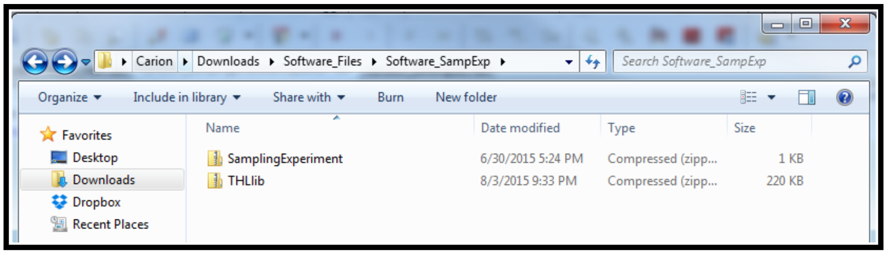
Select a Destination and Extract Files

Files will be extracted to this folder:
C:\Users\Carion\Downloads\Software_Files

☑ Show extracted files when complete

Figure 5.3: Zipped Folder Extract

8. A new window should open. Double-click the *Software_SampExp* folder and ensure the zipfiles in Figure 5.4 are shown



Name	Date modified	Type	Size
SamplingExperiment	6/30/2015 5:24 PM	Compressed (zipp...	1 KB
THLlib	8/3/2015 9:33 PM	Compressed (zipp...	220 KB

Figure 5.4: Software Files

9. Right-click the *SamplingExperiment* folder, and select "Extract All..."

10. Set the path for the folder to be extracted somewhere other than the current location, and select "Extract". You will need to know this folder location later in the Experimental Procedures section, so you may want to write it down. You can close the folder if it opens up after extracting it.

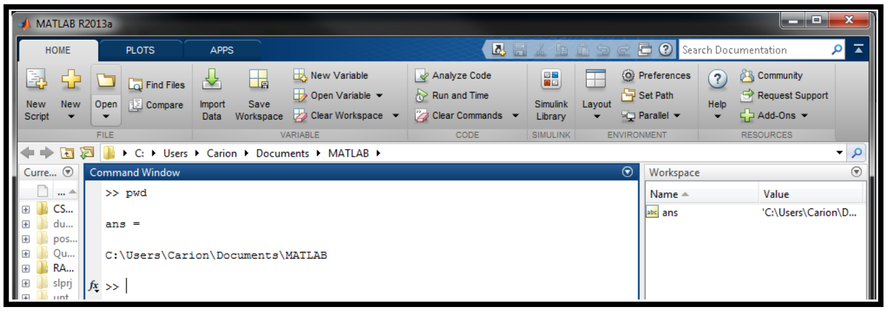
11. Leave the *Software_SampExp* folder that contains the zipfiles open, and proceed to the next section.

Additional Arduino Support Package

Many of the Take Home Labs experiments listed on the website require some type of hardware, other than the Arduino. The hardware may include DC motors, encoders, motor drivers, ultrasonic sensors, etc. In order to make it easier to interface with these hardware components, there are custom Simulink S-Function blocks that have been created. Essentially, the S-Function is just a Simulink block representation of some code. More information about S-Functions can be found on the Mathworks website. The Arduino coding language is just a set of C/C++ functions, and the S-Function blocks you will download contain the code that is needed to interface with these hardware elements. This eliminates the need to to write C code, and it also helps keep the Simulink model organized.

The S-Function blocks that will be needed throughout the experiments will all be contained in one location in the Simulink Library Browser. It should be mentioned that the additional library that will be added is a modified version of the Rensselaer Arduino Support Package (RASPlib). Their library is available on their website `http://homepages.rpi.edu/~hurstj2/` along with their own curriculum dedicated labs for their own specific hardware that you can purchase. The concept of their labs being dedicated to a curriculum and run with the Arduino and Simulink is similar to the concept of our take home labs. This made their labs a good guide for working with the Arduino and Simulink. Therefore, after learning how to use their library, some of the custom blocks and files they created were modified to work with our experiments and hardware. More information about their labs and what they do can be found on `http://minseg.webs.com/`. Follow the steps below to add the additional support to Simulink needed for this experiment.

12. Open MATLAB, and in the Command Window type *pwd* to get MATLAB's home directory on your PC. Copy the location or write it down.



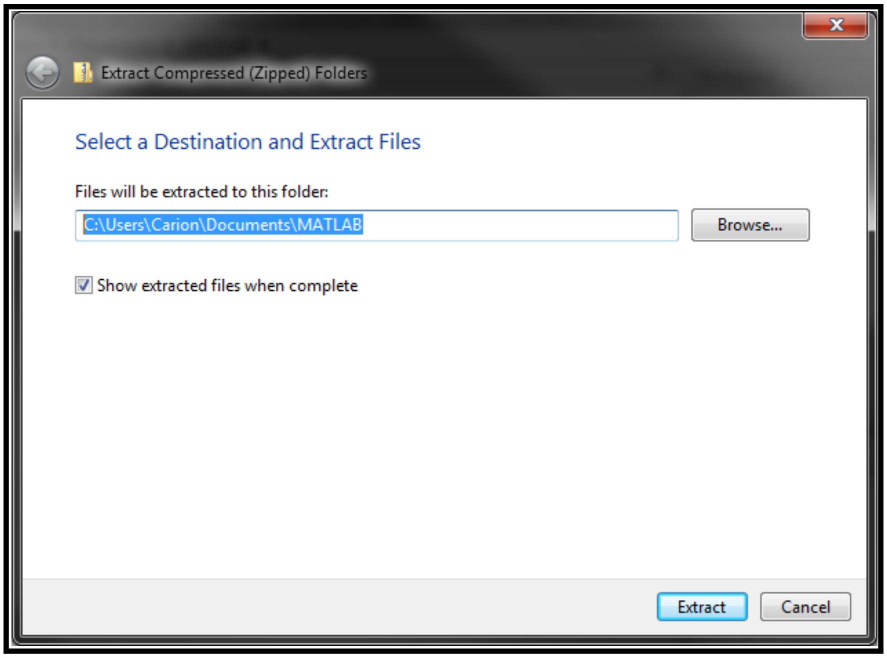
```
>> pwd

ans =

C:\Users\Carion\Documents\MATLAB

fx >> |
```

Name	Value
ans	'C:\Users\Carion\D...

Figure 5.5: MATLAB Home Directory

13. In the *Software_SampExp* folder, shown in Figure 5.4, right-click the *THLlib* folder and choose "Extract All..."

14. Extract the files to the location of your MATLAB home directory. You can paste the directory if you copied it, browse and find it, or manually type it in.



Extract Compressed (Zipped) Folders

Select a Destination and Extract Files

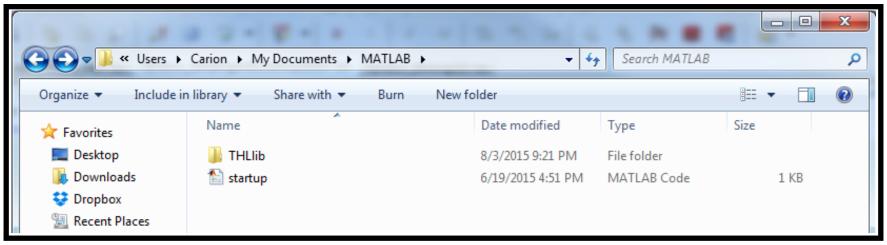
Files will be extracted to this folder:

C:\Users\Carion\Documents\MATLAB

☑ Show extracted files when complete

Figure 5.6: Support Folder

15. The *THLlib* folder should now be in the MATLAB home directory. There should also be a MATLAB code file named "startup" in the same location, as shown in Figure 5.7. If the file does not exist, navigate back to MATLAB.

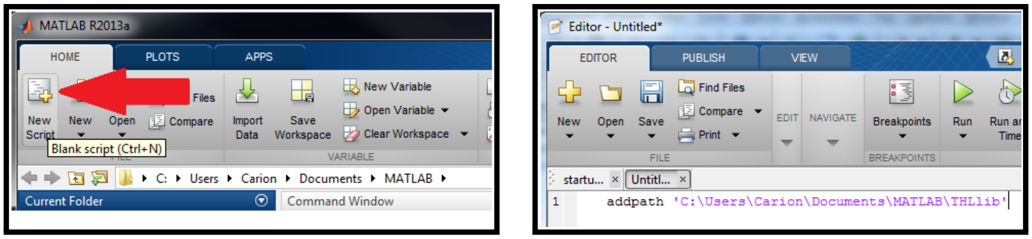


« Users ▸ Carion ▸ My Documents ▸ MATLAB ▸

Name	Date modified	Type	Size
THLlib	8/3/2015 9:21 PM	File folder	
startup	6/19/2015 4:51 PM	MATLAB Code	1 KB

Figure 5.7: Directory Folder

16. In MATLAB, select "New Script" , and in the script type your path to the *THLlib* folder, as shown in Figure 5.8b for my path. Make sure to put the single quotes around the folder path.



```
addpath 'C:\Users\Carion\Documents\MATLAB\THLlib'
```

(a) New Script

(b) Startup File

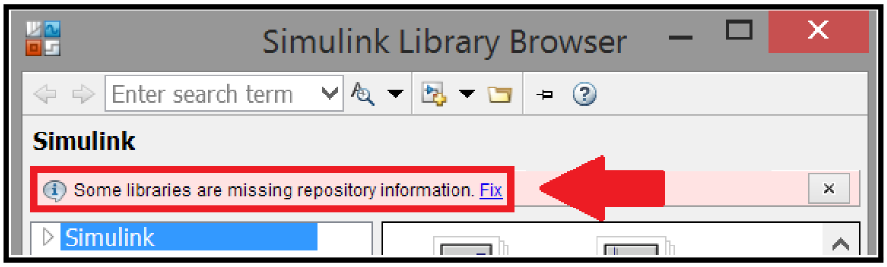
Figure 5.8: Creating Startup File

17. Save the file as *startup*. This script will execute every time MATLAB starts, and will add the path to the *THLlib* folder each time automatically. Close and reopen MATLAB.

18. After you reopen MATLAB, type *simulink* in the MATLAB Command Window. Make sure the Library "Take Home Labs Arduino Support Package" is now shown in the Simulink Library Browser. If it is not shown, follow the instructions described in the sub-steps below.

    (a) The Simulink library "Take Home Labs Arduino Support Package" was created with version R2013a. Newer versions will need to fix the Simulink environment so that the the older library will show. In the Simulink Library Browser window, press F5 or View → Refresh Tree View.

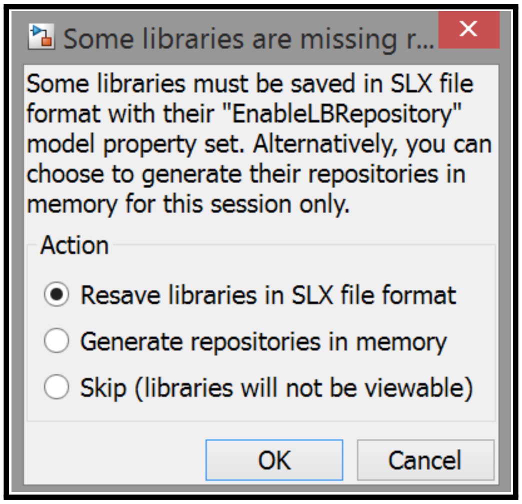
    (b) After the library refreshes, the message "Some libraries are missing repository information. Fix" will appear at the top of the window as shown in Figure 5.9. Press "Fix".



Some libraries are missing repository information. Fix

Figure 5.9: Simulink THL Library Error

    (c) The window shown in Figure 5.10 should appear. Select "Resave libraries in SLX file format" and press OK.


Some libraries are missing r...

Some libraries must be saved in SLX file format with their "EnableLBRepository" model property set. Alternatively, you can choose to generate their repositories in memory for this session only.

Action

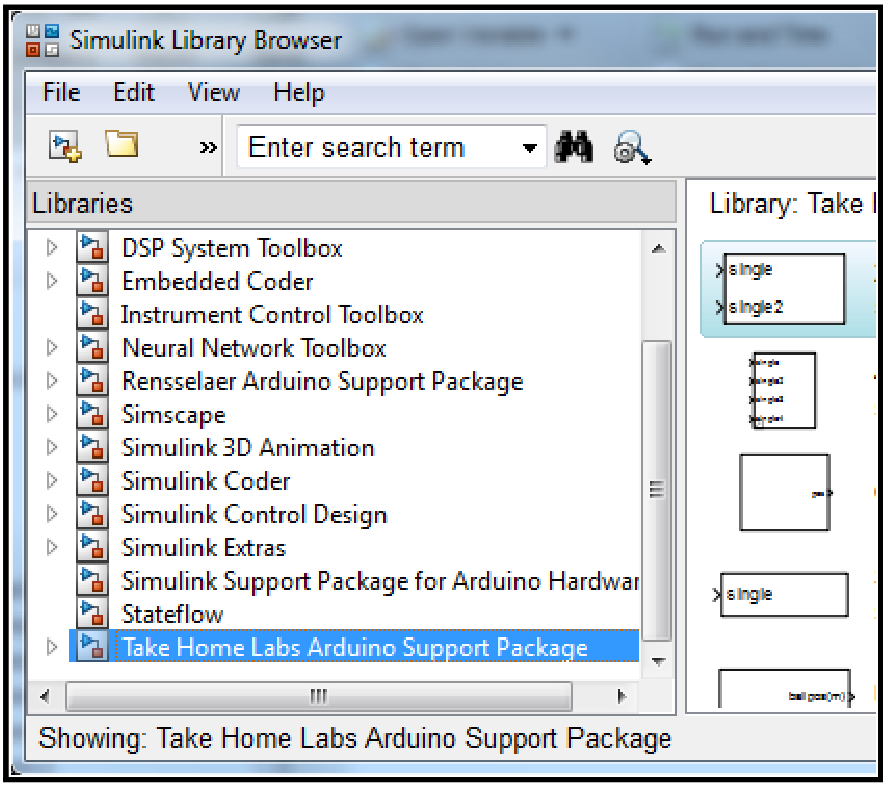
● Resave libraries in SLX file format

○ Generate repositories in memory

○ Skip (libraries will not be viewable)

Figure 5.10: Simulink THL Library Error (2)

(d) Once the library is fixed, the "Take Home Labs Arduino Support Package" should now show, as in Figure 5.11. If it does not show, try closing and reopening MATLAB. Then recheck to see if the library shows by navigating back to the Simulink Library Browser.


Simulink Library Browser

Libraries

- DSP System Toolbox
- Embedded Coder
- Instrument Control Toolbox
- Neural Network Toolbox
- Rensselaer Arduino Support Package
- Simscape
- Simulink 3D Animation
- Simulink Coder
- Simulink Control Design
- Simulink Extras
- Simulink Support Package for Arduino Hardwar
- Stateflow
- Take Home Labs Arduino Support Package

Library: Take I

Showing: Take Home Labs Arduino Support Package

Figure 5.11: Simulink THL Library

3-D Printing Files

In this section you will be required to download all of the 3-D printer files for this lab, and then print them. Assembling the parts will be covered in the Hardware Setup section.

19. Navigate back to the Normal vs External Mode and Sampling page on the Take Home Labs website.
20. In the rightmost section, select “3-D Printer Files”.
21. Open the zipfile and extract the files.
22. Use the techniques learned in experiment *Intro to 3-D Printing* and print the two .STL files provided in the folder.

5.2.3 Hardware Setup

This section contains the instructions for assembling the hardware components. First, the DC motor, Arduino, and motor shield will be connected to the base. Next, the DC motor and encoder will be connected to the Arduino, and the 3-D printed load will be attached to the motor. Connecting the motor shield to the Arduino is covered in the *Simple DC Motor* experiment, therefore these steps will not be shown in this experiment.

Base, Arduino, and Motor Assembly

23. Set the DC motor onto the base as shown in Figure 5.12.

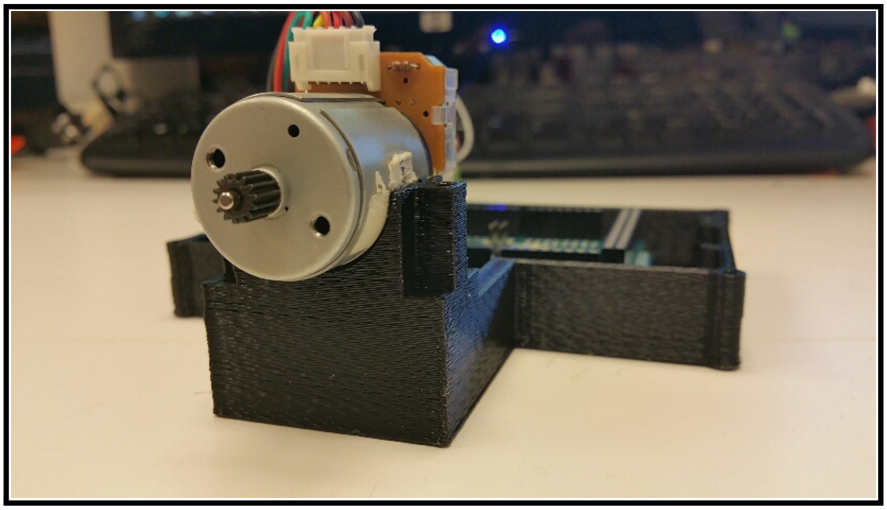


Figure 5.12: Motor and Base

24. Place the DC motor clamp over the DC motor so that the drill holes line up with the holes of the bottom piece.

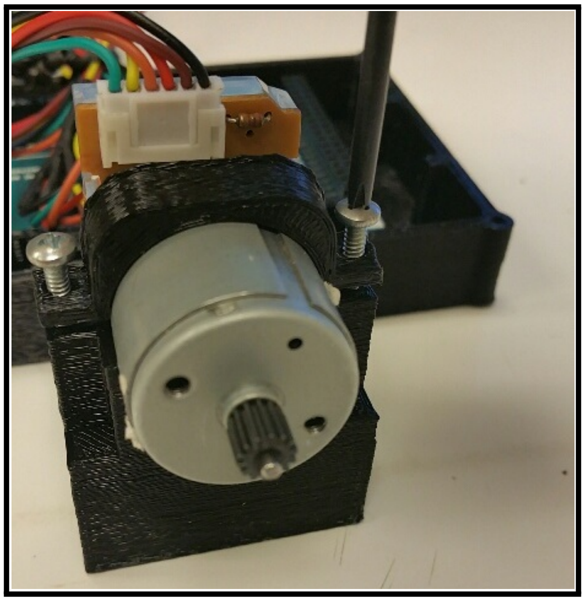


Figure 5.13: Motor Clamp

25. Use two #4-40 1/2" screws to tighten the DC motor mount top piece around the motor so that it is secure.

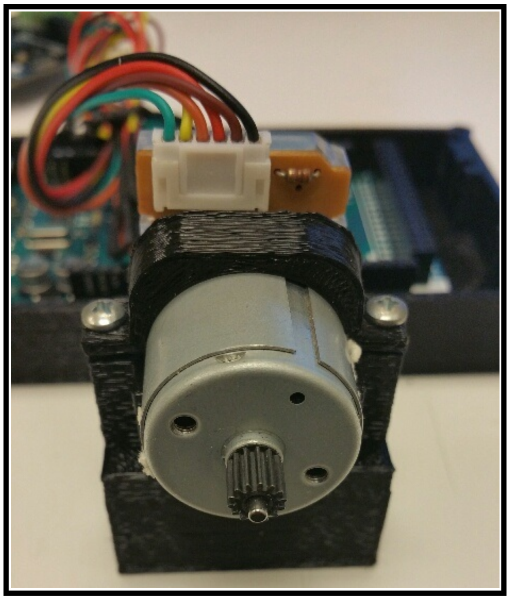


Figure 5.14: Securing the Motor

26. Take the Arduino and position it on the base so that the drill holes line up. If you still have the motor shield connected to the Arduino from the *Simple DC Motor* experiment, and you do not want to screw the Arduino onto the base, you can skip the next step. Otherwise, you will need to disconnect the motor shield from the Arduino to screw the board onto the base, and then reconnect the motor shield back onto the Arduino after.

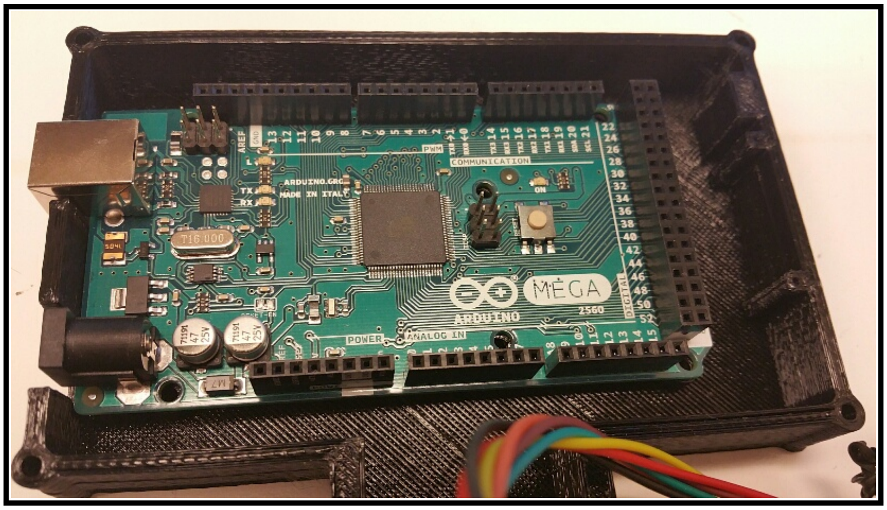
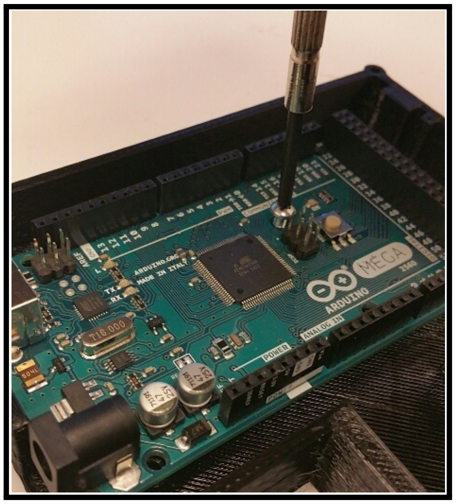
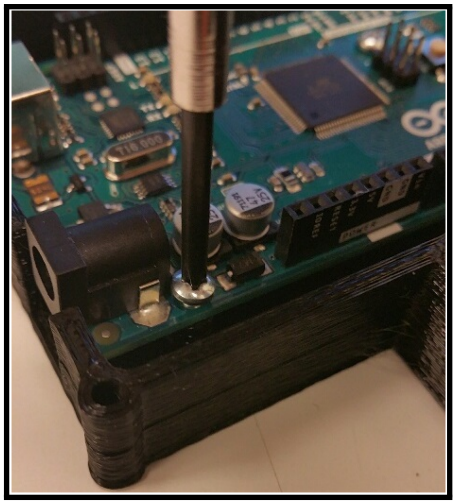


Figure 5.15: Arduino on Base

27. Screw the Arduino onto the base using at least two screws, as shown in Figure 5.16b. The screws may not tighten down completely; this is fine.



(a) Screw 1



(b) Screw 2

Interconnections

Now that the base and Arduino have been connected, remember to connect the motor shield back to the Arduino, if you removed it. The DC barrel jack and motor should be connected to the motor shield as well. This was completed in steps 1-8 of the *Simple DC Motor* experiment. If you have not completed this, or if either of them have been disconnected, follow the steps in that experiment. Be sure to notice that the wires and power supply you will use in this experiment may be different from those shown in that experiment. Therefore, the figures used in that section will not exactly reflect what your final setup will look like. The main concern is that the motor and female barrel jack should both be connected to the motor shield once steps 1-8 of the *Simple DC Motor* experiment are completed, as shown in Figure 5.17.

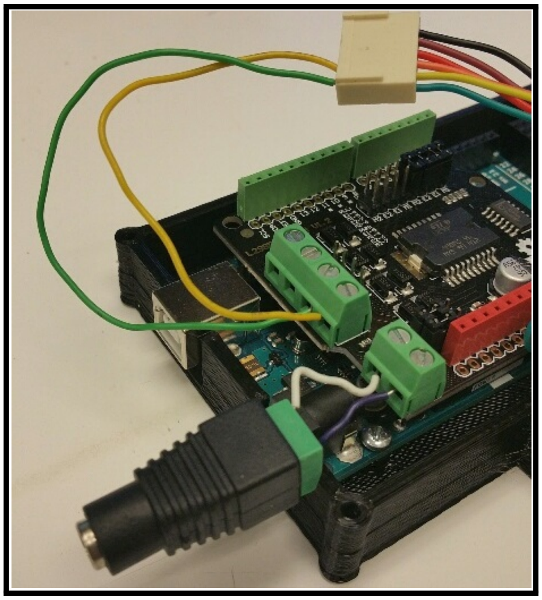
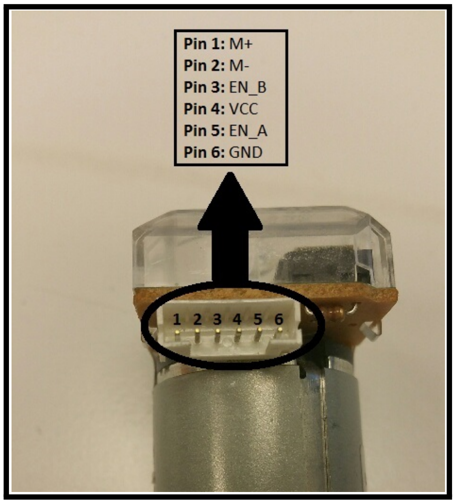
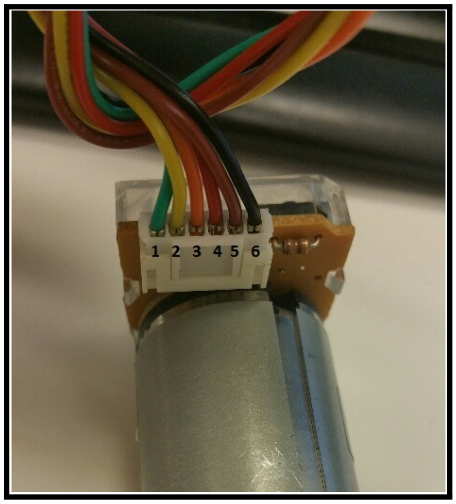


Figure 5.17: Connections from Simple DC Motor Experiment

The motor and encoder pinout is shown in the *Simple DC Motor* experiment, but it is also shown below. The wire connector may change colors in the future, so do not rely on them in the figures. Focus on the label for each pin.



(a) Motor Pinout



(b) Motor Pinout w/ Wires

28. The motor's encoder, pins 3-6, needs to be connected next. Place four wires into the remaining slots of the motor's connector.

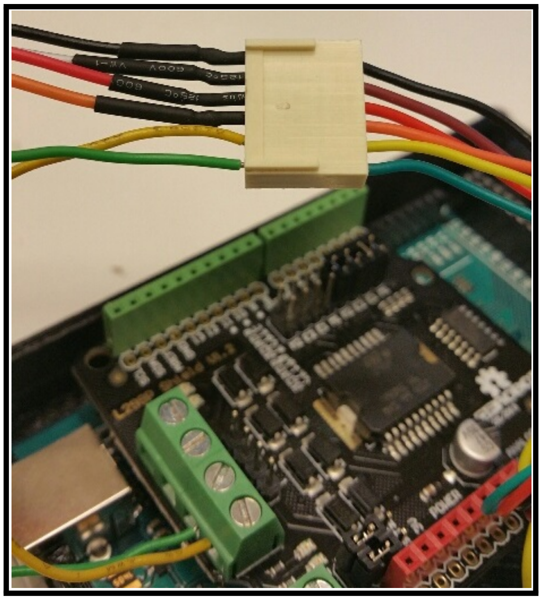


Figure 5.19: Encoder Connection

29. Take the wire connected to the “EN_B” slot, pin 3, and connect it to digital pin 3 of the motor shield. Connect the wire in the “EN_A” slot, pin 5, to digital pin 2 of the motor shield.

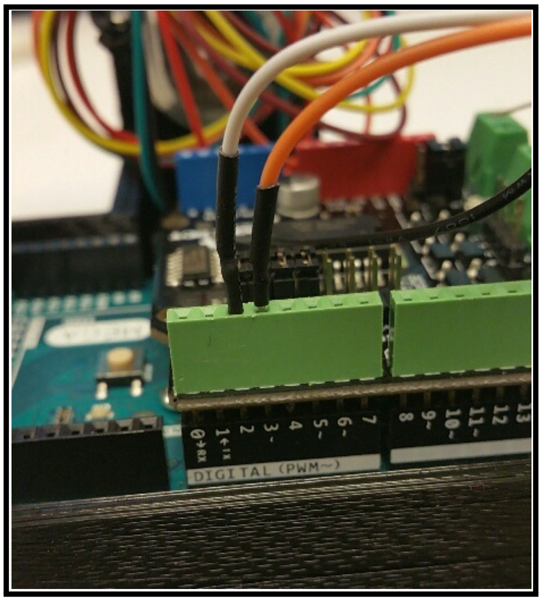


Figure 5.20: Enc_A and Enc_B Connect

30. Connect the wire in the “VCC” slot, pin 4, to the 5V pin on the motor shield. Connect the wire in the “GND” slot, pin 6, to any of the GND pins of the motor shield.

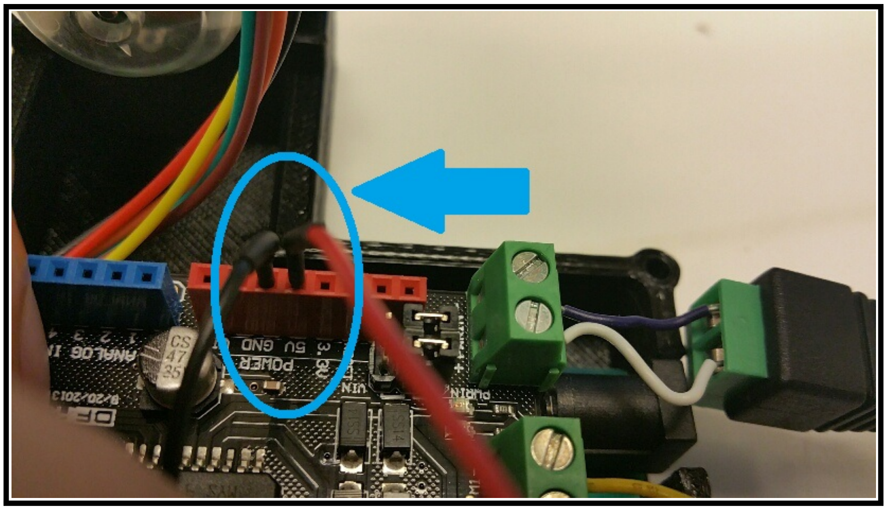


Figure 5.21: VCC and GND Connect

31. The motor and encoder should now both be connected to the motor shield. Verify the encoder is receiving power by connecting the USB cable from the Arduino to the computer. The encoder should now be powered, so a red light inside the encoder's enclosure should turn on.

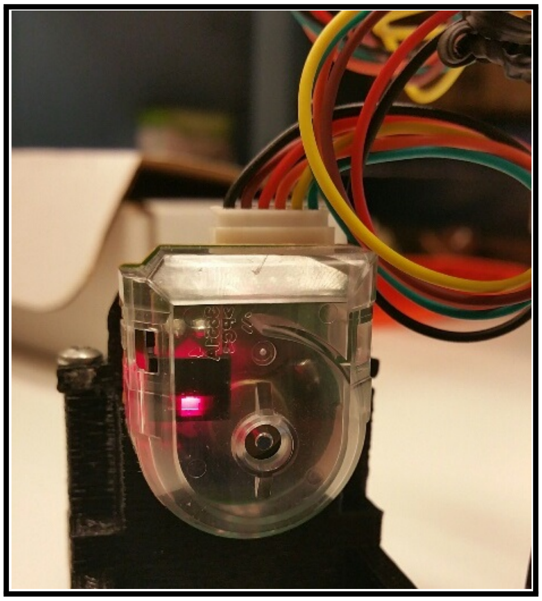
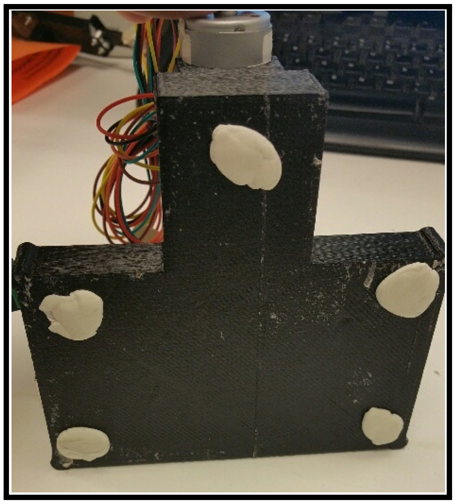
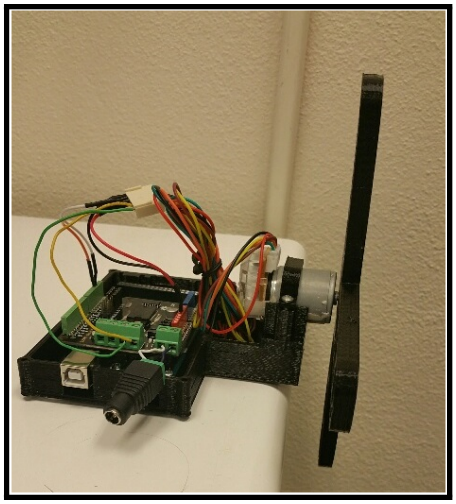


Figure 5.22: Encoder Power ON

32. Finally, attach the load to the motor, which you printed in the *Intro to 3-D Printing* experiment. Then place the base onto the edge of a table so that the load has space to spin. Place some sticky tack in different places on the bottom of the base to help the base stay in place during experiments.



(a) Sticky Tack



(b) System on the Edge of Table

5.3 Experimental Procedures

Digital systems, such as a PC and the Arduino, have limitations on how fast they can perform tasks. The rate at which the Arduino is able to sample is the key limitation that you will explore. Sampling is the process of creating a discrete-time signal from a continuous-time signal. The value of a continuous signal is sampled, or measured, at certain time intervals. Depending on the continuous signal being sampled, there are requirements on how fast you need to sample in order to preserve the information in the signal. The first exercise of this section will discuss sampling theory, using Simulink to help illustrate the idea. This will be done by sampling a sine wave at different rates and observing the output response. The second exercise of this section will explore sampling with the Arduino using Normal Mode. A sine wave voltage will be input into the motor. Since the motor is a linear system, the output position and velocity should be sine waves of the same frequency. The amplitude and phase of the output sine waves will change with frequency. Various sampling rates will be tested to provide an idea of how fast or how efficiently the Normal Mode samples data. The third exercise of this section is the same as exercise 2, but using External Mode.

Once these experimental procedures are complete, you will have a better understanding of what sampling is, and how to select a good sampling rate for an input of a certain frequency. Also, you will know how to acquire data in both Normal and External Mode, and you will understand the strengths and weaknesses of each method.

5.3.1 Exercise 1: Sampling Theory with Simulink

To help illustrate what sampling is, look at Figure 5.24 below. The signal at the top of the figure is one cycle of the sine wave, $A\sin(2\pi ft+\phi)$. A is the amplitude, f is the frequency, and ϕ is the phase. As shown in the figure, T equals the period of the signal. The period of the sine wave is the time it takes to complete one full cycle. The signal at the bottom of the figure shows the same sine wave, but it is sampled every T_s seconds, which is the sampling period. This means that every T_s seconds, the value of the top signal in the figure is being recorded. The sampled sine wave has a sampling frequency of 16/T Hz, which corresponds to a sampling period of 0.0625*T seconds.

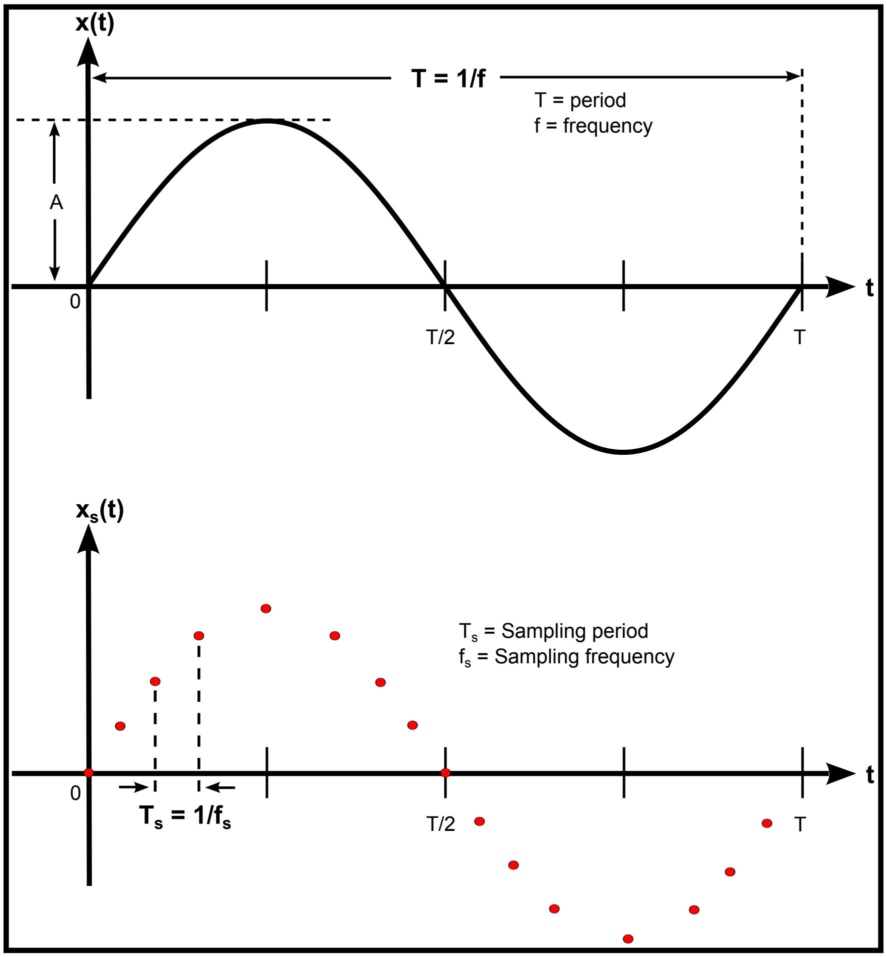


Figure 5.24: Continuous and Discrete Sine Wave

The Nyquist Sampling Theorem states that a band-limited signal can be recovered if the sampling frequency f_s is greater than twice the highest frequency of the signal being sampled. This is shown in Equation 5.1, where the term $2f_{max}$ is known as the

Nyquist Rate.

$$f_s > 2f_{max} \tag{5.1}$$

If the signal is not sampled at greater than the Nyquist Rate, then a phenomenon known as aliasing occurs. What this means is that higher frequency components will be aliased or folded in and appear as lower frequency components in the output signal. This would prevent the signal from being reconstructed properly. The following steps of this subsection will help illustrate this idea.

33. Open MATLAB, and select the "Browse for folder" icon

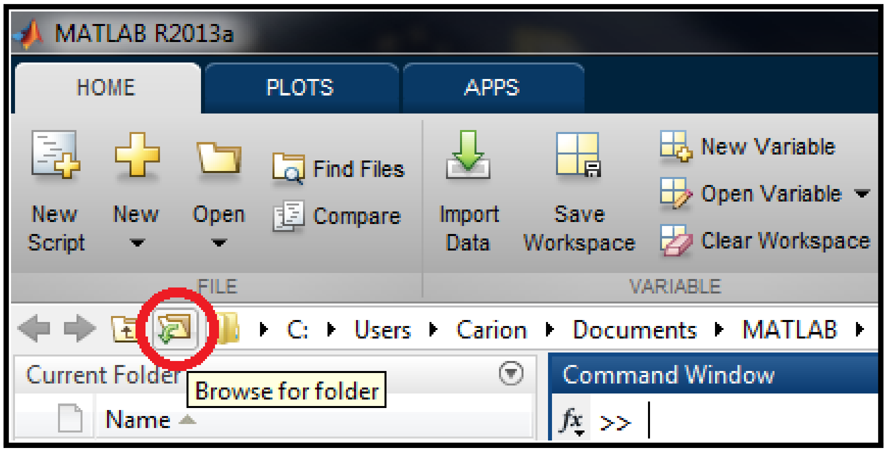
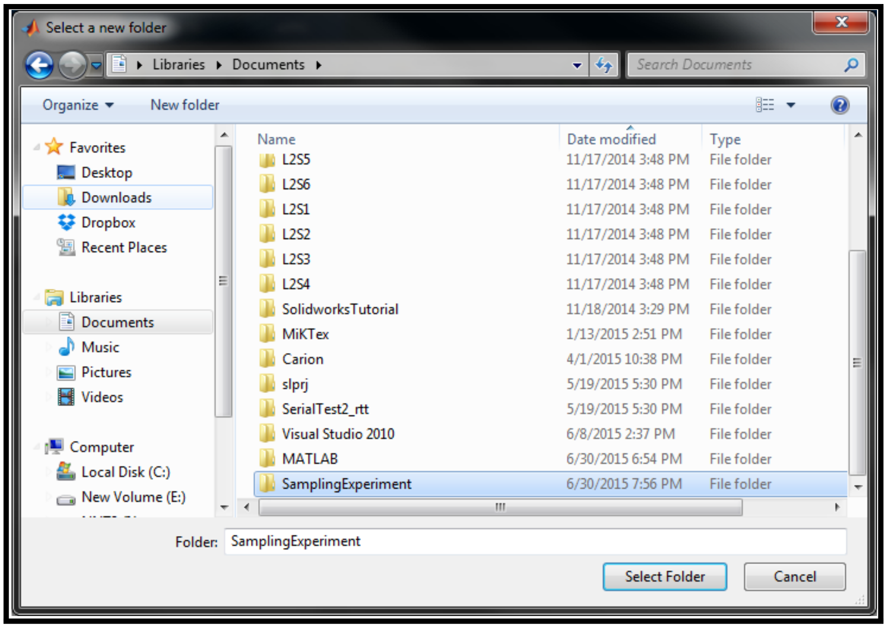


Figure 5.25: Browse for Folder

34. Navigate to the location where you extracted the *SamplingExperiment* folder in the software setup section, and press the "Select Folder" button.

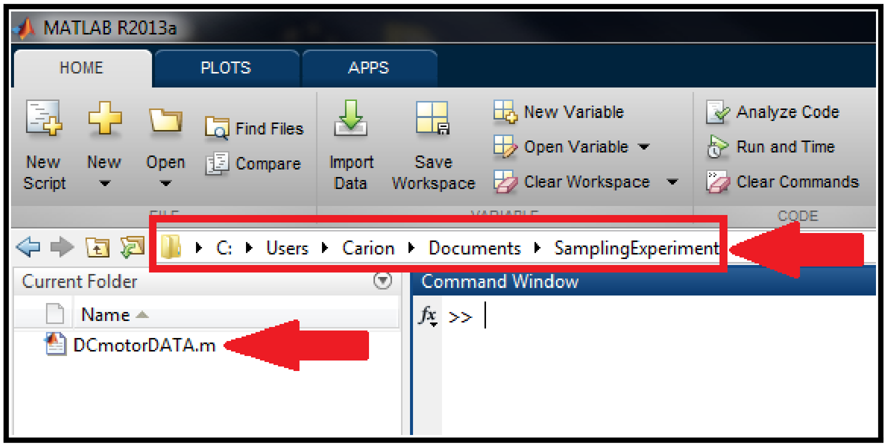


Name	Date modified	Type
L2S5	11/17/2014 3:48 PM	File folder
L2S6	11/17/2014 3:48 PM	File folder
L2S1	11/17/2014 3:48 PM	File folder
L2S2	11/17/2014 3:48 PM	File folder
L2S3	11/17/2014 3:48 PM	File folder
L2S4	11/17/2014 3:48 PM	File folder
SolidworksTutorial	11/18/2014 3:29 PM	File folder
MiKTex	1/13/2015 2:51 PM	File folder
Carion	4/1/2015 10:38 PM	File folder
slprj	5/19/2015 5:30 PM	File folder
SerialTest2_rtt	5/19/2015 5:30 PM	File folder
Visual Studio 2010	6/8/2015 2:37 PM	File folder
MATLAB	6/30/2015 6:54 PM	File folder
SamplingExperiment	6/30/2015 7:56 PM	File folder

Folder: SamplingExperiment

Figure 5.26: Experiment Folder Select

35. Make sure MATLAB's "Current Folder" section shows the folder you selected, and the "DCmotorDATA.m" file is in the directory.



C: ▸ Users ▸ Carion ▸ Documents ▸ SamplingExperiment

DCmotorDATA.m

```
fx >>
```

Figure 5.27: Current MATLAB Directory

36. In the MATLAB window, select New → Simulink Model

37. Choose File → Save As. Set the filename to *Exp2Sampling*, and select Save. "Exp2Sampling.slx" file should now be in MATLAB's "Current Folder" section.

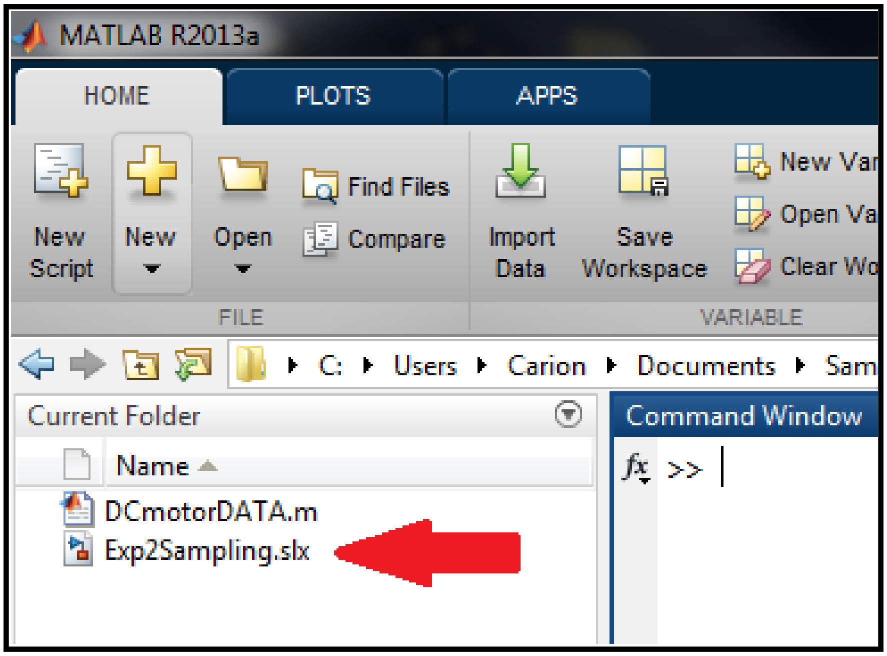


Figure 5.28: Current Folder Sampling File

38. In the Simulink Library Browser add a sine wave to the model. (Simulink → Sources → Sine Wave.) Add a Scope to the model. (Simulink → Sinks → Scope.) Then connect to the two blocks as shown in Figure 5.29

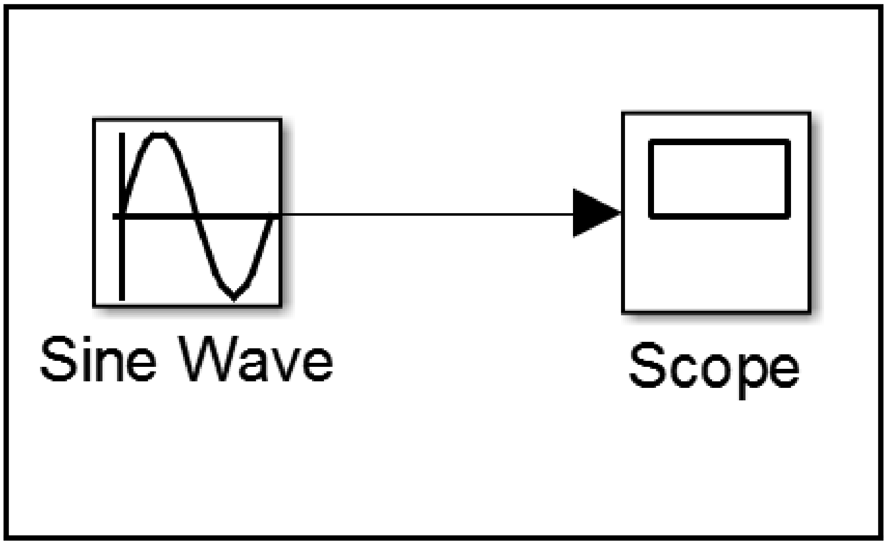
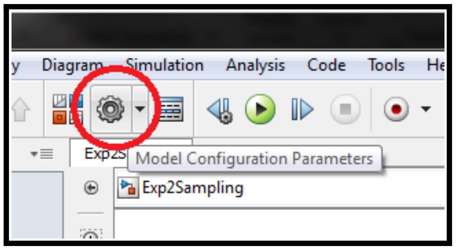


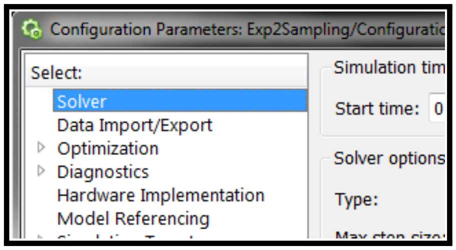
Figure 5.29: Sampling Experiment Simulink Model

39. Double-click the Sine Wave block, and change the “Frequency” to *2*pi*. Select OK.

40. Select Model Configuration Parameters ⚙, and choose Solver from the list on the left.

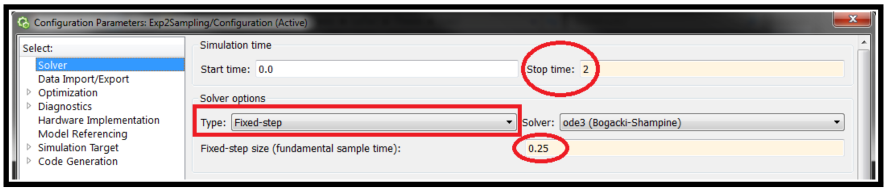


(a) Configuration Parameters



(b) Solver

41. Under Solver options, change "Type" to Fixed-step, then set the "Fixed-step size (fundamental sample time)" to 0.25 and "Stop time:" to 2 seconds. Select OK.



Start time: 0.0
Stop time: 2
Type: Fixed-step
Solver: ode3 (Bogacki-Shampine)
Fixed-step size (fundamental sample time): 0.25

Figure 5.31: Solver Options

42. Double-click the Scope block, and select "Parameters"

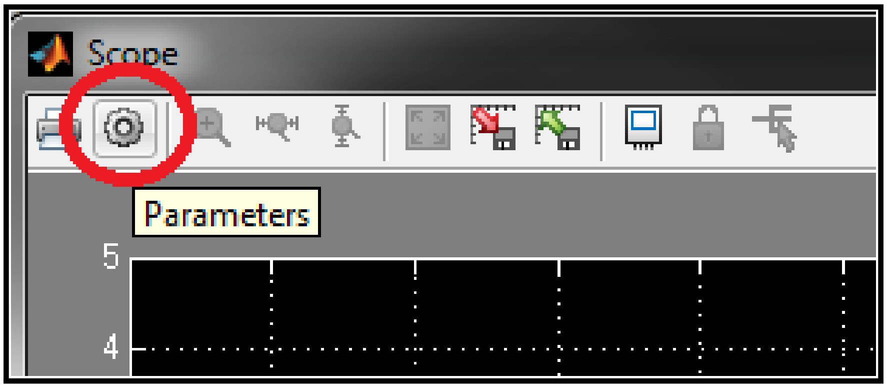
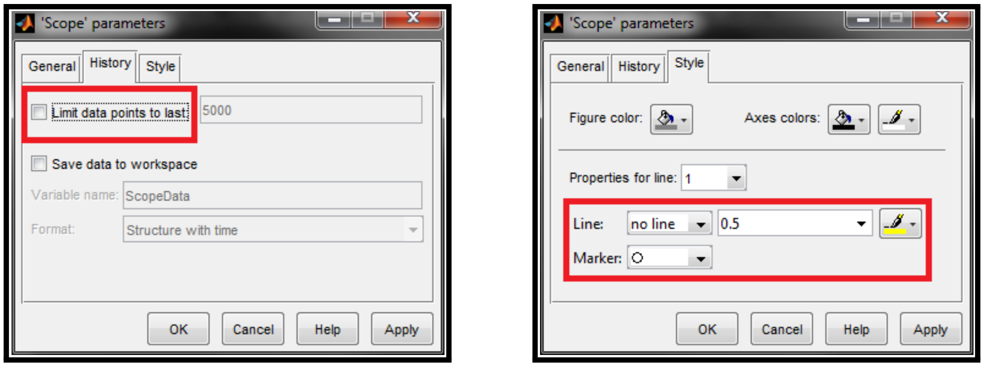


Figure 5.32: Scope Parameters

43. Select the History tab, and deselect the "Limit data points to last:" box. Select Apply at the bottom right. Then, select the Style tab. Change "Line:" to no line, and "Marker" to O. Select Apply then hit OK. Exit out of the Scope.



'Scope' parameters

General | History | Style

☐ Limit data points to last: 5000

☐ Save data to workspace

Variable name: ScopeData

Format: Structure with time

'Scope' parameters

General | History | Style

Figure color: Axes colors:

Properties for line: 1

Line: no line 0.5

Marker: ○

(a) Scope Data Points

(b) Scope Markers

44. In the Sine Wave block, the Amplitude is set to 1, Frequency ω is 2π, and the phase is 0. On paper, sketch what this sine wave will look like for two cycles. How fast would you need to sample it to avoid aliasing?

- ⋆ The sine wave would look like

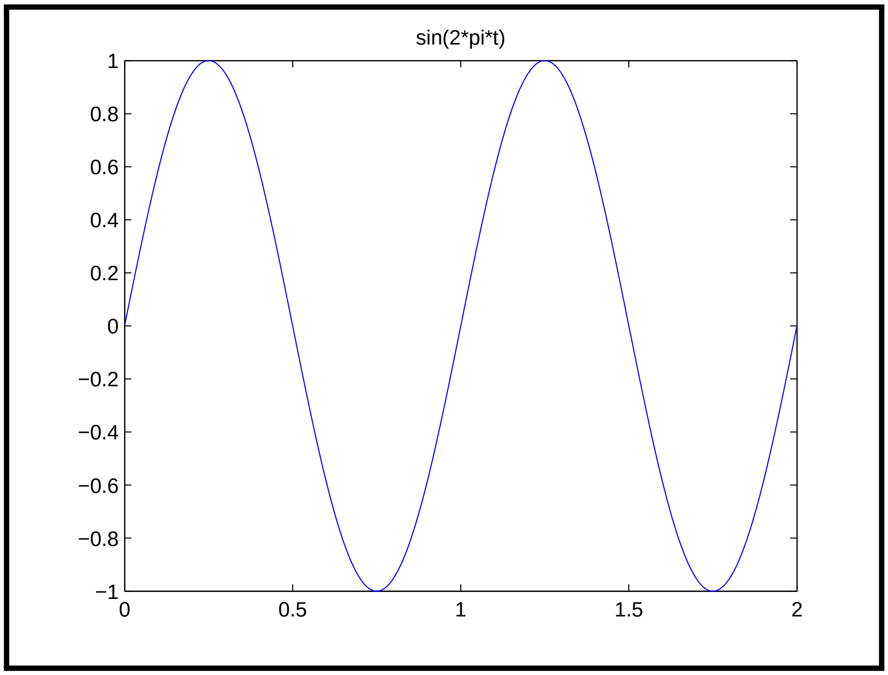


Figure 5.34: Input Sine Wave

- ⋆ Since ω is 2π, f_{max} for this sine wave is just 1 Hz. f_s has to be $> 2f_{max}$, so f_s has to be $>$ 2 Hz to avoid aliasing.

45. Re-sketch the sine wave assuming it is being sampled, and has a sampling frequency of 4 Hz. What is the Sampling Period?

- ⋆ The sine wave would look like

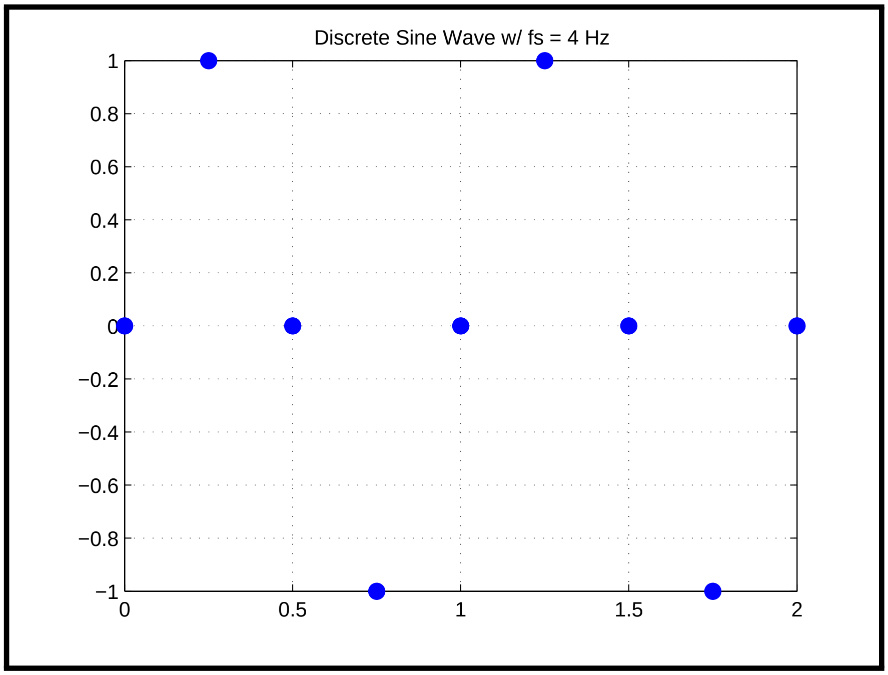


Figure 5.35: Output for 4Hz Sampling

- ⋆ $T_s = 0.25$ seconds.

46. In the Simulink model, select Run ▶.

47. Double-click the Scope. Is this plot that you expected? Why or why not? How does it compare to the sampled sine wave plot you drew?

48. Exit out of the scope, and navigate back to the Model Configuration Parameters ⚙.

49. Change the Fixed-step size (fundamental sample time) to 0.025 and select OK.

50. Run the model, and open the scope.

51. How many points are shown on the plot? (Ignore the point at 2 seconds, because it is the start of the third cycle of the sine wave.) Before you count the points, calculate how many there should be, using the sampling time.

    - ⋆ $T_s = 0.025$, which means $f_s = 40$ Hz (samples per second). 40 samples per second for 2 seconds = 80 points.

52. In Model Configuration Parameters, change the Fixed-step size (fundamental sample time) to 0.00025 and select OK. Run the model, and open the scope.

53. Ignoring the point at 2 seconds, how many points are shown on the plot? How has decreasing the sampling rate changed the outputs thus far?

    ⋆ $T_s = 0.00025$, which means $f_s = 4000$ Hz. There would be 8000 points on the plot.

    ⋆ Each time the sampling rate has been decreased, the number of points has increased. The more points, the closer the sine wave looks to the continuous time signal.

54. In Model Configuration Parameters, change the Fixed-step size (fundamental sample time) to 0.5 seconds and select OK.

55. Sketch what you expect the sampled output sine wave to look like for two cycles.

    ⋆ The sine wave would look like

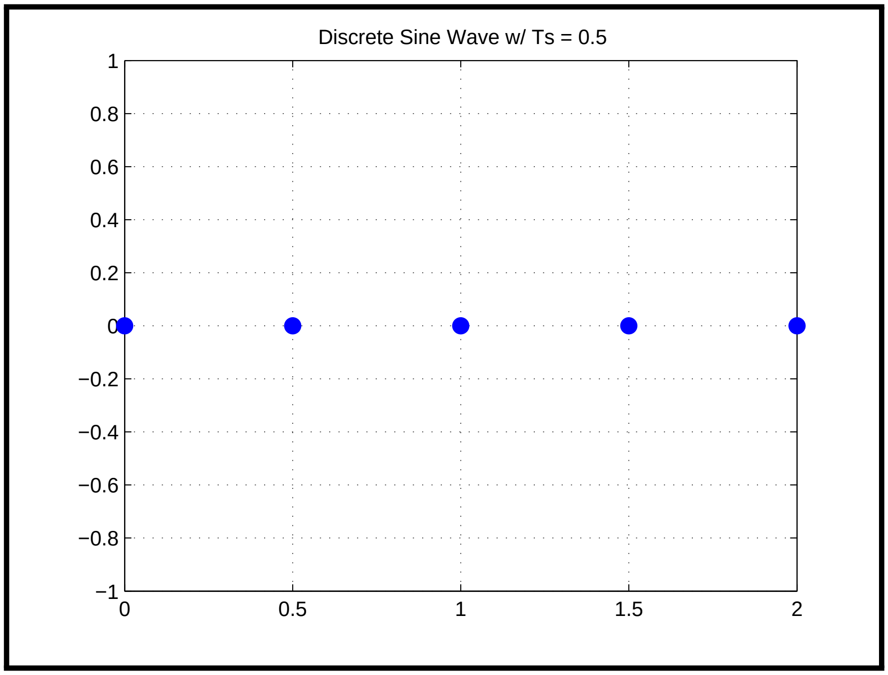


Figure 5.36: Output for 0.5 second Sampling

56. Run the model, and open the scope.

57. Does this agree with your sketch? Would the input sine wave be able to be reconstructed from this output? Why or why not?

- ⋆ No, because the sampling rate is not greater than 0.5 seconds.

58. In Model Configuration Parameters, change the Fixed-step size (fundamental sample time) to 0.8 and Stop time to 10. Select OK.

59. Run the model, and open the scope.

60. What does the frequency of this signal seem to be? Is this what it should be? If not, explain.

    - ⋆ The output sine wave will look like Figure 5.37. Figure 5.38 shows what a $-\sin(0.5\pi t)$ wave looks like. This is similar to the output sine wave if we sample every 0.8 seconds. The output is supposed to be $\sin(2\pi t)$, since we set $f = 1$ Hz. The actual output suggests that $f = 0.25$ Hz. This shows how aliasing affects the output, and will prevent the actual signal from being reconstructed.

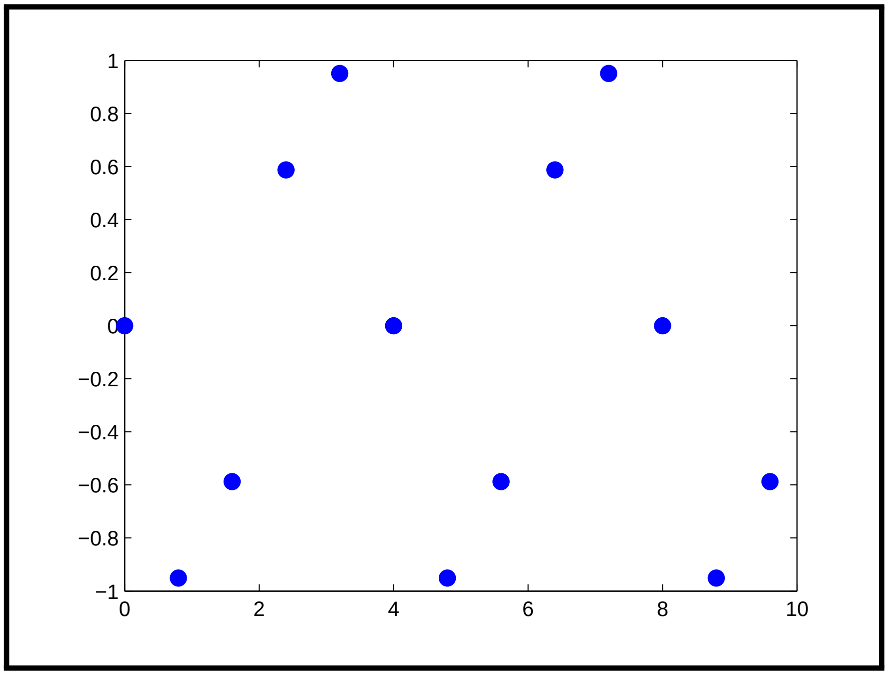


Figure 5.37: Output for 0.8 second Sampling

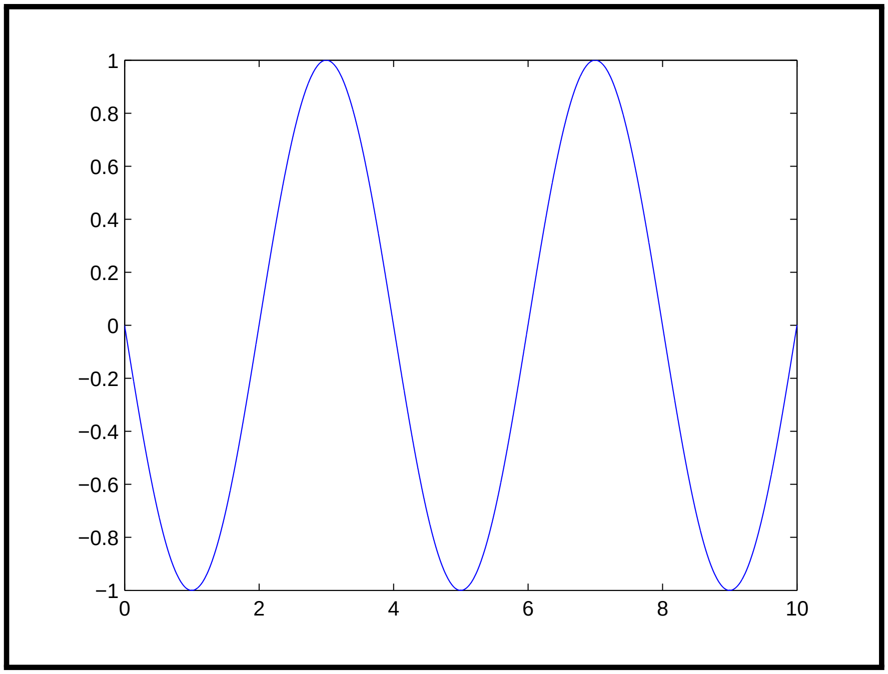


Figure 5.38: $-\sin(0.5\pi t)$

61. Save the model and close it.

5.3.2 Exercise 2: Sampling and Data Acquisition in Normal Mode

Now that the concept of sampling has been introduced, the next task is to sample with the Arduino in Normal Mode. This section will use all of the required hardware, and it is assumed that you know the entire process for running Simulink models on the Arduino using Normal Mode. This process was described in both the *Blinking LED* and *Simple DC Motor* experiments.

62. From the MATLAB window, open a new Simulink Model: New → Simulink Model

63. Save the file as *Sampling_NM*. The final Simulink Model will look like Figure 5.39

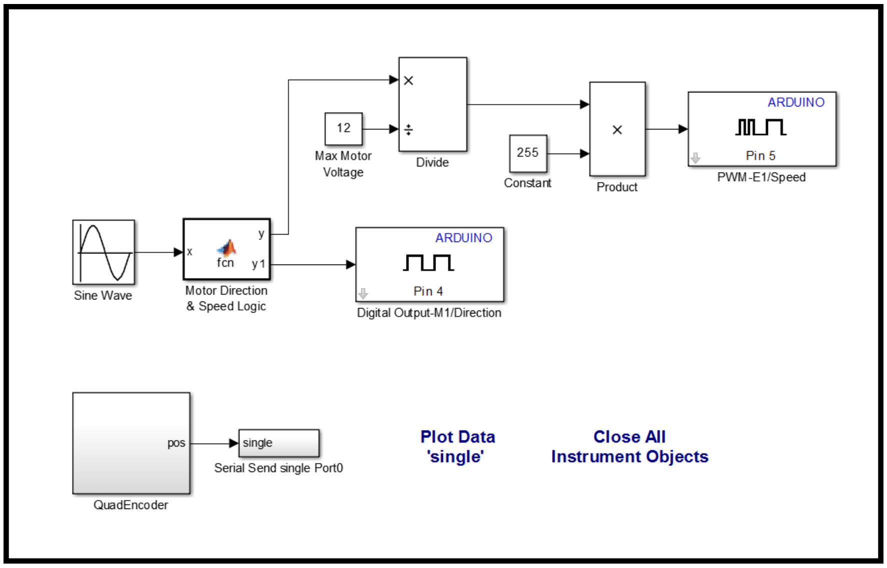
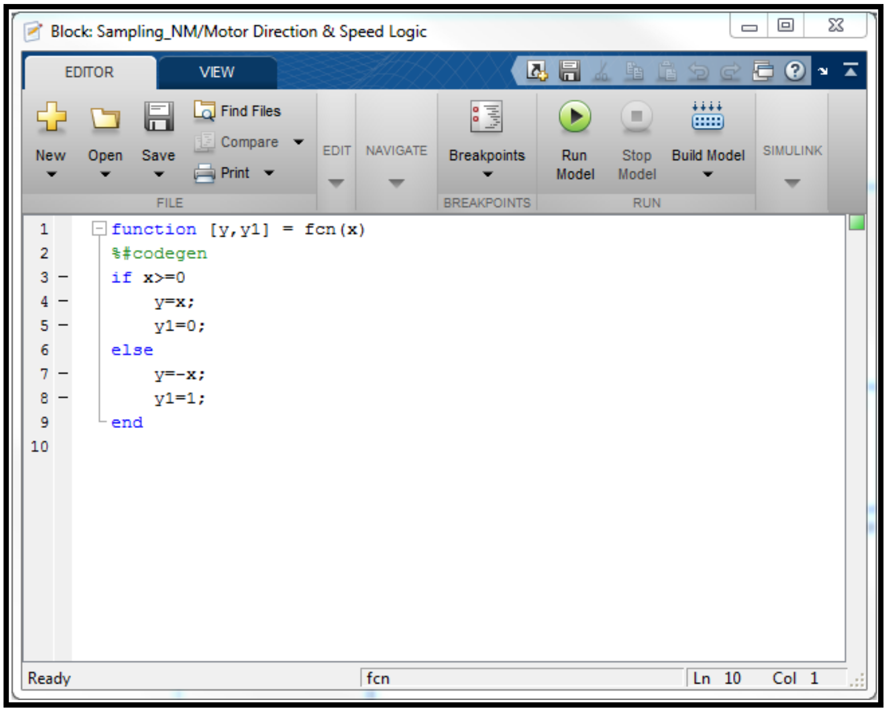


Figure 5.39: Simulink Model for Normal Mode

64. Open the Simulink Library Browser, and add the following blocks to your empty Simulink Model:

    - Sine Wave: Sources → Sine Wave
    - Constant: Sources → Constant
    - MATLAB Function: User-Defined Functions → MATLAB Function
    - Divide: Math Operations → Divide
    - Arduino Digital Output: Simulink Support Package for Arduino Hardware → Digital Output

65. Double-click the MATLAB Function block. The MATLAB editor will now open. Change the code to match that shown in Figure 5.40, then exit out of the editor.



```
function [y,y1] = fcn(x)
%#codegen
if x>=0
    y=x;
    y1=0;
else
    y=-x;
    y1=1;
end
```

Figure 5.40: Simulink MATLAB Function Block Code

66. Double-click the Sine Wave block, set the Sine Type to "Time based", Amplitude to 4, Frequency to *2*pi*, and Sample time to *Ts*. Press OK to exit.

67. Double-click the Constant block and change the Constant value to 12, or to whatever is the voltage of your power supply. Press OK.

68. Double-click the Digital Output block and change the Pin number to 4. Press OK.

69. Connect the current blocks as in Figure 5.41

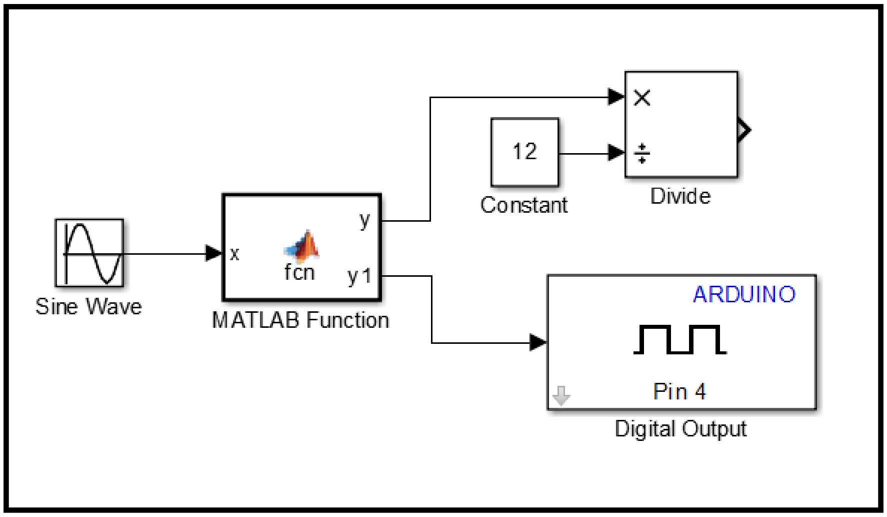


Figure 5.41: Simulink Model Setup - Normal Mode (1)

70. Add the following blocks to the Simulink Model:
    - Constant
    - Product: Math Operations → Product
    - Arduino PWM: Simulink Support Package for Arduino Hardware → PWM
    - Quad Encoder Block: Take Home Labs Arduino Support Package → QuadEncoder
    - Serial Send single: Take Home Labs Arduino Support Package → Serial Send single Port0
71. Change the new Constant block's value to 255, and set the PWM block to Pin 5. Then connect the blocks as in Figure 5.42

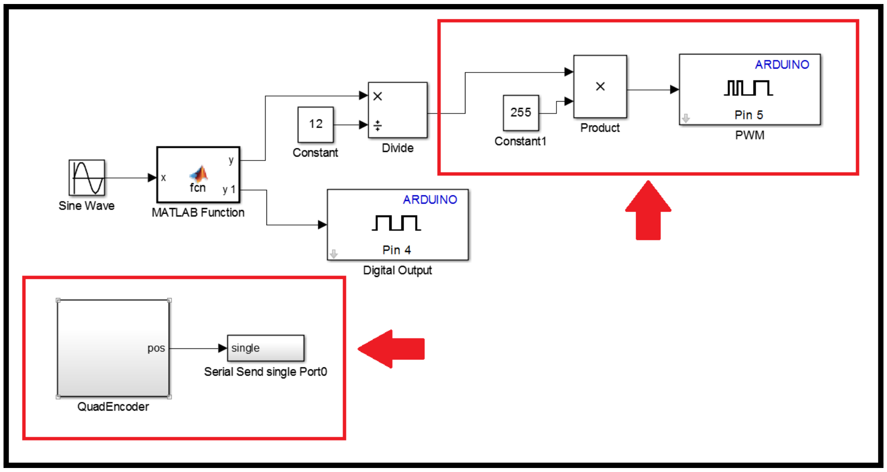


Figure 5.42: Simulink Model Setup - Normal Mode (2)

72. Finally, add the text blocks to the Simulink Model. In the Simulink Library Browser, right-click “Take Home Labs Arduino Support Package”, then select “Open Take Home Labs Arduino Support Package library”. A window similar to that shown in Figure 5.43 should open. Click and drag “Plot Data ’single’” and “Close All Instrument Objects” to the Simulink File. The final Simulink model should now resemble the model shown in Figure 5.39.

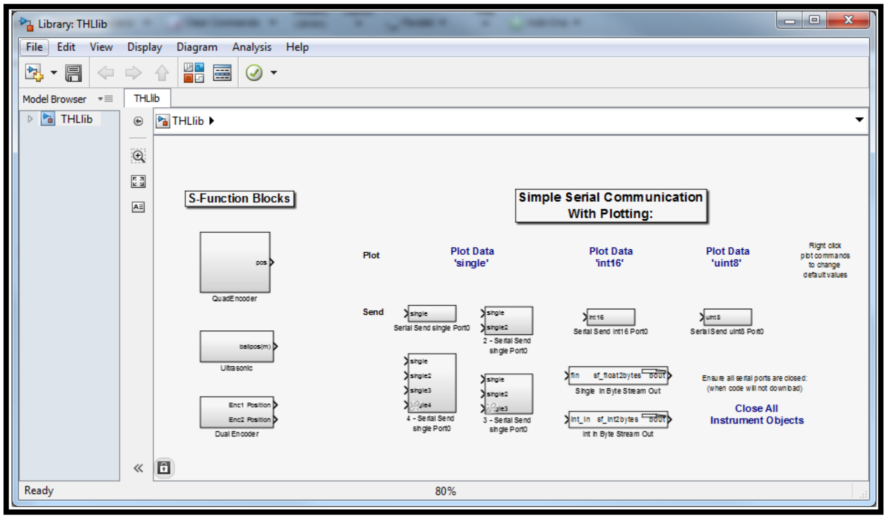
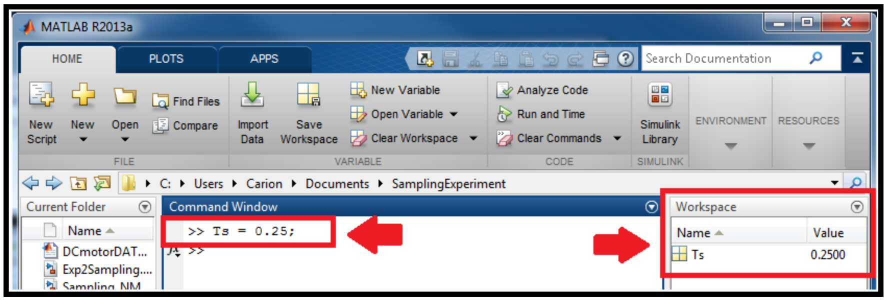


Figure 5.43: Simulink Model Setup - Normal Mode (3)

73. Set up the model to run on the Arduino in Normal Mode. Set Sampling Time to *Ts*. Refer to steps 26-28 of the *Blinking LED* experiment, if you do not remember the setup process. You may also want to review step 29, if you forgot how to run the model.

74. In the MATLAB Command Window, type *Ts = 0.25;*. “Ts” should now be a variable in the MATLAB Workspace.



```
>> Ts = 0.25;
>>
```

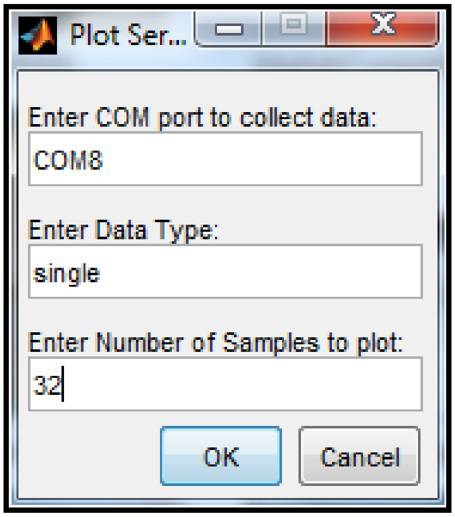
Name	Value
Ts	0.2500

Figure 5.44: Sampling Time - Normal Mode

75. Next, connect the Arduino to the PC, and load the model onto the Arduino. Remember that in Normal Mode, once you download the model to the Arduino, the code will begin running automatically. Therefore, do not connect the power supply just yet. *Note:* If you receive an error stating that Simulink cannot connect to the board, try to disconnect and reconnect the USB cable. Other troubleshooting tips are found in the *Blinking LED* and *Simple DC Motor* experiments.

76. Once the model is successfully downloaded to the Arduino, select the “Plot Data ’single’” text in the Simulink model.

77. A window, “Plot Serial Data”, should open. Change the COM port to the port your Arduino is using, and set the number of samples to plot to 32, as shown in Figure 5.45. Then hit OK.



Plot Ser...

Enter COM port to collect data:

COM8

Enter Data Type:

single

Enter Number of Samples to plot:

32

Figure 5.45: Plot Serial Data

78. A figure should open up that will start plotting the serial data, as shown in Figure 5.47. It may take a few seconds to open.

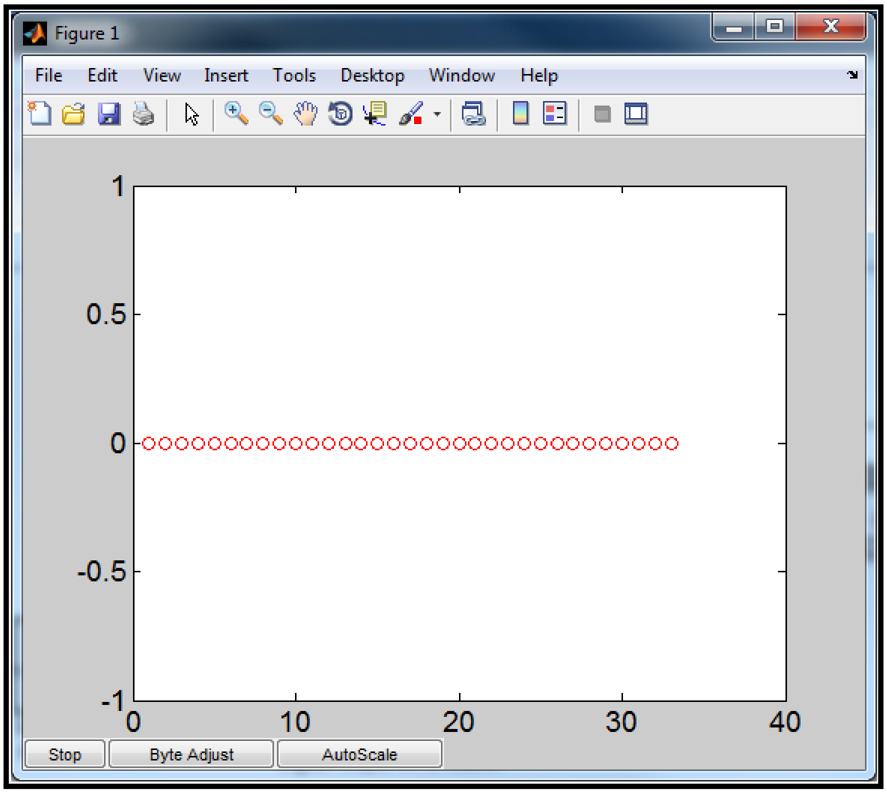


Figure 5.46: Serial Plot Window

79. Before connecting the power, check to see if the serial data that is being captured is in sync. Press the "Autoscale" at the bottom of the plot window, as in

Figure 5.47

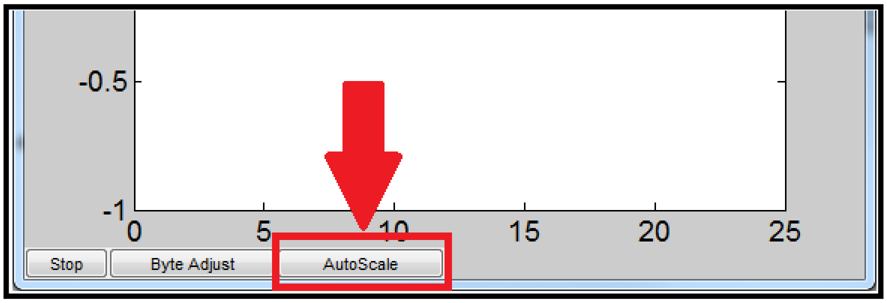


Figure 5.47: Autoscale Serial Plot

80. If the scale on the vertical axis stays -1 to 1, as in Figure 5.48, skip ahead to step 82. If the scale on the vertical axis changes, the plot may be out of sync, proceed to step 81.

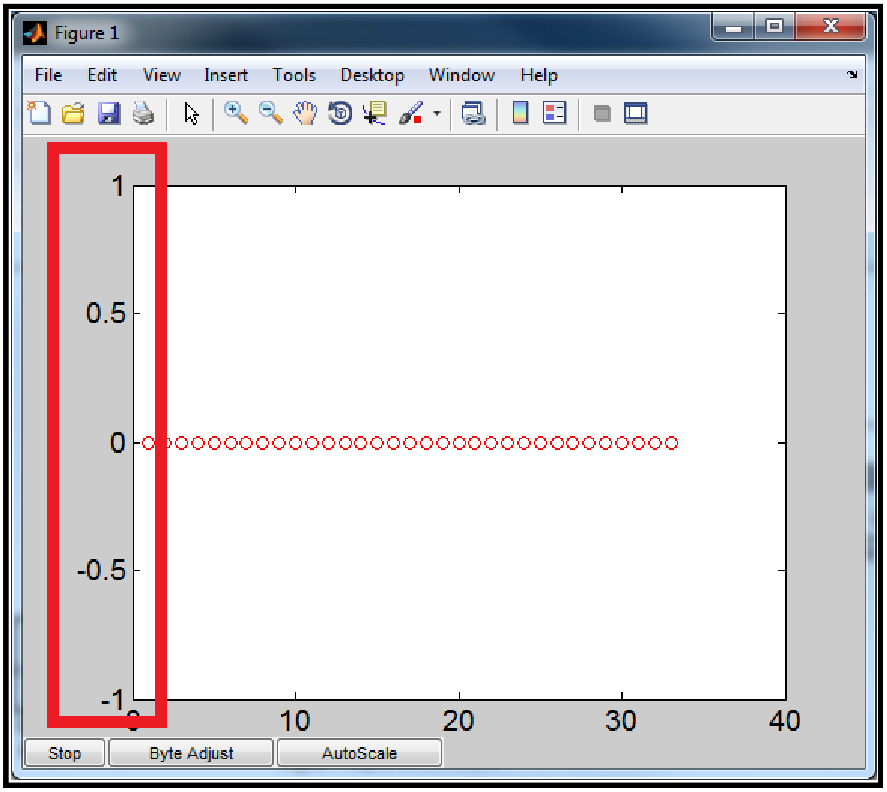
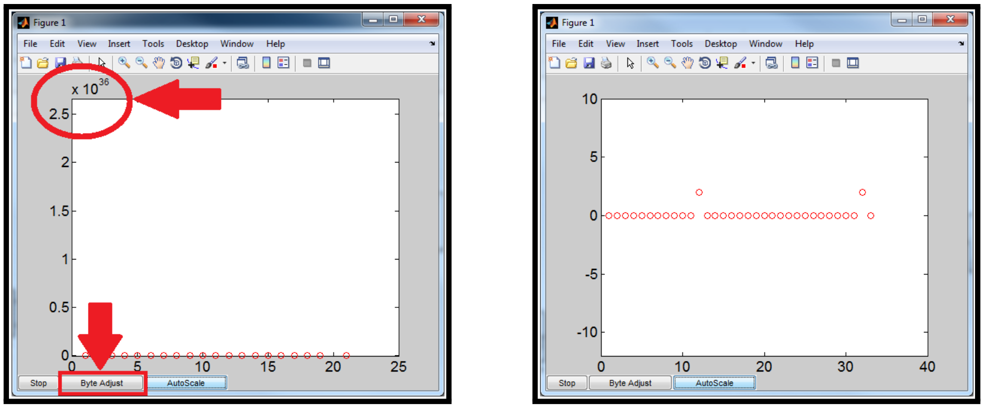


Figure 5.48: Autoscale Test

81. If the plot's y-axis values are very large (as in Figure 5.49a), or the values are changing on the plot, while the load is not moving (as in Figure 5.49b), the



(a) Large Out of Sync Values

(b) Small Out of Sync Values

Figure 5.49: Out of Sync Plot Examples

data is being read incorrectly. There are a few ways to get the data in sync (see the substeps below the figure).

(a) Press the "Byte Adjust" button to skip a byte. If the data was plotting incorrectly, especially if the scale became very large, you will need to wait until the incorrect data is cleared from the plot window because autoscaling will not adjust to the much smaller correct values until those large values are no longer shown on the plot.

(b) Once the incorrect data has passed, press the autoscale button to see if the y-axis scale is now fixed, and the serial data equals zero.

(c) If the data is not fixed, you can repeat steps (a) and (b) until the y-axis scale is no longer very large, or the data on the plot is at zero and not changing. If you have to attempt to byte adjust the data more than 3 or 4 times, press the "Stop" button, and close the plot.

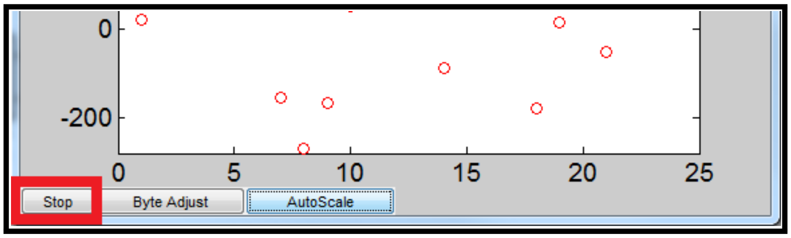


Figure 5.50: Serial Plot Stop Button

(d) Start the serial plotting process over by returning to step 76.

(e) If neither byte adjust or stopping and restarting the plot gets the data in sync, try disconnecting and reconnecting the USB cable, and starting over from step 75.

(f) Once the data is in sync, proceed to step 82.

82. Slightly move the load to the left and right, and then press the "Autoscale" button again. The data in the plot should look similar to Figure 5.51, ensuring the data is in sync.

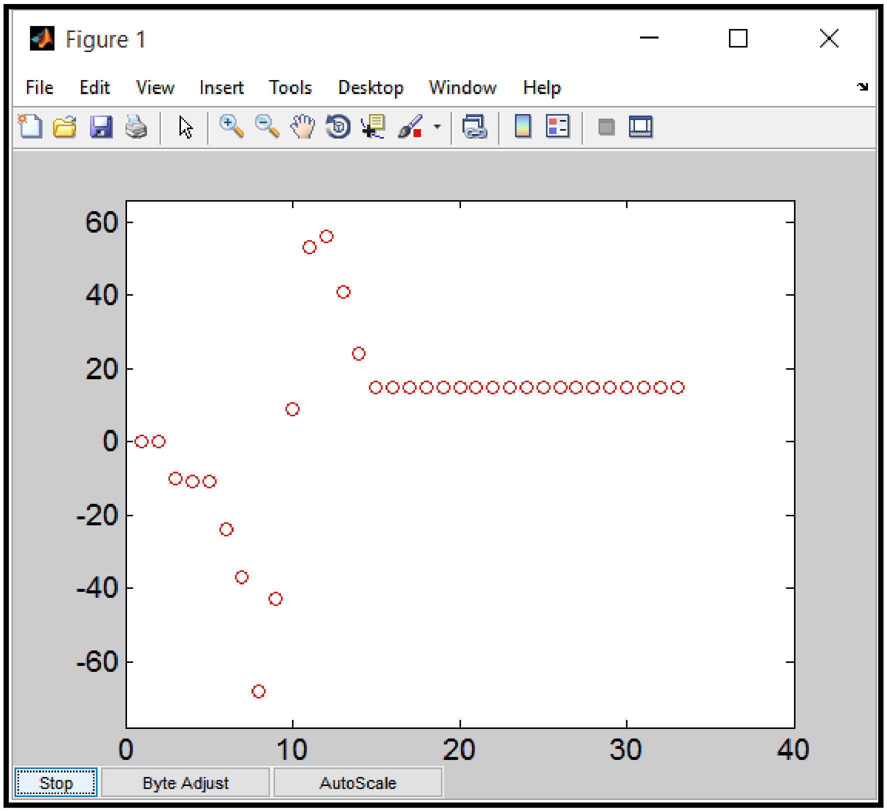
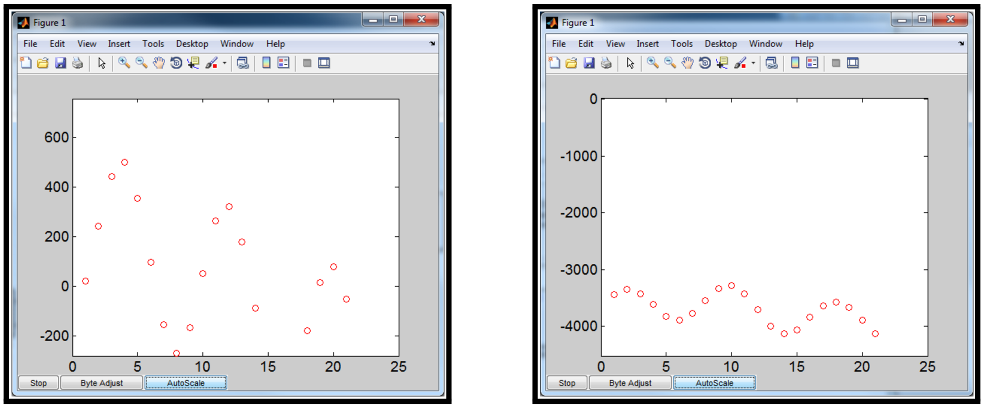


Figure 5.51: Serial Data Sync Test

83. Before plugging in the power, make sure the load is attached to the motor shaft securely, and that your hands and any other objects are away from the spinning load, to avoid potential injuries. Then manually move the load position back as close to 0 as you can.

84. Plug in the power supply. ***Caution:*** The motor will start to spin the attached load when you connect the power supply.

85. The plot should now show a sine wave similar to Figure 5.52a or Figure 5.52b. You may need to press Autoscale while the load is spinning to fit the plot axes to the data. The output plot you want should have seven to eight cycles of the sine wave, since there are 32 samples being plotted. When the plot shows seven to eight cycles of the sine wave, press the "Stop" button located at the bottom left of the plot window.



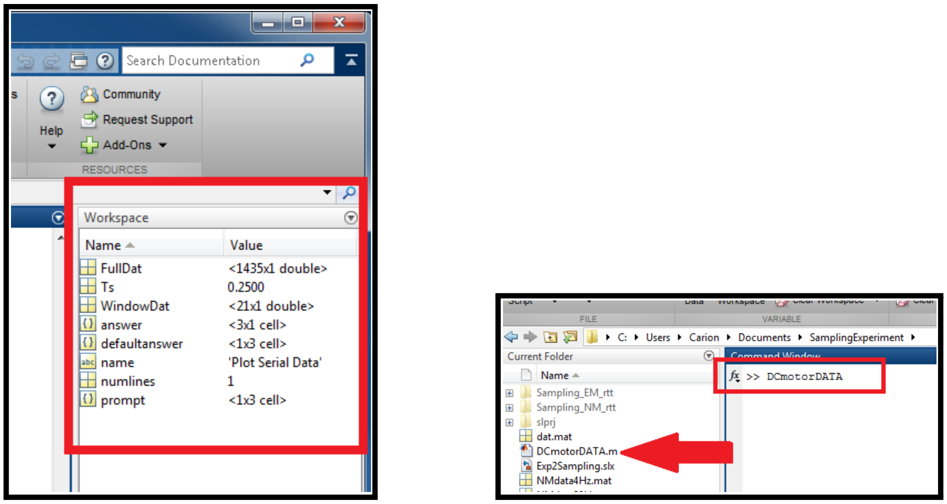
(a) Initial Correct Data

(b) Byte Adjusted Correct Data

Figure 5.52: Correct Data Serial Plots

86. Disconnect the power supply from the Arduino.

87. Once the plot is stopped, the variables shown in Figure 5.53a should now be in the MATLAB workspace. The "WindowData" variable is the data from the serial plot window. Type *DCMotorDATA* in the MATLAB command window as shown in Figure 5.53b. This will run the "DCMotorDATA.m" file that was inside the "SamplingExperiment" folder that was downloaded in the Software Setup section. This file will produce two figures, the motor position in counts, which is the "WindowData" variable, and the velocity of the motor in counts/sec. Both figures plot the data versus time, and not versus samples, as it did in the serial plot window. This will allow you to check if the frequency of the sine wave is correct.



Name	Value
FullDat	<1435x1 double>
Ts	0.2500
WindowDat	<21x1 double>
answer	<3x1 cell>
defaultanswer	<1x3 cell>
name	'Plot Serial Data'
numlines	1
prompt	<1x3 cell>

```
>> DCmotorDATA
```

(a) Workspace Data from Plot Serial

(b) Running the Plot File

Figure 5.53: Handling the Serial Plot Data

88. Save all your figures for your lab report.

89. How can the velocity plot be created from the position data? What would you say the frequency of the sine wave is based on the position and velocity plots? Explain.

   ⋆ The velocity data is found by taking the position at the next time step and subtracting the position at the current time, and then dividing by the sampling time. $v(t) = \frac{y(t+1)-y(t)}{T_s}$.

   ⋆ Example figures are shown in Figure 5.54a and Figure 5.54b. The output sine wave frequency is 1 Hz, since it takes one second to complete a cycle of the sine wave.

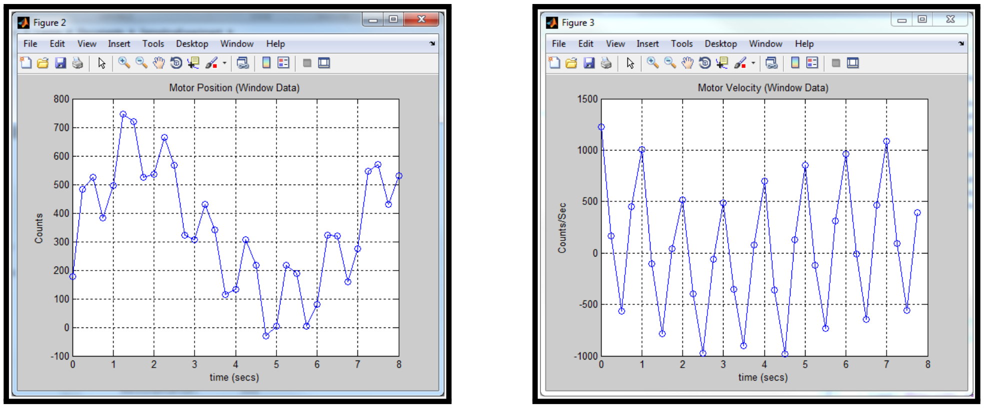


Figure 2
File Edit View Insert Tools Desktop Window Help
Motor Position (Window Data)
Counts
800
700
600
500
400
300
200
100
0
-100
0 1 2 3 4 5 6 7 8
time (secs)
Figure 3
File Edit View Insert Tools Desktop Window Help
Motor Velocity (Window Data)
Counts/Sec
1500
1000
500
0
-500
-1000
0 1 2 3 4 5 6 7 8
time (secs)

(a) Normal Mode Position

(b) Normal Mode Velocity

Figure 5.54: Example Plots for 4Hz Sampling

90. After you have saved your figures, in the MATLAB command window, type *close all* and press enter to close all figures. Type *clear all* and press enter to clear the workspace of all variables. Then type *clc* and press enter to clear the command window.

91. Type *Ts = 0.05;* in the MATLAB command window. Then go back to the simulink file and download the model to the Arduino.

92. Once the model is successfully downloaded to the board, press the "Serial Plot 'single'" text in the model as you did before. Enter your correct COM port, and this time plot 80 samples. Press "OK".

93. Connect the power supply to the Arduino.

94. Get the correct data, a sine wave, to show in the Serial Plot Window for three to four cycles, and press "Stop" to save it to the workspace. An example plot is shown in Figure 5.55.

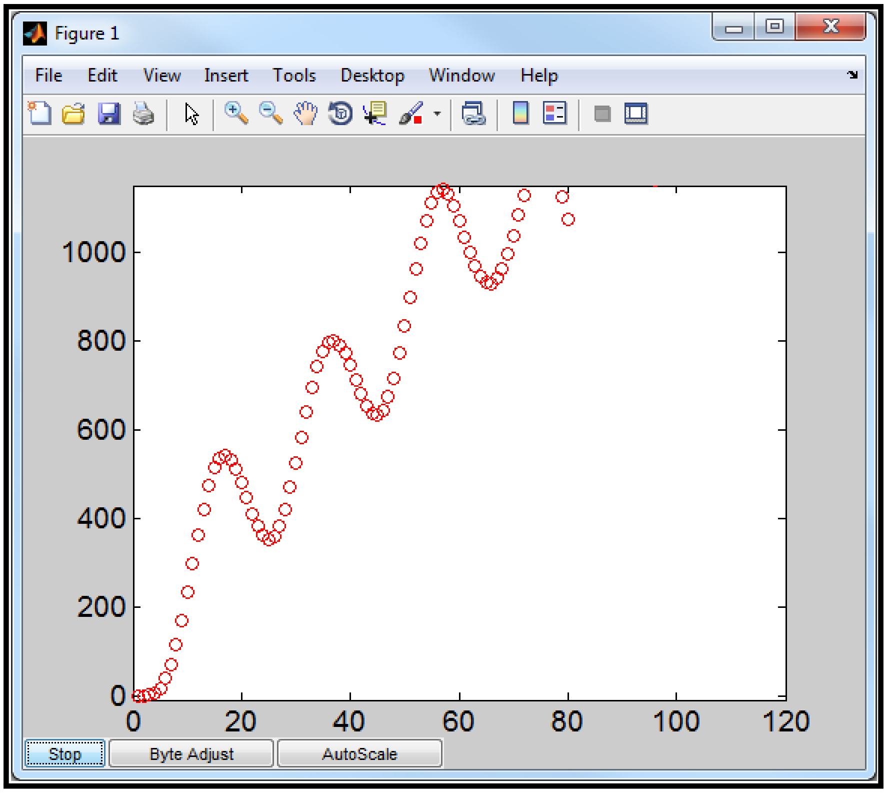


Figure 5.55: Example Plot for 20Hz Sampling

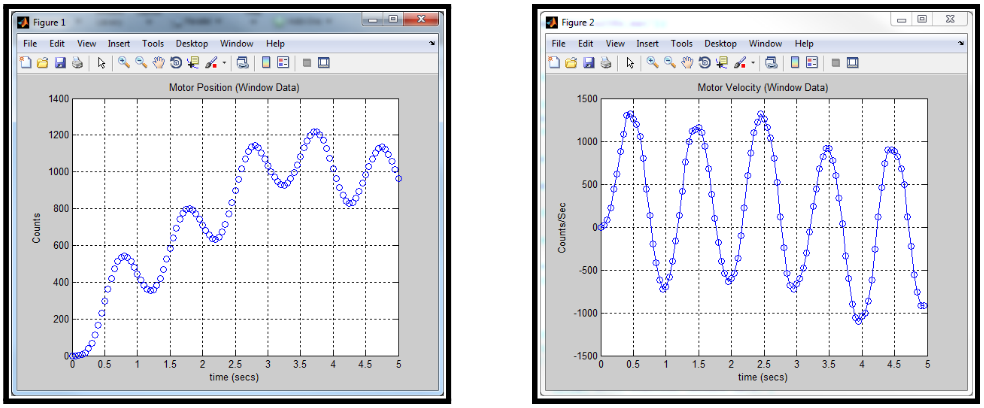
95. Disconnect the power supply from the Arduino.

96. Type *DCMotorDATA* in the MATLAB command window to run the .m file to plot the position and velocity. Save the figures for your lab report.

97. What is the frequency of the sine wave, based on the position and velocity plots? Explain. Discuss the difference between the results for 20 Hz and the results for 4 Hz.

    ⋆ Example figures are shown in Figure 5.56a and Figure 5.56b. The output sine wave frequency is 1 Hz, since it takes one second to complete a cycle of the sine wave.

    ⋆ The 20 Hz results show much smoother sine waves than the 4 Hz results.



(a) Normal Mode Position

(b) Normal Mode Velocity

Figure 5.56: Example Plots for 20Hz Sampling

98. In the MATLAB command window, type *close all* and press enter. Type *clear all* and press enter. Then type *clc* and press enter.

99. Type *Ts = 0.01;* in the MATLAB command window. Then go back to the simulink file and download the model to the Arduino.

100. Once the model has been downloaded to the board, press the "Serial Plot 'single'" text in the model as you did before. Enter your correct COM port, and this time plot 400 samples. Press "OK".

101. Connect the power supply to the Arduino.

102. Get the correct data, a sine wave, to show in the Serial Plot Window for three to four cycles, and press "Stop" to save it to the workspace. An example plot is shown in Figure 5.57.

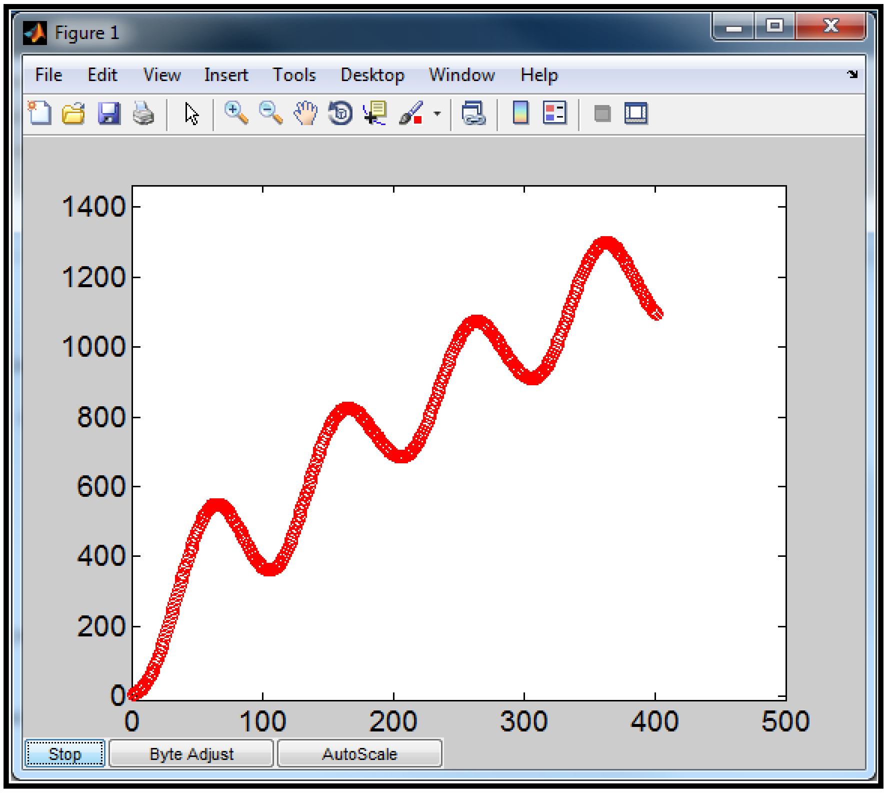


Figure 5.57: Example Plot for 100Hz Sampling

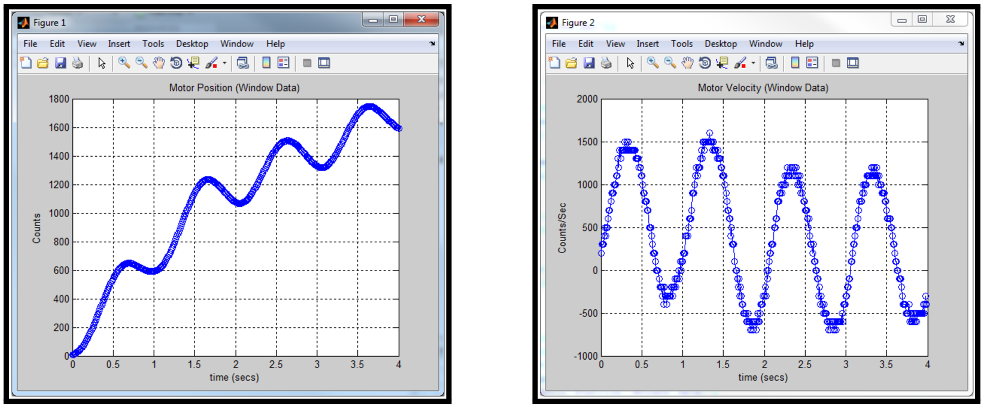
103. Disconnect the power supply from the Arduino.

104. Type *DCMotorDATA* in the MATLAB command window to run the .m file to plot the position and velocity. Save the figures for your lab report.

105. What is the frequency of the sine wave, based on the position and velocity plots? Explain. Discuss the difference between the results for 100 Hz and the previous two cases. Which rate would you use for this signal? Why?

   - ⋆ Example figures are shown in Figure 5.58a and Figure 5.58b. The output sine wave frequency is 1 Hz, since it takes one second to complete a cycle of the sine wave.

   - ⋆ The 100 Hz results contain more samples than the first two cases. This makes the plots look smoother and more like a continuous signal. The 100 Hz rate provides the best results of the three so it should be used.



(a) Normal Mode Position

(b) Normal Mode Velocity

Figure 5.58: Example Plots for 100Hz Sampling

106. In the MATLAB command window, type *close all* and press enter. Type *clear all* and press enter. Then type *clc* and press enter.

107. Now type *Ts = 0.0075;* in the MATLAB command window. Then go back to the simulink file and download the model to the Arduino.

108. Once the model has been downloaded to the board, press the "Serial Plot 'single'" text in the model as you did before. Enter your correct COM port, and this time plot 534 samples. Press "OK".

109. Connect the power supply to the Arduino.

110. Get the correct data, to show in the Serial Plot Window for three to four cycles, and press "Stop" to save it to the workspace. An example plot is shown in Figure 5.59.

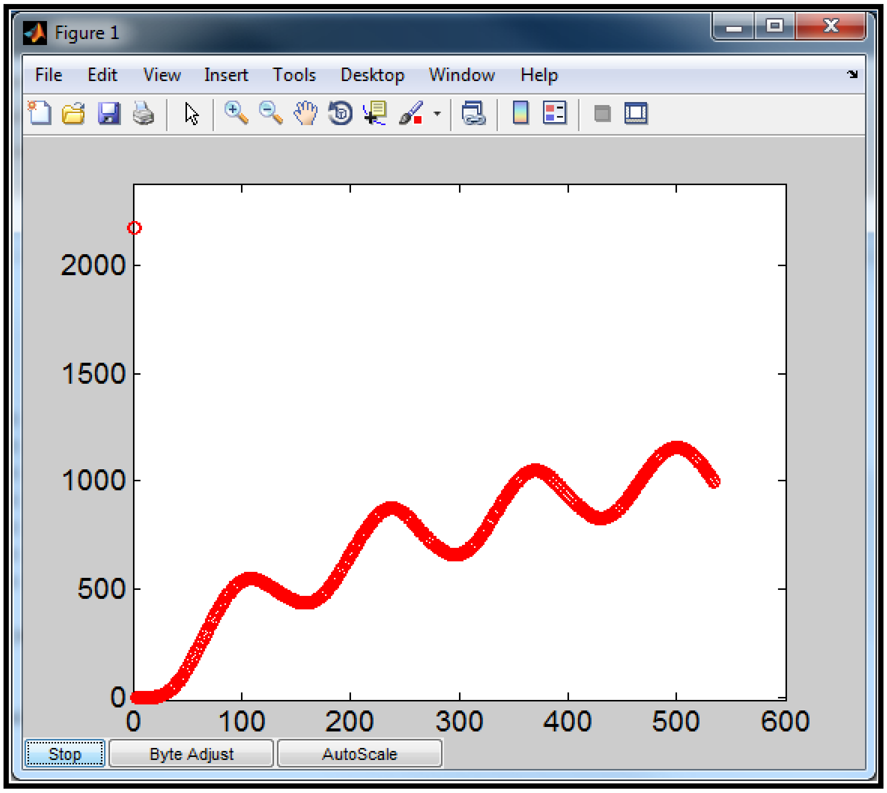


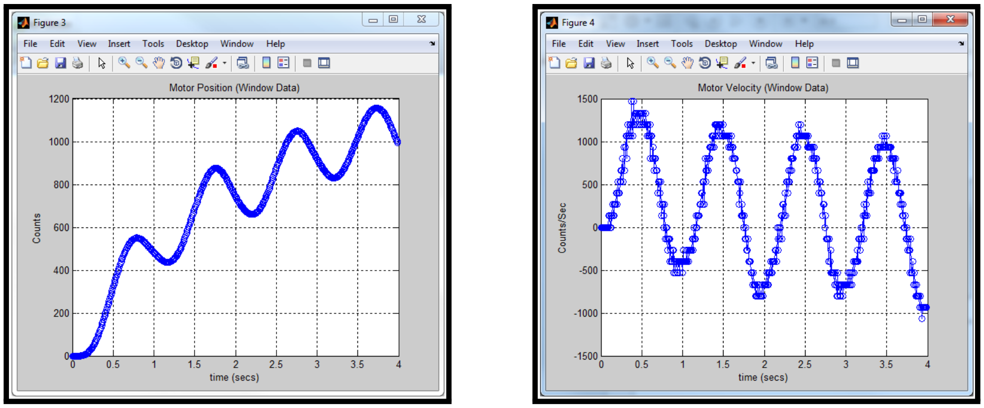
Figure 5.59: Example Plot for 133Hz Sampling

111. Disconnect the power supply from the Arduino.

112. Type *DCMotorDATA* in the MATLAB command window to run the .m file to plot the position and velocity. Save the figures for your lab report.

113. What is the frequency of the sine wave, based on the position and velocity plots? Explain.

   ⋆ Example figures are shown in Figure 5.60a and Figure 5.60b. The output sine wave frequency is 1 Hz, since it takes one second to complete a cycle of the sine wave.



(a) Normal Mode Position

(b) Normal Mode Velocity

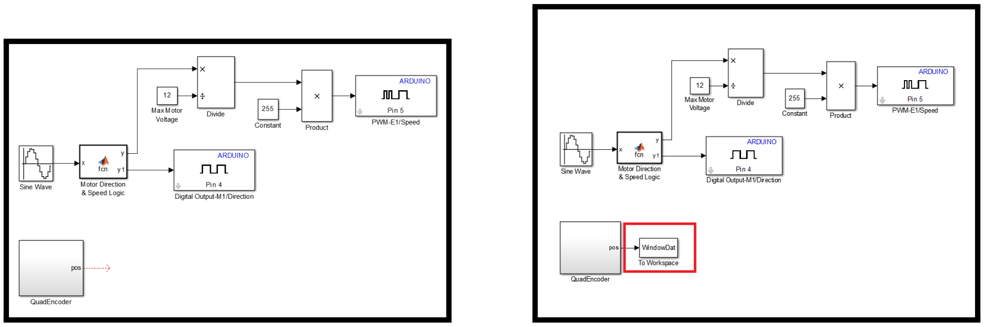
Figure 5.60: Example Plots for 133Hz Sampling

114. Save the Simulink file, and disconnect the USB cable from the PC.

5.3.3 Exercise 3: Sampling and Data Acquisition in External Mode

115. The Simulink model that will be used in External Mode is very similar to the model used in the Normal Mode section. So start with the final Normal Mode model by opening the *Sampling_NM.slx* that was used in Exercise 2.

116. Save the file as a different name, *Sampling_EX*, then delete the “Serial Send single Port0” block, the “Plot Data 'single'” text, and the “Close All Instrument Objects” text from the model, as shown in Figure 5.61a. Then add a “To Workspace” block from the Library Browser(Simulink → Sinks → To Workspace) and connect it to the output of the “QuadEncoder” block, as shown in Figure 5.61b.

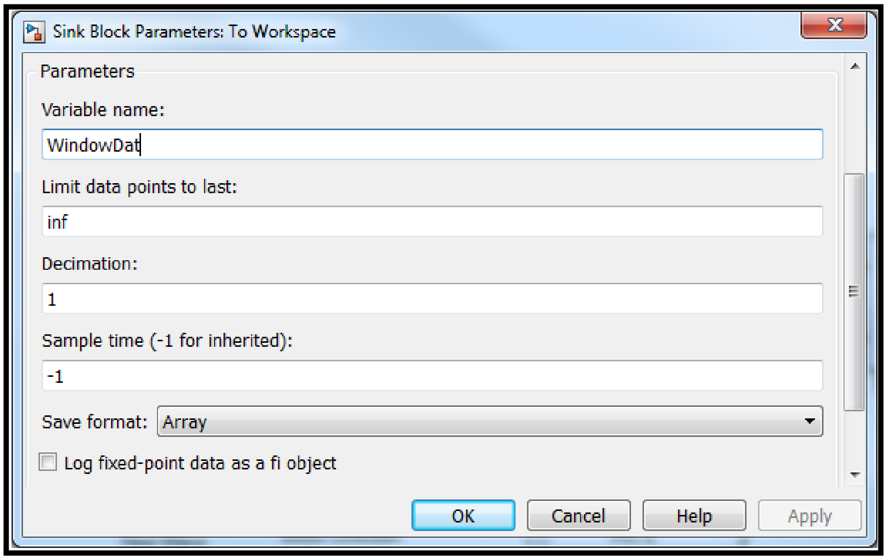


(a) Model Starting Place

(b) Final Model

Figure 5.61: External Mode Model Setup

117. Double-click the “To Workspace” block. In the block parameters, change “Variable name:” to *WindowDat*, and for “Save format:” choose “Array”. Press “OK” to exit the block parameters.



Sink Block Parameters: To Workspace

Parameters

Variable name:
WindowDat

Limit data points to last:
inf

Decimation:
1

Sample time (-1 for inherited):
-1

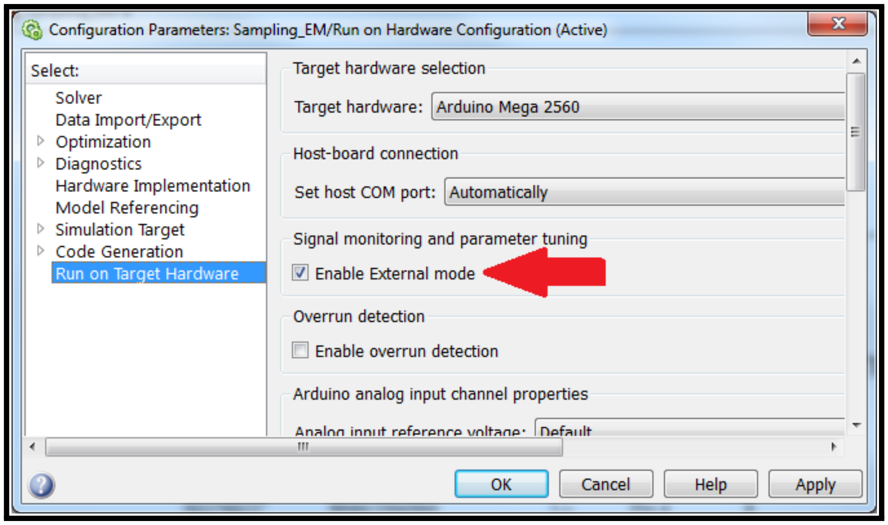
Save format: Array

□ Log fixed-point data as a fi object

Figure 5.62: Workspace Block Parameters

118. Change the Simulink model configuration parameters to run in External Mode. Since we created this file from the model in Exercise 2 that is configured to run in Normal Mode, just change the following:

- Open the “Model Configuration Parameters” ⚙. In the “Solver” section, leave the sampling time equal to “Ts”, as it was in Exercise 2.
- In the “Run on Target Hardware” section, select the box next to “Enable External mode”, as shown in Figure 5.63, and press “OK” to close the window.



Target hardware selection

Target hardware: Arduino Mega 2560

Host-board connection

Set host COM port: Automatically

Signal monitoring and parameter tuning

☑ Enable External mode

Overrun detection

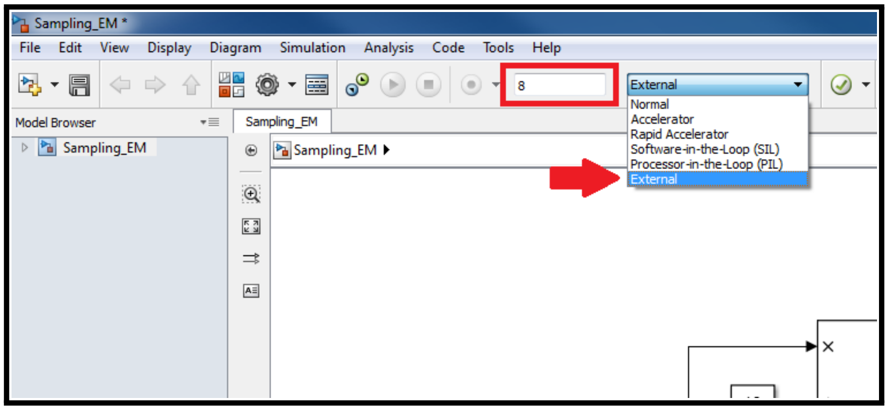
□ Enable overrun detection

Arduino analog input channel properties

Analog input reference voltage: Default

Figure 5.63: External Mode Parameters

- Set the stop time to 8 seconds, and change the simulation mode to "External", as shown in Figure 5.64.



8

External

Figure 5.64: Model Run Settings

119. In the MATLAB command window, type *Ts = 0.25;* and press enter.

120. Connect the USB cable to the PC and Arduino, and load the model onto the Arduino.

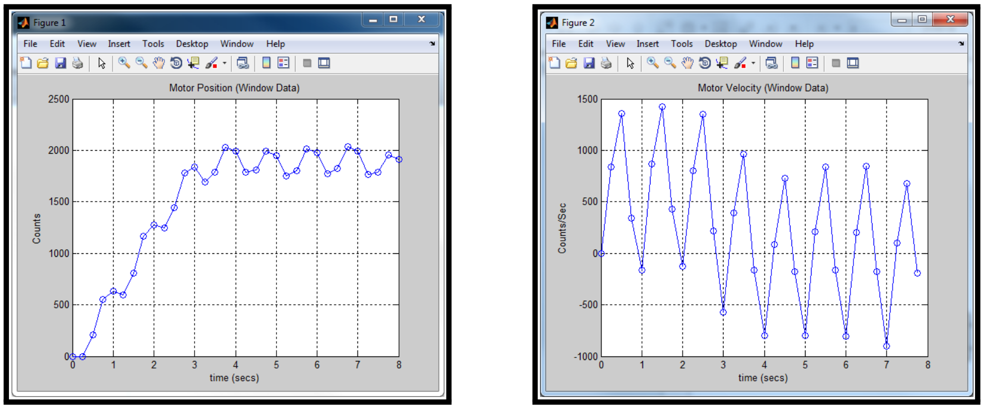
121. Plug in the power supply, press to connect to the Arduino. ***Caution:*** The motor may begin to rotate the load when you connect to the Arduino. If the model does not start running, you will also need to press Run ▶.

122. After 8 seconds the model will stop running and disconnect from the Arduino automatically. The position data will be stored into the "WindowData" variable

in the workspace. Type *DCMotorData* in the MATLAB command window to generate the position and velocity plots versus time. Save the plots for your lab report.

123. What is the output frequency for both the position and velocity plots? Explain any differences between these plots and the plots you found for 4 Hz sampling using Normal Mode. *Remark:* Check for any differences in the output magnitudes, and check for large jumps between time steps.

    ⋆ Example figures are shown in Figure 5.65a and Figure 5.65b. The output sine wave frequency is 1 Hz, since it takes one second to complete a cycle of the sine wave.



(a) External Mode Position

(b) External Mode Velocity

Figure 5.65: Example Plots for 4Hz Sampling

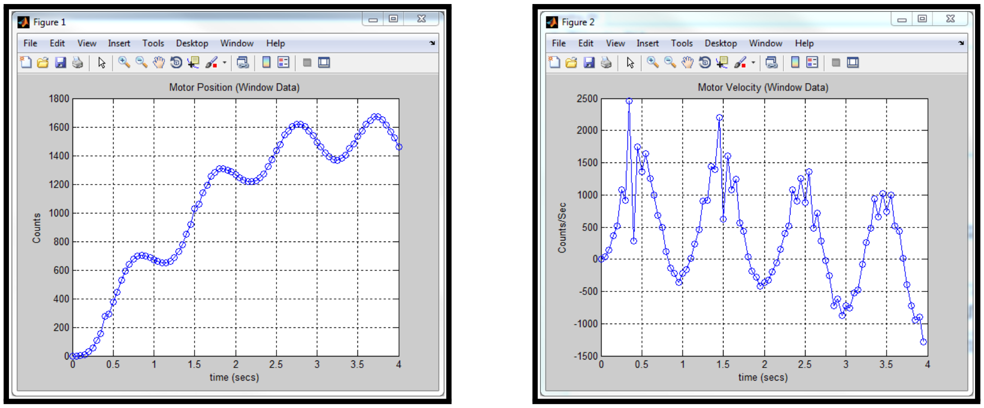
124. In the MATLAB command window, type *close all* and press enter. Type *clear all* and press enter. Then type *clc* and press enter.

125. Type *Ts = 0.05;* in the MATLAB command window. Then go back to the simulink file and change the stop time to 4 seconds. Download the model to the Arduino. Press the button to connect to the board, and then press the run button.

126. After 4 seconds the model will stop running and disconnect from the Arduino automatically. The position data will be stored into the "WindowData" variable in the workspace. Type *DCMotorData* in the MATLAB command window to generate the position and velocity plots versus time. Save the plots for your lab report.

127. What is the output frequency for both the position and velocity plots? Explain any differences between these plots and the plots you found for 20 Hz sampling using Normal Mode. *Remark:* Check for any differences in the output magnitudes, and check for large jumps between time steps.

   ⋆ Example figures are shown in Figure 5.66a and Figure 5.66b. It is difficult to see anything wrong in the position plot, so the frequency would seem to be 1 Hz as it should be. The velocity plot shows the correct time between cycles, which is 1 second, but there are some fairly big jumps, which should not happen if the sampling time is constant. In the Normal Mode plots for sampling at 20 Hz, the magnitude of the velocity output was about 2000 counts/sec. The most extreme case for External Mode has a magnitude of about 2900 counts/sec. Another noticeable issue for the External Mode velocity plot are the large jumps between time steps. An example of this is shown in Figure 5.67, where there was a jump from about 2500 counts/sec to 300 counts/sec between 0.35 and 0.4 seconds. The most logical explanation would be that the sampling time is not consistent. If this is true, this could be a major issue because it will give incorrect data. The position data does not seem to be affected much, but slightly wrong position values could lead to large errors in the calculated velocity.



(a) External Mode Position

(b) External Mode Velocity

Figure 5.66: Example Plots for 20Hz Sampling

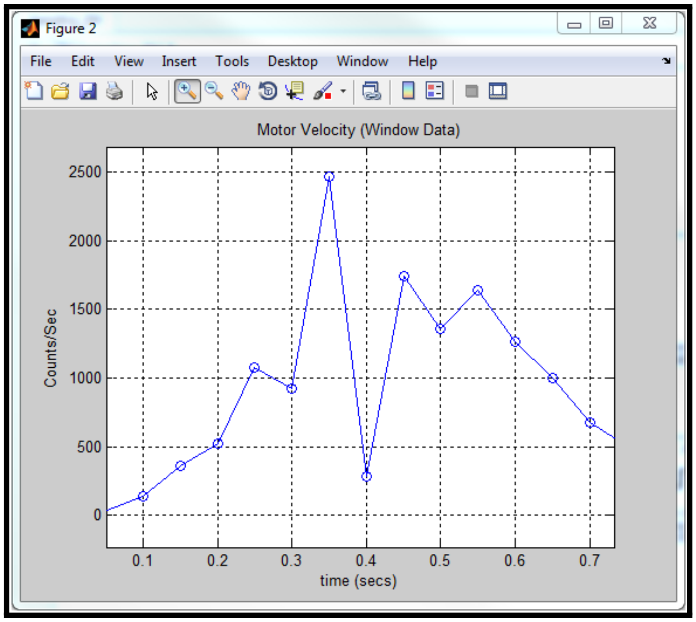


Figure 5.67: EM Velocity Plot Large Jump

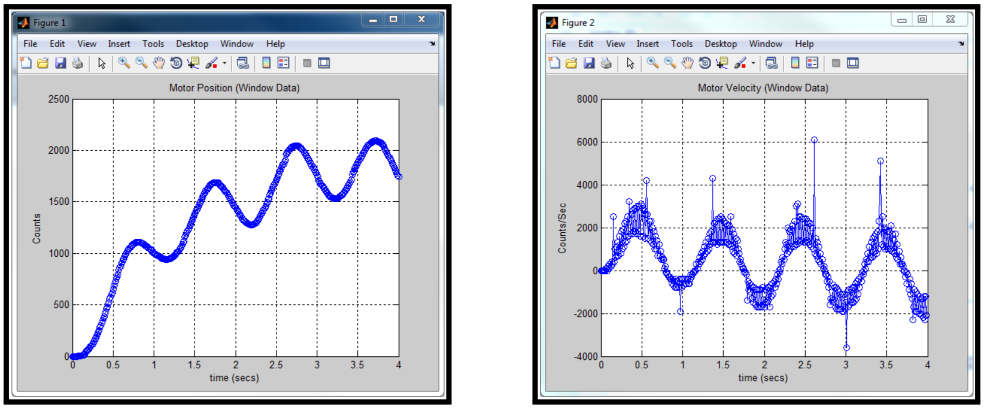
128. In the MATLAB command window, type *close all* and press enter. Type *clear all* and press enter. Then type *clc* and press enter.

129. Type *Ts = 0.01;* in the MATLAB command window. Then go back to the Simulink file and download the model to the Arduino. Press the button to connect to the board, and then press the run button.

130. After 4 seconds the model will stop running and disconnect from the Arduino automatically. The position data will be stored into the "WindowData" variable in the workspace. Type *DCMotorData* in the MATLAB command window to generate the position and velocity plots versus time. Save the plots for your lab report.

131. What is the output frequency for both the position and velocity plots? Explain any differences between these plots and the plots you found for 100 Hz sampling using Normal Mode. *Remark:* Check for any differences in the output magnitudes, and check for large jumps between time steps.

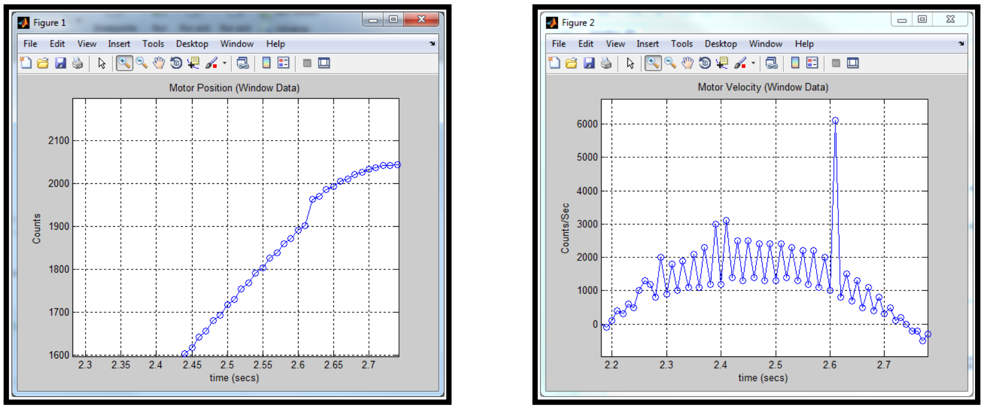
    ⋆ Example figures are shown in Figure 5.68a and Figure 5.68b. Again it is difficult to see anything wrong in the position plot because there are so many samples, so the frequency would seem to be 1 Hz as it should be. The velocity plot shows the correct time between cycles, which is 1 second, but the big jumps and the peak to peak differences look even worse than at 20 Hz. Figure 5.69a and Figure 5.69b show one of the large jumps in position and velocity. In comparison, the largest jump in the position and velocity plots for the Normal Mode are shown in Figure 5.70a and Figure 5.70b. Even if the very large jump is ignored in the velocity plot for External Mode, the average difference between time steps is still about 1000 counts/sec, which is much higher than those shown in the Normal Mode velocity plot.



(a) External Mode Position

(b) External Mode Velocity

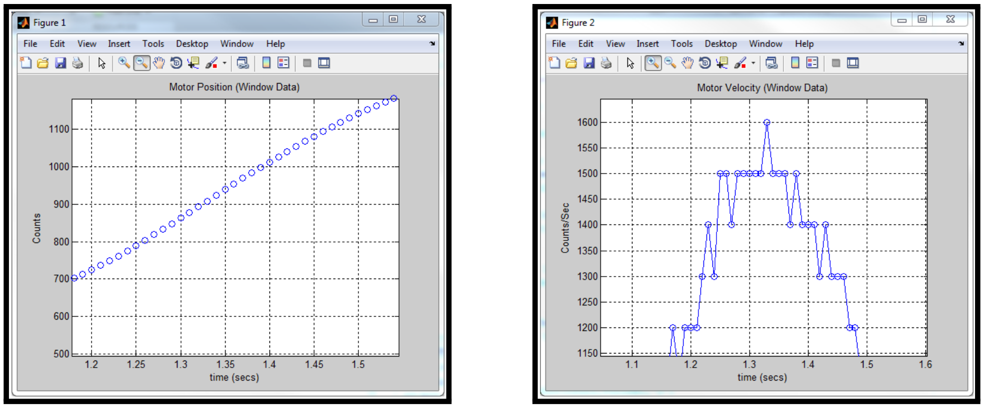
Figure 5.68: Example Plots for 100Hz Sampling



(a) External Mode Position

(b) External Mode Velocity

Figure 5.69: External Mode Large Jumps



(a) Normal Mode Position

(b) Normal Mode Velocity

Figure 5.70: Jumps in Normal Mode Plots

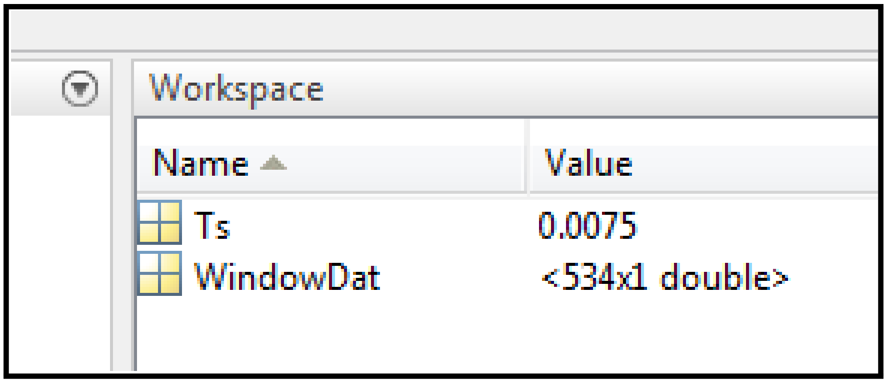
132. In the MATLAB command window, type *close all* and press enter. Type *clear all* and press enter. Then type *clc* and press enter.

133. Type *Ts = 0.0075;* in the MATLAB command window. Then go back to the Simulink file and download the model to the Arduino.

134. If you have a stopwatch on your cellphone, or any other way to record time, get it ready to time this experiment. If you do not have one, just count to yourself and watch the simulation time at the bottom of the Simulink model, as shown in Figure 5.70a. The simulation time will not appear until the model starts to run.

135. Connect to the board. If you have a stopwatch press start at the same time you press the run button, and if you do not have a stopwatch just begin counting when you press the run button. In addition, watch the motor load, and measure how long it takes to complete one cycle. Does it take one second?

136. How long did it take the experiment to finish? In the MATLAB command window, type *Ts = (the time you calculated)/(the number of data points);.* The number of data points can be found from the size of the WindowData variable in the MATLAB workspace. For example, in Figure 5.71 the number of samples would be 534, and say it took my experiment 7 seconds to finish. In the MATLAB command window I would type *Ts = 7/534;* to adjust the sampling time based on how long the experiment ran and how many data points were recorded.



Workspace

Name	Value
Ts	0.0075
WindowDat	<534x1 double>

Figure 5.71: Example Output Data Size

137. Type *DCMotorData* in the MATLAB command window to generate the position and velocity plots versus time. Save the plots for your lab report.

138. What is the output frequency for both the position and velocity plots? Is this what you expect? Explain.

    ⋆ The output frequency should not be 1 Hz anymore. The expected number of output samples, when sampling every 0.0075 seconds for 4 seconds, is 534. Even though the experiment will run longer than 4 seconds, the output is usually still going to be 534 data points. Therefore the real sampling time will not be 0.0075. When the output plots are generated based on the real sampling time, the frequency will be smaller. A typical experiment resulted in a .5 Hz output sine wave, so the cycles of the sine wave took 2 seconds to complete instead of 1 second.

139. Disconnect the power supply and USB cable.

140. Discuss the sampling accuracies for Normal and External Mode. Compare their ease of use for collecting data. Overall, which method do you think would make the best choice for running experiments on the Arduino with Simulink?

    ⋆ Normal Mode can accurately sample at least as fast as 133 Hz. External Mode sampled poorly as early as 20 Hz. It was difficult to tell if External Mode had sampling issues at 4 Hz, but additional sampling times could be explored by the students to see if problems occur at rates between 4 Hz and 20 Hz. From my own tests, External Mode does sample poorly for frequencies lower than 20 Hz. Normal Mode does have its limitations also.

Since the data is being transmitting through the serial port, the sampling rate is limited by the baud rate of the serial port. The default baud rate is only 9600. Faster baud rates can be used so that you can sample faster, but the probability of producing errors will increase.

- ⋆ External Mode is much easier than Normal Mode for collecting data. External Mode only requires sending the data to a scope in Simulink, or sending it to the workspace as a variable. In contrast, Normal Mode required downloading additional custom Simulink blocks, and writing a script to open the serial port and plot the data. Getting the right data to plot is also a hassle, since the data can be out of sync and require skipping bytes until it starts receiving it correctly.
- ⋆ Overall, Normal Mode is really the only reliable option if an experiment requires accurate data, even at slow sampling rates. However, External Mode is useful when you need to test sensors or when data accuracy is not that important.

5.4 Conclusion/Student Feedback

This experiment was focused around the concept of sampling, and sampling with the Arduino using Normal Mode and External Mode. The process for capturing data in both modes was compared as well. The experiment proves that Normal Mode performs accurate sampling, but it is somewhat complex for collecting data. The current data acquisition method in Normal Mode also limits how fast you can run experiments, because the output data is transmitted over a serial port. External Mode proved to be much easier for acquiring data, but performed poorly when sampling, even at slow speeds. If an experiment requires accurate sampling, then Normal Mode is by far the best option. External Mode would be best suited for testing and calibrating sensors before they are used within an experiment.

CHAPTER 6

OPEN LOOP FREQUENCY RESPONSE

6.1 Objective

This experiment will demonstrate the importance of frequency response by finding the open loop frequency response of a DC motor with an attached load. The open loop frequency response will be found by applying a sinusoidal voltage, for multiple frequencies, into the motor. For each frequency, the motor's output position and velocity will be recorded. The magnitude of the output position and velocity waveforms will be compared to the magnitude of the input voltage. Magnitude bode plots for both output position and velocity versus input voltage will be generated. The motor's first order transfer function will be estimated from the experimental velocity vs input bode plot. Bode plots for the estimated transfer functions will be generated and compared to the experimental bode plots.

6.2 Setup

6.2.1 Required Materials

Hardware

The majority of the hardware materials required for this experiment are the same materials used in the *Sampling and Data Acquisition* experiment. Listed below are additional materials that may be needed if you wish to vary the mass of the load. A list of the materials that were used in the *Sampling and Data Acquisition* experiment will also be listed below for reference. The new materials are:

- Pennies - Minimum: 2 Maximum: 10
- Scotch tape
- Small 3-D Printed Load

Previous materials from the *Sampling and Data Acquisition* experiment:

- DC Motor with Encoder
- Arduino Mega 2560 Rev3
- Motor Shield (DFRobot L298P)
- 12V DC Power Supply (12VDC 3A Wall Adapter Power Supply)
- USB cable (Standard A to B plug)
- Female Barrel Jack (2.1mm x 5.5mm Female CCTV Power Jack Adapter)
- 8 wires
- 3D Printed Base
- 3D Printed Motor Clamp
- 2 #4-40 1/2" screws
- 2 #4-40 1/4" screws
- Small Flat screwdriver
- Small Phillips screwdriver
- Sticky Tack
- Sandpaper or Sanding Sponge (Optional)

<u>Software</u>

- MATLAB/Simulink 2013a or later
  - ⋆ *The steps and images related to MATLAB/Simulink for this experiment were created using MATLAB/Simulink 2013a. Therefore some steps and images may be a little different if you are not using this version. If you are in fact using a different version, make sure you know the steps for running models onto the Arduino for your version of Simulink.*
- MATLAB files
- 3-D Printer files

Prerequisite Experiments

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

6.2.2 Software Setup

The necessary software files that are needed for this experiment can be downloaded from the Take Home Labs webpage. One method for downloading the files is shown in the steps below.

1. Open your internet browser and navigate to `csl.okstate.edu`
2. On the left side of the Homepage, select "Courses"
3. In the middle section of the Courses page select "System Dynamics"
4. In the middle section of the System Dynamics page find and select *Open Loop Frequency Response.*
5. On the Open Loop Frequency Response page select "Software/Code" in the rightmost section. A zipfile named *Experiment_OLFR* should download.
6. Right-click the file and choose "Extract All...", or any other method of extracting files on your PC.
7. Extract this folder somewhere convenient, and remember the location. This will be the folder where all of the files and plots created for this experiment are saved.
8. Navigate back to the Open Loop Frequency Response page, and select "3-D Printer Files" in the rightmost section.
9. Select the Open Loop Frequency Response link to download the files. Open and 3-D print both of the ".STL" files. You will print the required load and gear insert. The gear insert is the same insert used for the load you printed in the *Intro to 3-D Printing* experiment, and the load you will print is a smaller version of the load printed in that experiment. Use the steps discussed in that experiment to connect the two parts after printing them.

6.2.3 Hardware Setup

The majority of the hardware for this experiment was used in the *Sampling and Data Acquisition* experiment. If your hardware is no longer set up, review the "Hardware Setup" section of the *Sampling and Data Acquisition* experiment. To vary the load's mass by adding pennies, follow the steps below.

1. Place the load on a flat surface with the penny holes facing up, as in Figure 6.1.

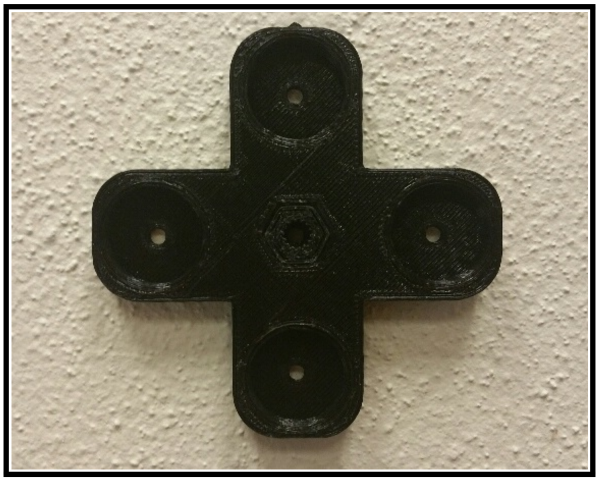


Figure 6.1: Load with no Pennies

2. Place sticky tack in one of the holes, as in Figure 6.2. The amount of sticky tack that should be used depends on the number of pennies that will be inserted into the hole. The more pennies, the less sticky tacky is needed. The size of the sticky tack shown in Figure 6.2 is a good size if inserting two pennies in a hole.

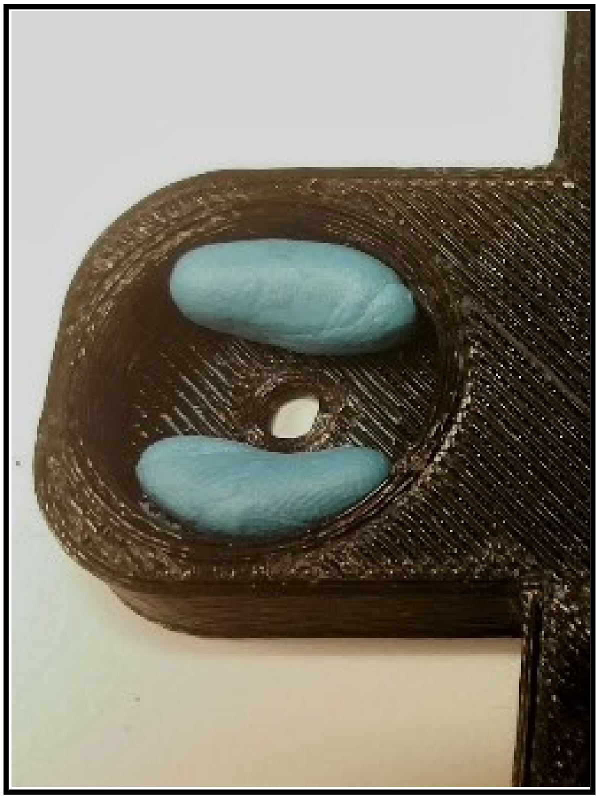


Figure 6.2: Sticky Tack Placement

3. Place a penny on top of the sticky tack that is in the hole, as in Figure 6.3. If this is the only penny that will be inserted, skip the next step.

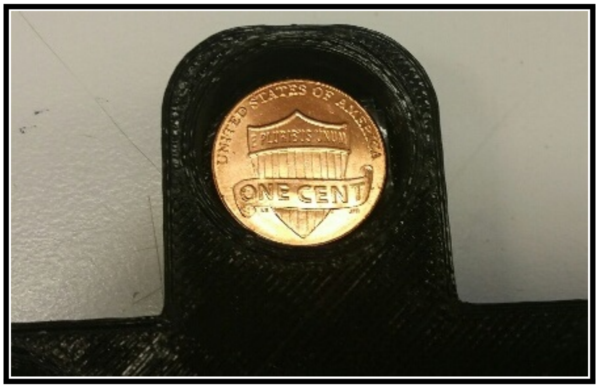


Figure 6.3: Penny Hole Placement

4. Place a small piece of sticky tack on the penny, as in Figure 6.4. Then place another penny in the hole on top of the current one. Repeat this step until you have your desired amount of pennies in the hole. Do not place sticky tack on top of the last penny you add.

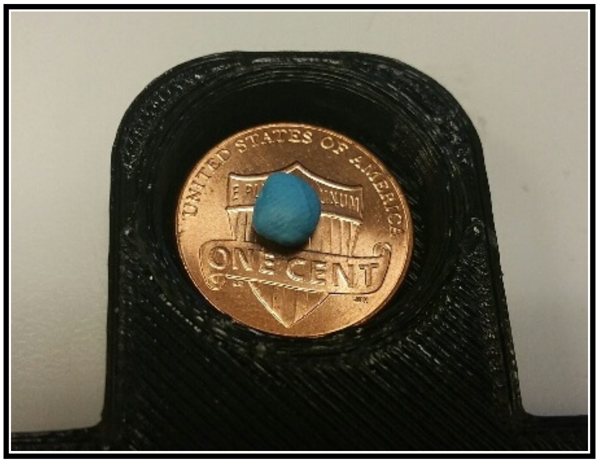


Figure 6.4: Sticky Tack on Penny

5. Press down on the pennies until the top penny sits flush with the face of the load. Tape over the pennies and around the outside of the load a few times, as in Figure 6.5. This will help prevent the pennies from moving during the experiment.

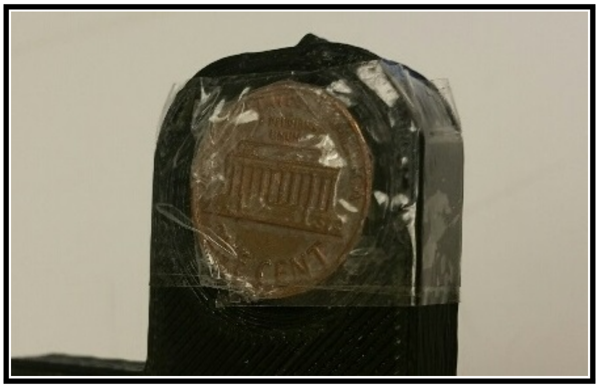


Figure 6.5: Taping the Pennies

6. Repeat the steps of this section for the opposite penny slot. If desired, also repeat the steps for the other two penny slots. With pennies in every slot, the final load should resemble the load in Figure 6.6.

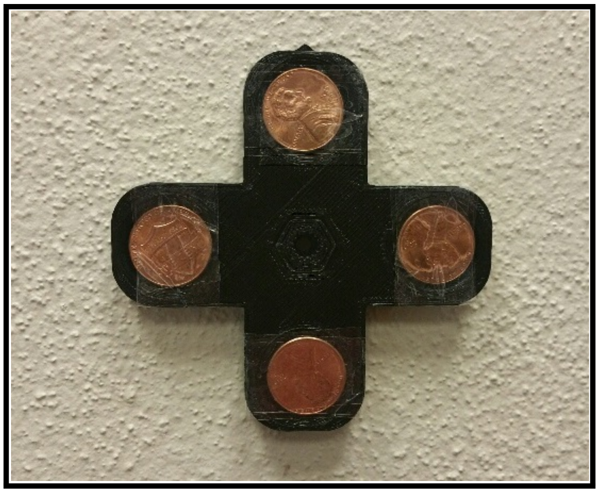


Figure 6.6: Load with Pennies

7. Once all of the pennies have been inserted, place the load back onto the motor shaft. The full setup is shown in Figure 6.7.

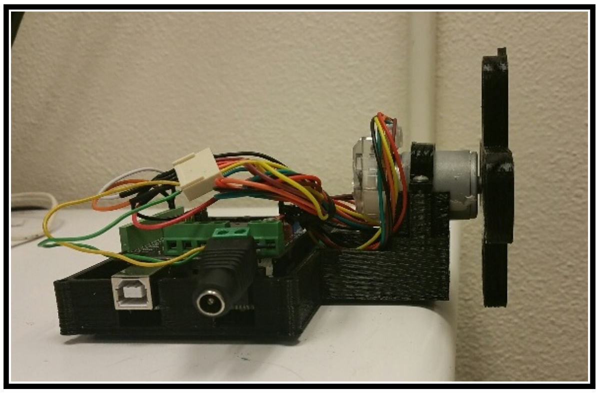


Figure 6.7: Full Assembly

6.3 Experimental Procedures

The exercises in this section will demonstrate how to find the transfer function of a Linear Time-Invariant system (LTI) using its frequency response. Frequency response is the steady-state response of a system to a sinusoidal input. The DC motor is the system that will be used. The first order model of the motor will be estimated using the experimental results of the frequency response. In the first exercise you will derive the motor's first order transfer function, sketch the theoretical Bode Plot, and find the transient response. In exercise 2 you will find the frequency response of the motor by applying a sinusoidal input voltage for a range of frequencies. For each sine wave input, you will find the output position and velocity of the motor. For each frequency, the peak to peak magnitude of the output position and velocity will be compared to the peak to peak magnitude of the input sinusoidal voltage. The magnitudes will be recorded and used to construct bode plots for position vs. input, and velocity vs. input. The transfer functions of the motor will be estimated in exercise 3 using the velocity vs. input experimental bode plot. The bode plot of the estimated transfer functions will then be simulated and compared to the experimental bode plots.

6.3.1 Exercise 1: The Motor Transfer Function

This first exercise will use the physical characteristics of the motor to derive the transfer function of the motor. Figure 6.8 shows the circuit diagram of the motor:

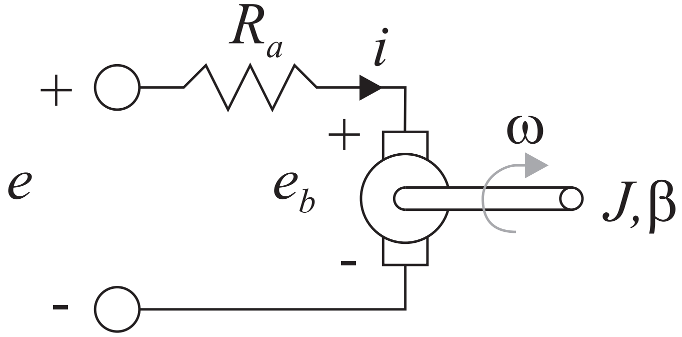


Figure 6.8: DC Motor Circuit Diagram

The motor's equations of motion can be found from the circuit diagram. The equations are shown below:

$$e = R_a i + e_b \tag{6.1}$$

$$J\dot{\omega} = \tau - \beta\omega \tag{6.2}$$

$$\tau = K_t i \tag{6.3}$$

$$e_b = K_b \omega \tag{6.4}$$

where J is the inertia of the armature and the load, K_t is the torque constant, R_a is the armature resistance, K_b is the back emf constant, τ is the torque, i is the armature current, e is the voltage applied to the motor, ω is the angular velocity of the motor, and e_b is the motor back emf.

8. Using Equations (1.1 - 1.4), solve for the blocks G_1, G_2, G_3, and G_4 in Figure 6.9.

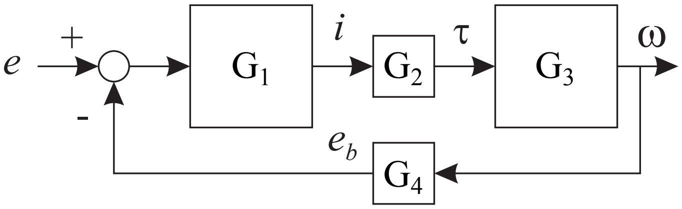


Figure 6.9: Empty Block Diagram for Motor

9. Use block diagram reduction to find the overall open loop transfer function $\frac{\omega(s)}{e(s)}$ for the motor. Simplify the transfer function to match the form of $\frac{K_m}{\tau_m s+1}$, where K_m and τ_m are constants.

   ⋆ The simplified overall open loop transfer function is shown in Equation 6.5

$$\frac{\omega(s)}{e(s)} = \frac{\frac{K_t}{R_a \beta} + \frac{1}{K_b}}{\left(\frac{J}{\beta} + \frac{J R_a}{K_b K_t}\right) s + 1} \tag{6.5}$$

   where

$$K_m = \frac{K_t}{R_a \beta} + \frac{1}{K_b} \tag{6.6}$$

   and

$$\tau_m = \frac{J}{\beta} + \frac{J R_a}{K_b K_t} \tag{6.7}$$

10. Sketch the magnitude of the bode plot for $\frac{K_m}{\tau_m s+1}$. Label the key points in the plot as functions of K_m and τ_m.

   ⋆ An example of the bode plot sketch is shown in Figure 6.10. The constant maximum magnitude of the bode plot corresponds to $20log_{10}(K_m)$, and the corner frequency ω_c, or break frequency, corresponds to $\frac{1}{\tau_m}$

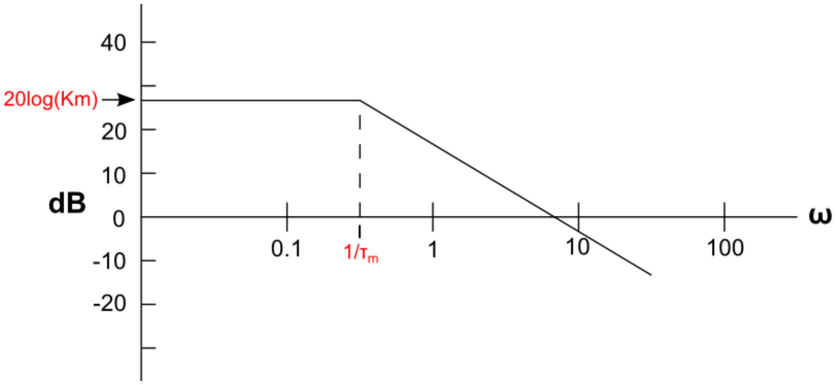


Figure 6.10: Example Bode Plot

11. Find the transient response of the system, $G(s) = \frac{K_m}{\tau_m s+1}$, given an input $x(t) = Asin(\omega t)$, and also find the steady state response.

   - $\star$ First, convert the transfer function to the form: $G(s) = \frac{\frac{K_m}{\tau_m}}{s+\frac{1}{\tau_m}}$.
   - $\star$ $X(s) = \frac{A\omega}{s^2+\omega^2}$ and $Y(s) = G(s)X(s)$.
   - $\star$ From $Y(s) = \frac{\frac{K_m}{\tau_m}}{(s+\frac{1}{\tau_m})}\frac{A\omega}{(s^2+\omega^2)}$, find the partial-fraction expansion.
   - $\star$ The partial-fraction expansion will give: $Y(s) = \frac{\frac{K_m}{\tau_m}}{(s+\frac{1}{\tau_m})}\frac{A\omega}{(s^2+\omega^2)} = \frac{a}{s+\frac{1}{\tau_m}} + \frac{cs+d}{s^2+\omega^2}$
   - $\star$ By taking the Laplace Transform of the partial-fraction expansion, the transient response will be: $y(t) = ae^{-\frac{t}{\tau_m}} + A\left|G(j\omega)\right| sin(\omega t + \angle G(j\omega))$.
   - $\star$ The steady state response will be: $y_{ss}(t) = A\left|G(j\omega)\right| sin(\omega t + \angle G(j\omega))$ since the $ae^{-\frac{t}{\tau_m}}$ term decays to zero.

6.3.2 Exercise 2: Experimental Frequency Response

In this exercise you will find the frequency response of the motor with the load attached by using the Arduino and Simulink. You will apply a sinusoidal input voltage, for multiple frequencies, into the motor. For each frequency, you will need to record the peak to peak steady-state output position and peak to peak steady-state output velocity, and compare these magnitudes to the peak to peak amplitude of the input sine wave voltage. The required frequencies for the sine wave input voltage are provided in Table 6.1. Once the table is complete, the values will be used to generate bode plots for the output position vs input voltage and output velocity vs input voltage. Using the velocity vs input bode plot, you will estimate K_m and τ_m.

12. Open MATLAB and set the MATLAB "Current Folder" location to the *Experiment_OLFR* folder you extracted in the software setup section. The files *SerialPlotData.m* and *BodePlotData.m* should show in the "Current Folder" section.

13. Open your *Sampling_NM* model used in the *Sampling and Data Acquisition* experiment, and save it as *OpenLoopFreqResponse*. Your model should now resemble the model in Figure 6.11.

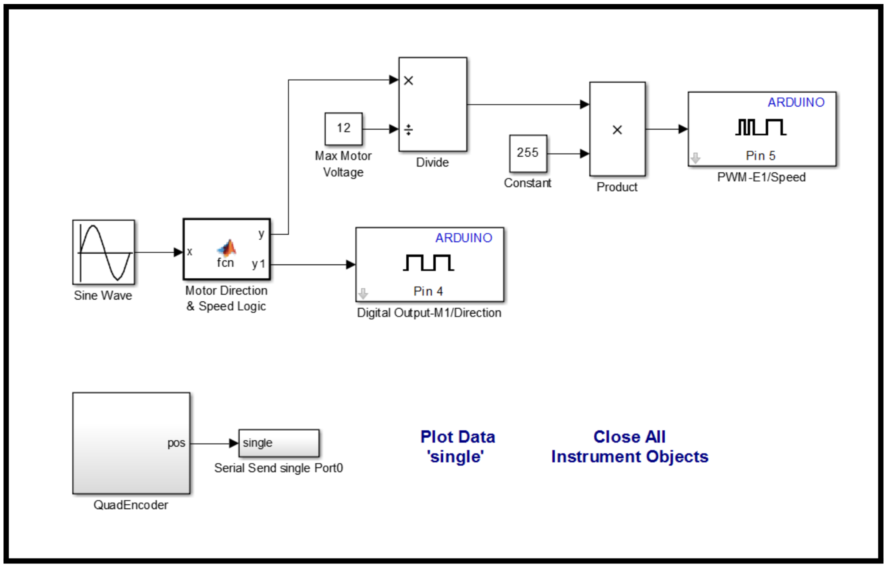


Figure 6.11: Existing Simulink Model

14. Add a "Gain" block to the model (Commonly Used Blocks → Gain).

15. Delete the connection between the "QuadEncoder" block and the "Serial Send single Port0", and connect the "QuadEncoder", "Gain", and "Serial Send single Port0" blocks as in Figure 6.12.

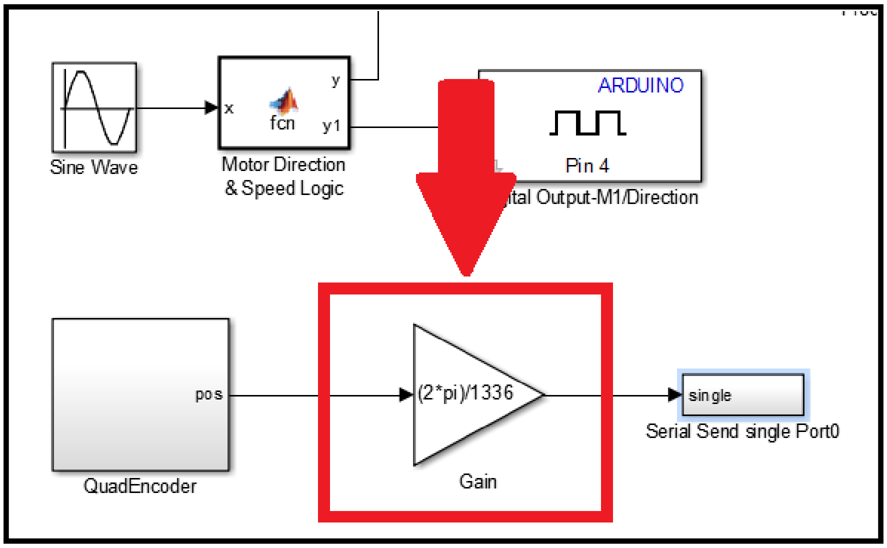


Figure 6.12: Simulink Model Changes

16. Ensure your model looks like the model in Figure 6.13, then save it.

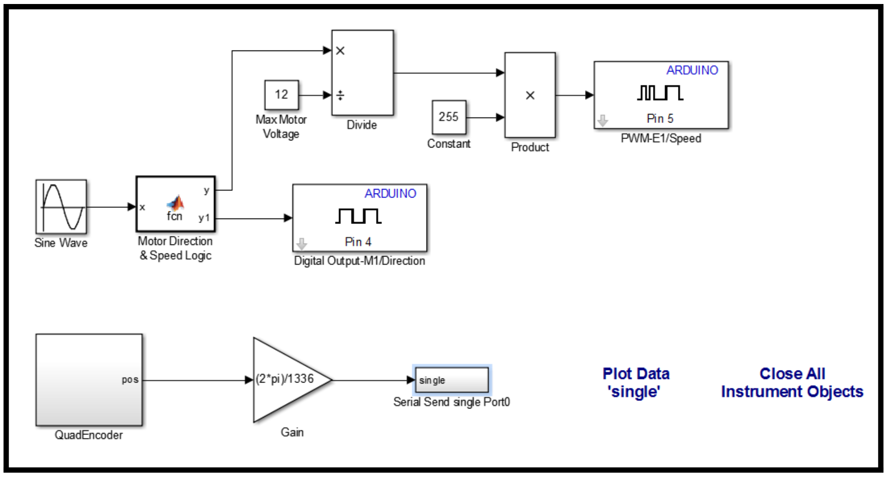


Figure 6.13: Final Simulink Model

17. Double-click the sine wave block and change the "Amplitude:" to 4, and the "Frequency (rad/sec):" to *2*pi*fs*.

18. In the MATLAB Command Window, type *Ts = 0.01;* and press enter. Then type *fs = .005;* and press enter.

19. Connect the Arduino to the PC, and download the Simulink model to the Arduino.

20. Once the model is successfully downloaded to the Arduino, select the "Plot Data 'single'" text.

21. In the pop-up window, change the COM port to the port your Arduino is using, and set the number of samples to plot to *50000*. Press "OK" to start plotting the data.

22. Check to see if the plot data is out of sync by manually moving the load left and right. If the data is in sync, it should resemble Figure 6.14. If your plot does not look similar to this, press "Byte Adjust" and manually move the load again. If the data is still incorrect, repeat this step a few more times if needed. After a few attempts, if the data is still incorrect, press "Stop" and close the plot window. Initiate the plotting process again by selecting the "Plot Data 'single'" text and entering the number of samples to plot. Repeat this step until the data is correct. Additional techniques for troubleshooting the serial data can be found in steps 79-82 of the *Sampling and Data Acquisition* experiment.

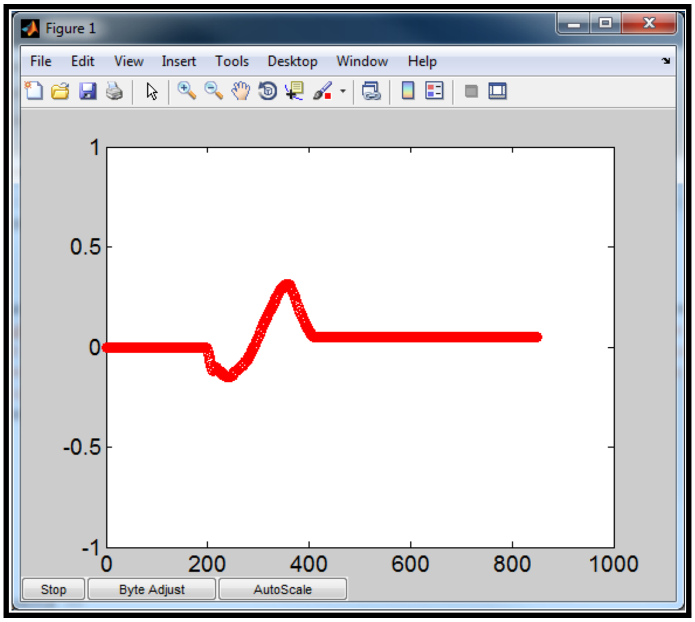


Figure 6.14: Serial Data Sync Test

23. Once the data is in sync, move the load back to the 0 position and plug in the power supply.

24. Press "Autoscale" once the motor starts rotating, if the values are not visible. Let the experiment run until you see at least 2 cycles of the output sine wave before pressing "Stop".

25. After pressing stop in the serial plot window, open the *SerialPlotData.m* file in the MATLAB Current Folder section, and run ▶ the file to generate the position vs time and velocity vs time plots.

26. Find the steady state peak to peak magnitudes for each plot, and record the values in the first row of Table 6.1. Also, in the table, record the peak to peak magnitude of the input sine wave, and the position vs input ($\frac{|\theta|}{|e|}$) and velocity vs input ($\frac{|\omega|}{|e|}$) magnitudes in decibels (dB), using $20log_{10}$.

27. In the MATLAB Command Window, type *clear all; close all; clc;* and press enter.

28. Repeat steps 18-26 for each frequency provided in Table 6.1. When returning to step 18, always set *Ts = 0.01;*, and set *fs* equal to the frequencies listed in the table. The extra rows in the table can be used for additional frequencies. Take as many readings as you need to get an accurate frequency response.

Frequency Response						
Number of Points	Frequency (Hz)	$\|e\|$ (peak to peak volts)	$\|\theta\|$ (rad)	$\|\omega\|$ (rad/s)	$\frac{\|\theta\|}{\|e\|}$ (dB)	$\frac{\|\omega\|}{\|e\|}$ (dB)
50,000	0.005					
25,000	0.01					
5,000	0.05					
2500	0.1					
1000	0.5					
500	1					

Table 6.1: Open Loop Frequency Response

- ⋆ The experimental data found for the load using zero pennies is shown in Table 6.2. The load weighed 17 grams, and two additional frequencies were used.

Frequency Response						
Number of Points	Frequency (Hz)	$\|e\|$ (peak to peak volts)	$\|\theta\|$ (rad)	$\|\omega\|$ (rad/s)	$\frac{\|\theta\|}{\|e\|}$ (dB)	$\frac{\|\omega\|}{\|e\|}$ (dB)
50,000	0.005	8	8267.5	310	60.29	31.77
25,000	0.01	8	4186	310	54.37	31.77
5,000	0.05	8	829.5	294	40.31	31.31
2500	0.1	8	396.15	255	33.86	30.07
1000	0.5	8	30	91	11.48	21.12
500	1	8	7.3	47.5	-0.795	15.47
1500	0.25	8	104.25	160	22.299	26.02
2000	0.175	8	182.9	200	27.18	27.96

Table 6.2: Experimental Data - No Pennies

⋆ The experimental data found for the load using 10 pennies is shown in Table 6.3. The load weighed 46.7 grams, and no additional frequencies were used.

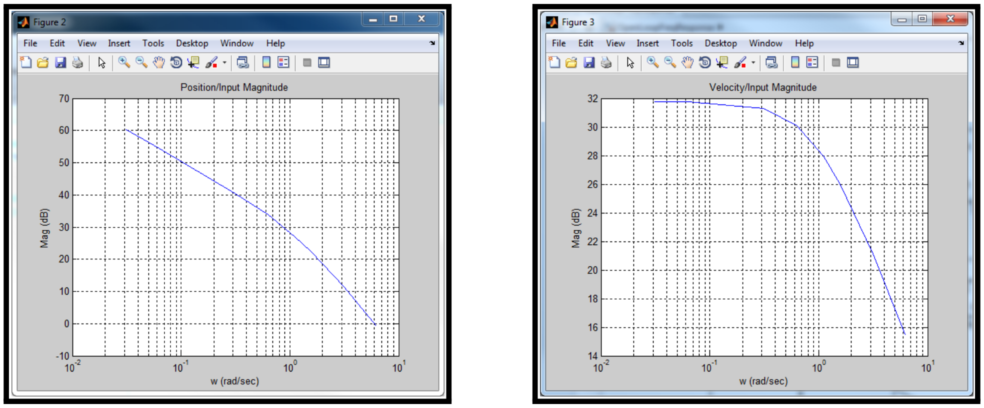
Frequency Response						
Number of Points	Frequency (Hz)	$\|e\|$ (peak to peak volts)	$\|\theta\|$ (rad)	$\|\omega\|$ (rad/s)	$\frac{\|\theta\|}{\|e\|}$ (dB)	$\frac{\|\omega\|}{\|e\|}$ (dB)
50,000	0.005	8	9201	344	61.25	32.67
25,000	0.01	8	4612.5	338	55.22	32.52
5,000	0.05	8	704.6	218	38.89	28.71
2500	0.1	8	217	135	28.67	24.54
1000	0.5	8	10.7	91	33.45	12.43
500	1	8	2.81	47.5	19.4	7.69

Table 6.3: Experimental Data - 10 Pennies

29. Open the *BodePlotData.m* from the MATLAB "Current Folder" location. Input the $\frac{|\theta|}{|e|}$ values from your table inside the brackets of the *GpdB* variable. For example, if all of the $\frac{|\theta|}{|e|}$ values were equal to 1, and no additional frequencies were used, *GpdB = [];* would become *GpdB = [1, 1, 1, 1, 1, 1];*. The order you enter the magnitude values depends on the order of the frequencies in the variable *f*. The frequencies are in numerical order from smallest to largest. Therefore, if you find output magnitudes for any additional frequencies, you will need to add the additional frequencies, in the correct order, to the variable *f*. For example, if additional frequencies of 0.25 Hz and 2 Hz are used and the output magnitudes equal 1.5 and 2, the frequency variable in the *BodePlotData.m* file needs to be changed to *f = [0.005, 0.01, 0.05, 0.1, 0.25, 0.5, 1, 2];*, and the magnitude variable should be changed to *GpdB = [1, 1, 1, 1, 1.5, 1, 1, 2];*.

30. Input the $\frac{|\omega|}{|e|}$ values from your table inside the brackets of the *GvdB* variable, and run the file to produce the two bode plots.

   $\star$ The frequency response data for the load with no pennies produced the bode plots shown in Figure 6.15a and Figure 6.15b.

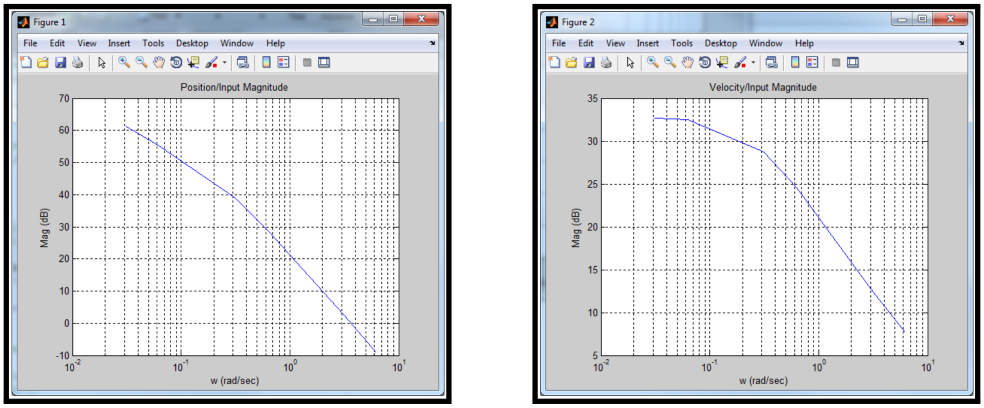


(a) Position vs Input

(b) Velocity vs Input

Figure 6.15: Bode Plots - Load with No Pennies

- ⋆ The frequency response data for the load with 10 pennies produced the bode plots shown in Figure 6.16a and Figure 6.16b.



(a) Position vs Input

(b) Velocity vs Input

Figure 6.16: Bode Plots - Load with 10 Pennies

31. Comparing the experimental output velocity vs input voltage bode plot with your theoretical sketch from step 10, estimate K_m and τ_m.

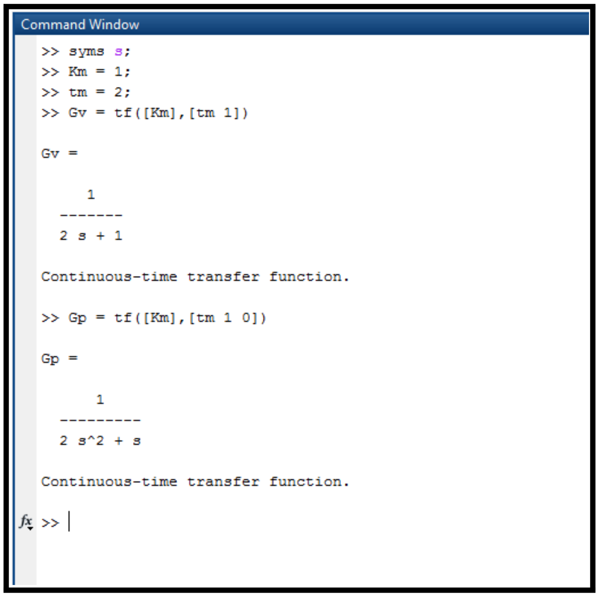
    - ⋆ The velocity vs input bode plot for the load with no pennies shows that $20log_{10}(Km) = 31.77$, which means $Km = 38.75$, and $\tau_m = \frac{1}{0.887} = 1.127$.
    - ⋆ The velocity vs input bode plot for the load with 10 pennies shows that $20log_{10}(Km) = 32.65$, which means $Km = 43$, and $\tau_m = \frac{1}{0.2239} = 4.467$.

32. Save the plots for your lab report, and leave them open.

6.3.3 Exercise 3: Simulation and Experimental Results Comparison

In this subsection, you will use the velocity vs input bode plot from Exercise 2 to estimate K_m and τ_m. Then you will use MATLAB the bode plots for $\frac{\omega(s)}{e(s)} = \frac{K_m}{\tau_m s+1}$ and $\frac{\theta(s)}{e(s)} = \frac{K_m}{(\tau_m s+1)s}$ and compare them to the experimental bode plots.

33. In MATLAB, create the transfer functions $Gv(s) = \frac{K_m}{\tau_m s+1}$ and $Gp(s) = \frac{K_m}{(\tau_m s+1)s}$ using the values you estimated for K_m and τ_m in exercise 2. Figure 6.17 shows the code you need to type in the MATLAB Command Window in order to create the transfer functions. Change the variables Km and tm to equal your values for K_m and τ_m.

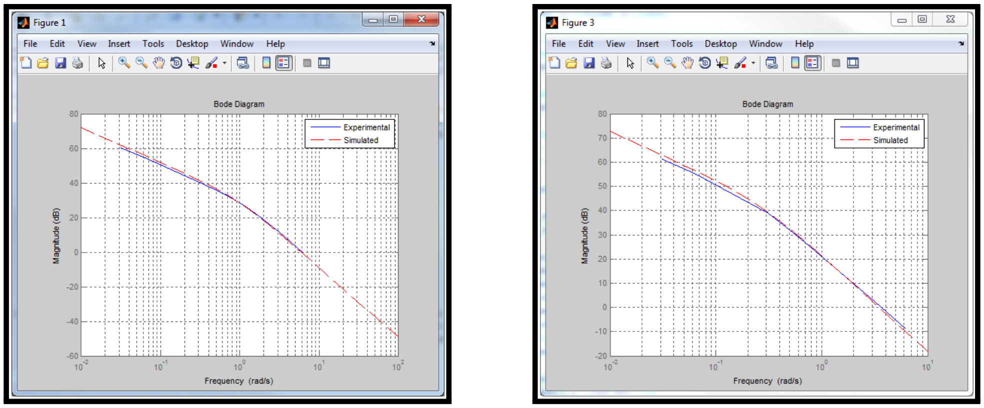


```
>> syms s;
>> Km = 1;
>> tm = 2;
>> Gv = tf([Km],[tm 1])

Gv =

     1
  -------
  2 s + 1

Continuous-time transfer function.

>> Gp = tf([Km],[tm 1 0])

Gp =

      1
  ---------
  2 s^2 + s

Continuous-time transfer function.

>> |
```

Figure 6.17: MATLAB Transfer Function

34. Plot the bode plot for your estimated $\frac{|\theta|}{|e|}$ on top of the experimental bode plot from exercise 2 by typing *figure(1);bodemag(Gp,w,'--r')* and press enter. Type *grid on* and press enter, then type *legend('Experimental','Simulated')* and press enter. Your MATLAB Figure 1 should now show both the experimental and simulated bode plots, with a legend in the top right corner.

   ⋆ A comparison of the bode plots for position vs input for the load with no pennies is shown in Figure 6.18a, and the comparison for the load with 10 pennies is shown in Figure 6.18b.



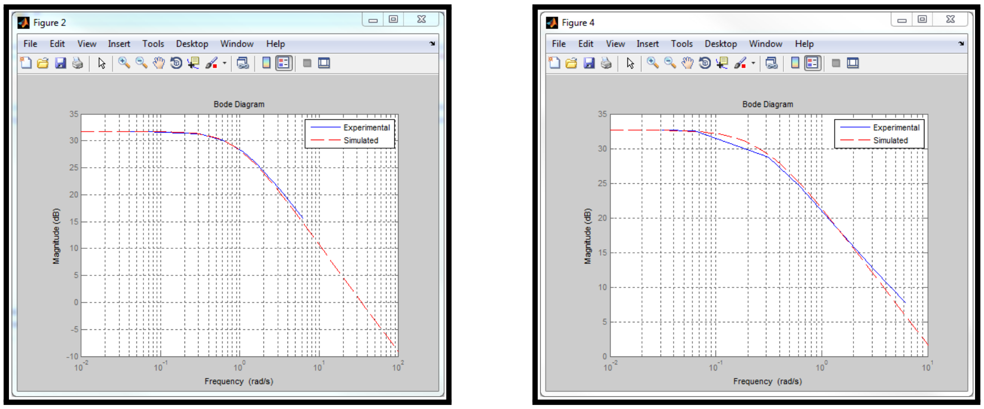
(a) Position vs Input - No Pennies

(b) Position vs Input - 10 Pennies

Figure 6.18: Bode Plot Comparisons

35. Plot the bode plot for your estimated $\frac{|\omega|}{|e|}$ on top of the experimental bode plot from exercise 2 by typing *figure(2);bodemag(Gv,w,'--r')* and press enter. Type *grid on* and press enter, then type *legend('Experimental','Simulated')* and press enter.

    ⋆ A comparison of the bode plots for position vs input for the load with no pennies is shown in Figure 6.19a, and the comparison for the load with 10 pennies is shown in Figure 6.19b.



(a) Velocity vs Input - No Pennies

(b) Velocity vs Input - 10 Pennies

Figure 6.19: Bode Plot Comparisons

36. Save both figures for your lab report as different names from what you saved them as in exercise 2.

37. How well do your simulated bode plots match your experimental bode plots?

Explain possible reasons for any differences.

6.4 Conclusion/Student Feedback

The open loop frequency response of a DC motor was used to demonstrate the importance of frequency response. The motor's frequency response was found by inputting sine wave voltages, of varying frequencies, into the motor and recording the output positions and velocities for each frequency. The motor's theoretical first order model was estimated using the experimental frequency response data. The response of the estimated model was then compared to experimental response, demonstrating how well the first order transfer function of a system can be estimated using the experimental frequency response of that system.

CHAPTER 7

CLOSED LOOP FREQUENCY RESPONSE

7.1 Objective

This experiment will demonstrate the importance of frequency response by finding the closed loop frequency response of a DC motor with an attached load. In the closed loop model, various types of feedback and various values of feedback gains will be used to demonstrate their effects on the system's frequency response. For the various feedback gains, the closed loop frequency response will be found by inputting a sinusoidal reference position, for multiple frequencies, into the motor. For each frequency, the motor's output position will be recorded. The magnitude of the output position will be compared to the magnitude of the reference input. Magnitude bode plots for output position versus reference input will be generated. Simulated bode plots of the closed loop model will be generated and compared to the experimental bode plots.

7.2 Setup

7.2.1 Required Materials

<u>Hardware</u>

The hardware materials required for this experiment are the same materials used in the *Open Loop Frequency Response* experiment. A list of the materials needed for this experiment is provided below for reference.

- DC Motor with Encoder
- Arduino Mega 2560 Rev3
- Motor Shield (DFRobot L298P)
- 12V DC Power Supply (12VDC 3A Wall Adapter Power Supply)
- USB cable (Standard A to B plug)

- Female Barrel Jack (2.1mm x 5.5mm Female CCTV Power Jack Adapter)
- 8 wires
- 3D Printed Base
- 3D Printed Motor Clamp
- 2 #4-40 1/2” screws
- 2 #4-40 1/4” screws
- Small Flat screwdriver
- Small Phillips screwdriver
- Sticky Tack
- Sandpaper or Sanding Sponge (Optional)
- Pennies - Same number used in the *Open Loop Frequency Response* experiment
- Scotch tape
- Small 3-D Printed Load

Software

- MATLAB/Simulink 2013a or later
  - ⋆ *The steps and images related to MATLAB/Simulink for this experiment were created using MATLAB/Simulink 2013a. Therefore some steps and images may be a little different if you are not using this version. If you are in fact using a different version, make sure you know the steps for running models onto the Arduino for your version of Simulink.*
- MATLAB files

Prerequisite Experiments

- Open Loop Frequency Response

7.2.2 Software Setup

The necessary software files that are needed for this experiment can be downloaded from the Take Home Labs webpage. One method for downloading the files is shown in the steps below.

1. Open your internet browser and navigate to `csl.okstate.edu`

2. On the left side of the Homepage, select "Courses"

3. In the middle section of the Courses page select "System Dynamics"

4. In the middle section of the System Dynamics page find and select *Closed Loop Frequency Response.*

5. On the Closed Loop Frequency Response page select "Software/Code" in the rightmost section. A zipfile named *Experiment_CLFR* should download.

6. Right-click the file and choose "Extract All...", or any other method of extracting files on your PC.

7. Extract this folder somewhere convenient, and remember the location. This will be the folder where all of the files and plots created for this experiment are saved.

7.2.3 Hardware Setup

Since there was no additional hardware needed for this experiment, there is no additional setup that needs to be done. The hardware should still be set up from the *Open Loop Frequency Response* experiment.

7.3 Experimental Procedures

In this section you will derive the theoretical closed loop DC motor model using the open loop transfer function that was found in the *Open Loop Frequency Response* experiment. You will sketch the bode plot of the the theoretical closed loop DC motor model, for various feedback gain values. Using the same gains, you will experimentally find the closed loop frequency response of the DC motor and load. Using MATLAB, you will then simulate the bode plots of the theoretical closed loop transfer functions, and compare them with the experimental bode plots.

7.3.1 Exercise 1: Simulated Closed Loop Response

8. The closed loop model for the DC motor is provided in Figure 7.1. Find the closed loop transfer function $\frac{Y(s)}{R(s)}$ using block diagram reduction.

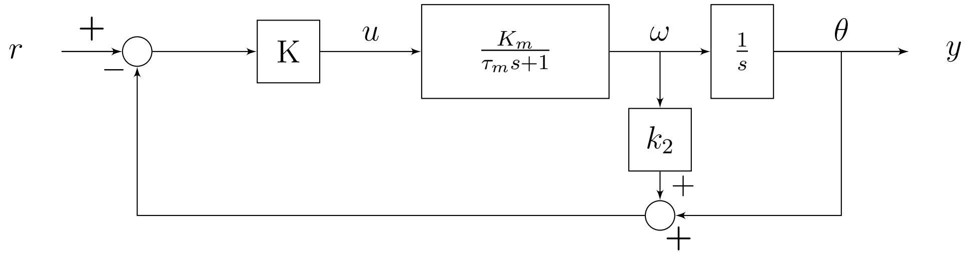


Figure 7.1: Motor Closed Loop Model

- ⋆ Figure 7.2 shows the first step in simplifying the model by moving k_2 over the $\frac{1}{s}$ block.

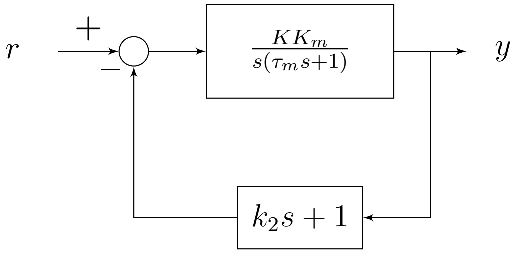
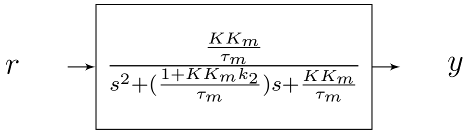


Figure 7.2: Block Diagram Reduction

- ⋆ Figure 7.3 shows the final transfer function after reducing the feedback loop and doing some simplification.



r
$\frac{\frac{KK_m}{\tau_m}}{s^2+(\frac{1+KK_mk_2}{\tau_m})s+\frac{KK_m}{\tau_m}}$
y

Figure 7.3: Block Diagram Reduction (2)

9. For $K = 2$ and $k_2 = 0$ find the closed loop poles, the damping ratio, and the natural frequency. Sketch the bode plot of the closed loop transfer function, and save it for your lab report. What type of response is this? Explain?

   - ⋆ The theoretical bode plot for the load with no pennies added is shown in Figure 7.4.

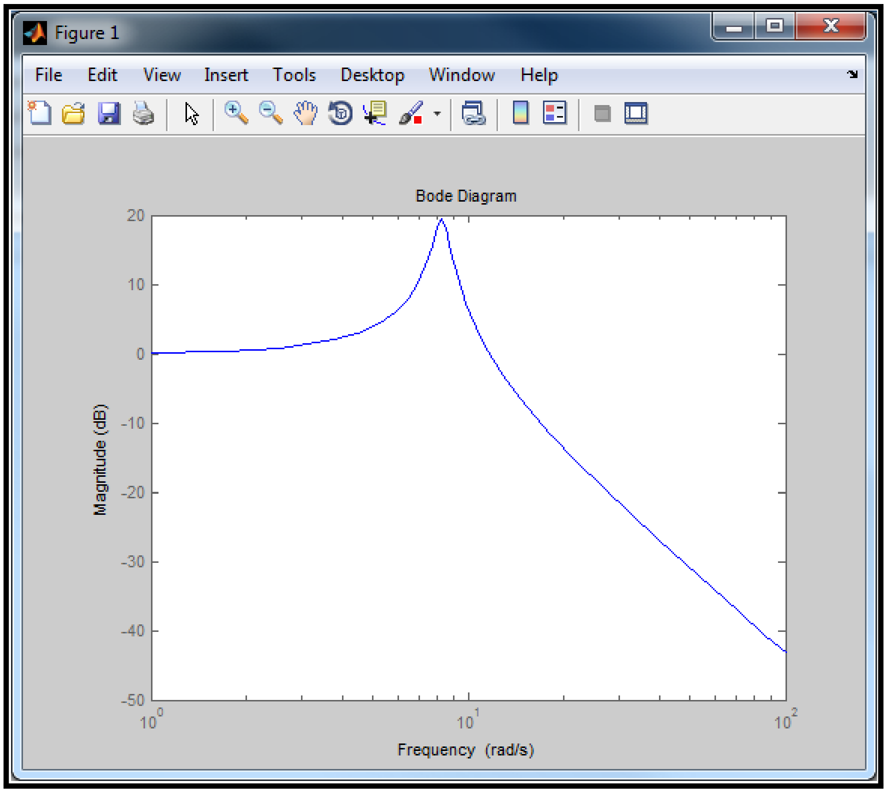


Figure 7.4: Theoretical Underdamped Bode Plot

⋆ The theoretical bode plot for the load with 10 pennies added is shown in Figure 7.5

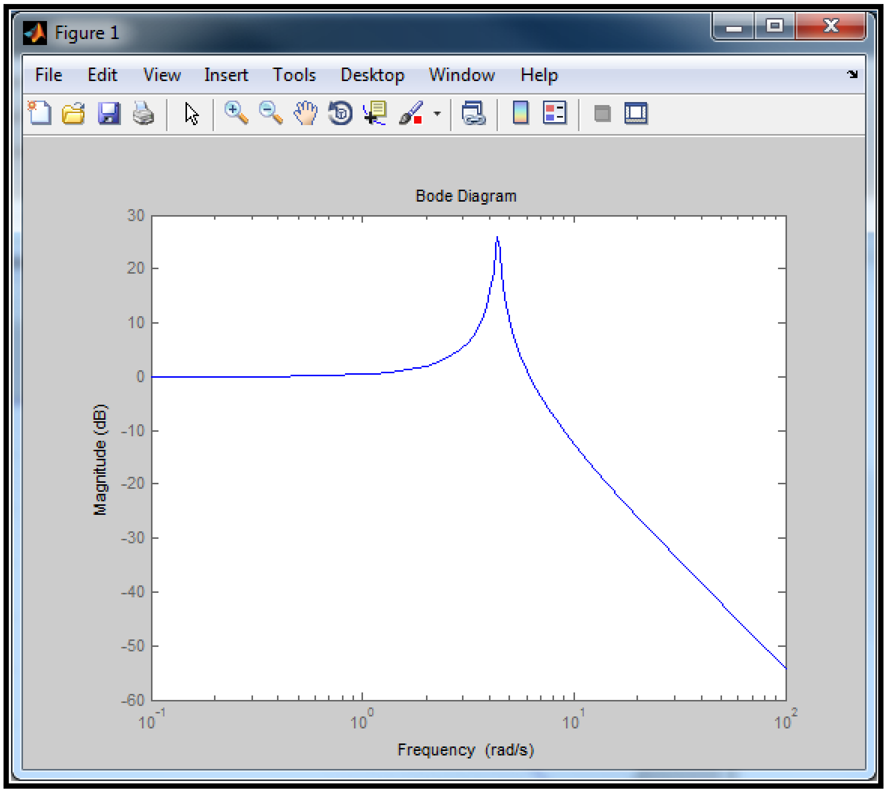


Figure 7.5: Theoretical Underdamped Bode Plot

⋆ The response will be underdamped because $\zeta < 1$. For the load with no pennies, the poles are located at $-0.4437 \pm j8.281$, $\zeta = 0.0535$, and $\omega_n = 8.2926$. Therefore, $\omega_r = 8.2688$ and $M_r = 19.423$ dB. For the load with 10 pennies, $-0.1119 \pm j4.386$, $\zeta = 0.0255$, and $\omega_n = 4.3877$. Therefore, $\omega_r = 4.3848$ and $M_r = 25.85$ dB. ζ and ω_n can be found by comparing the derived closed loop transfer function to the standard form of a second-order system, shown in Equation 7.1. Since the response is underdamped, and $0 \leq \zeta \ll 0.707$, the bode plot will have a peak at ω_r with a maximum value of M_r. Equation 7.2 and Equation 7.3 show how ω_r and M_r can be found from ζ and ω_n.

$$G(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n + \omega_n^2} \tag{7.1}$$

$$\omega_r = \omega_n\sqrt{1 - 2\zeta^2} \tag{7.2}$$

$$M_r = \frac{1}{2\zeta\sqrt{1 - \zeta^2}} \tag{7.3}$$

10. Repeat step 9 for $K = 2$ and $k_2 = 0.5$.

   ⋆ The theoretical bode plot for the load with no pennies added is shown in Figure 7.6.

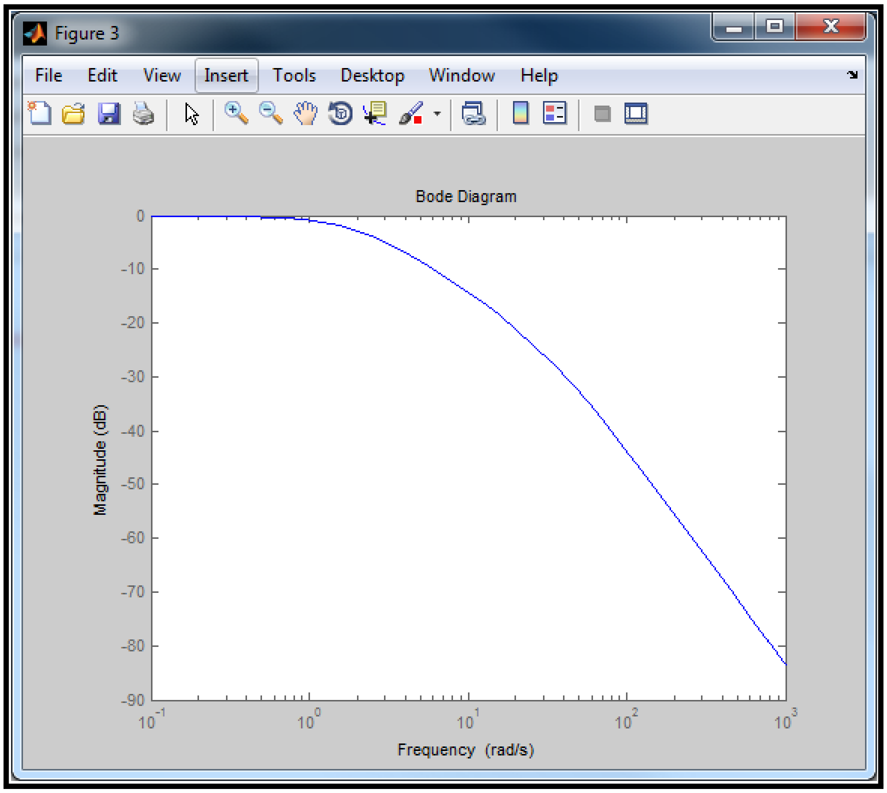


Figure 7.6: Theoretical Overdamped Bode Plot - No Pennies

   ⋆ The theoretical bode plot for the load with 10 pennies added is shown in Figure 7.7

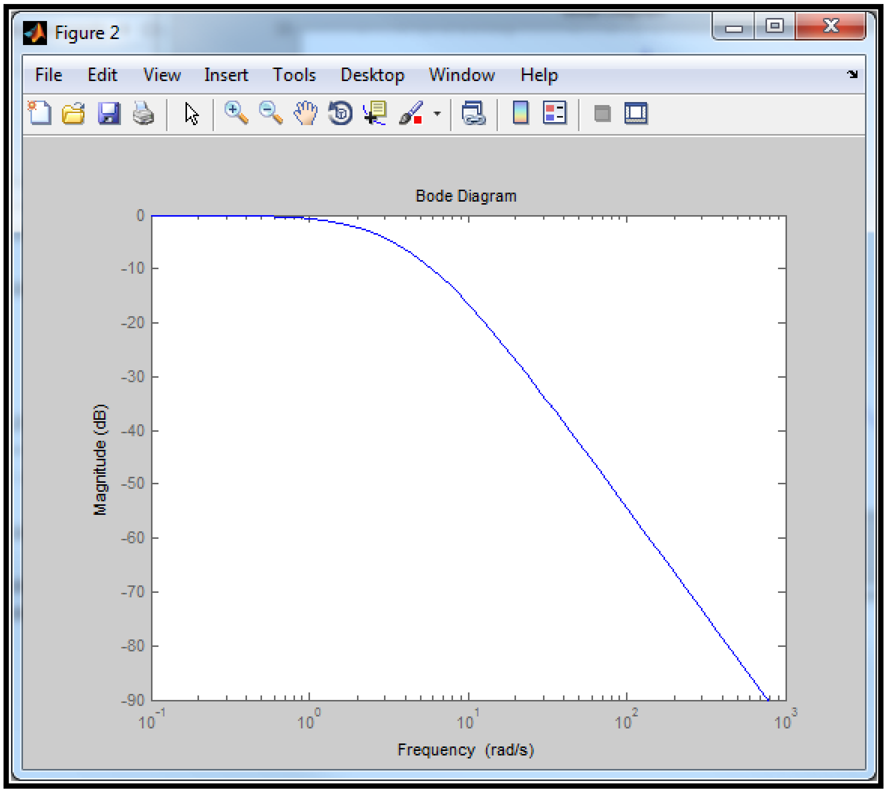


Figure 7.7: Theoretical Overdamped Bode Plot - 10 Pennies

⋆ The response will be overdamped because $\zeta > 1$. For the load with no pennies, the poles of the transfer function are located at -2.07 and -33.2, $\zeta = 2.127$, and $\omega_n = 8.2926$. For the load with 10 pennies, the poles of the transfer function are located at -2.688 and -7.1618, $\zeta = 1.122$, and $\omega_n = 4.3877$.. The bode plots will have a $-20\ \frac{dB}{decade}$ change in slope at frequencies equal to the magnitudes of the pole locations.

7.3.2 Exercise 2: Experimental Closed Loop Response

11. Open MATLAB and set the MATLAB "Current Folder" location to the *Experiment_CLFR* folder you extracted in the software setup section. The files *SerialPlotData.m* and *BodePlotData.m* should show in the "Current Folder" section.

12. Open your *OpenLoopFreqResponse* model used in the *Open Loop Frequency Response* experiment, and save it as *ClosedLoopFreqResponse*. Your model should now resemble the model in Figure 7.8.

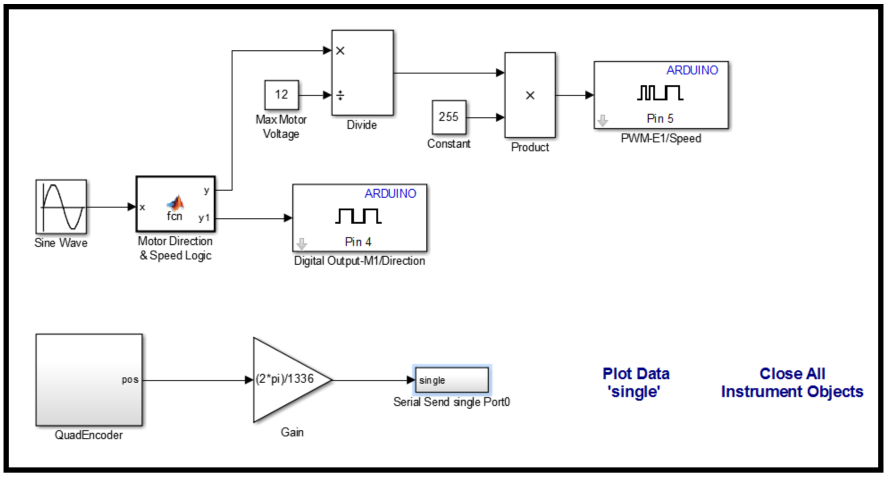


Figure 7.8: Existing Simulink Model

13. Click and drag over the motor control blocks as in Figure 7.9. Then press Ctrl+X to cut the blocks.

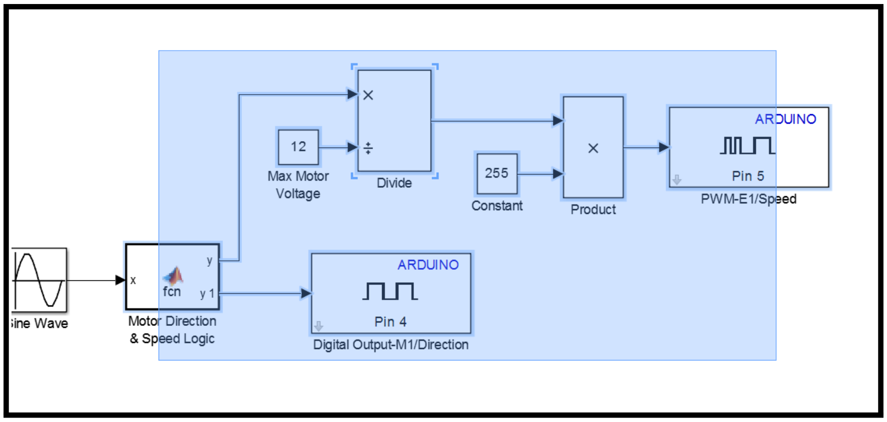


Figure 7.9: Select Motor Control Blocks

14. Add a Subsystem block to the model (Simulink → Ports and Subsystems → Subsystem).

15. Double-click the Subsystem block, and inside press Ctrl+V to paste the motor control blocks. Delete the “Out1” port and connect the “In1” port to the MATLAB function block, as in Figure 7.10.

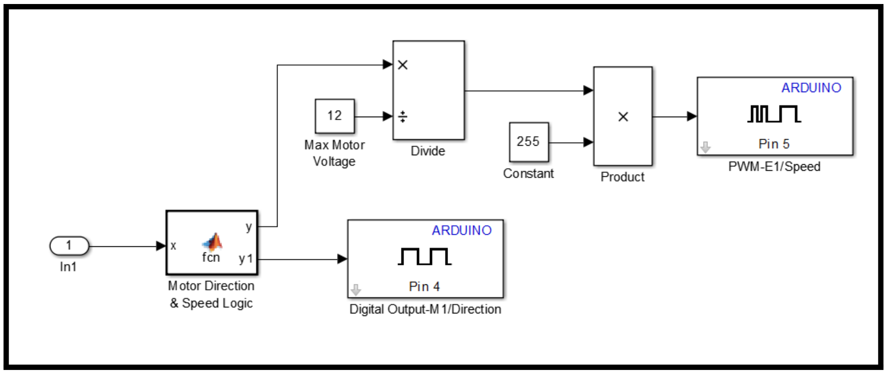


Figure 7.10: Motor Control Subsystem

16. Navigate back to the main model by pressing the "ClosedLoopFreqResponse" text, as in Figure 7.11.

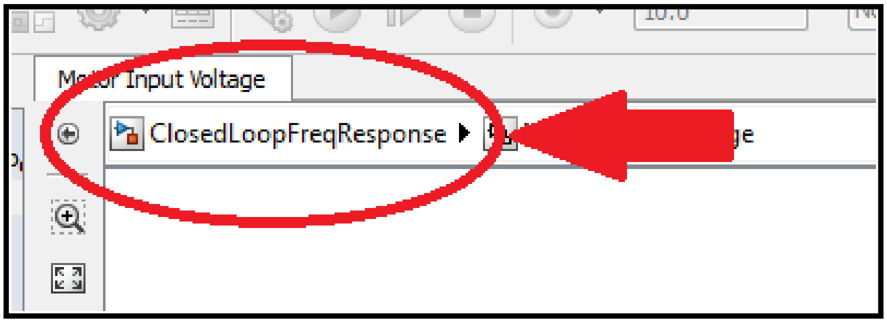


Figure 7.11: Exit the Subsystem Block

17. The model should now resemble Figure 7.12. Add a Sum block to the model (Simulink → Commonly Used Blocks → Sum).

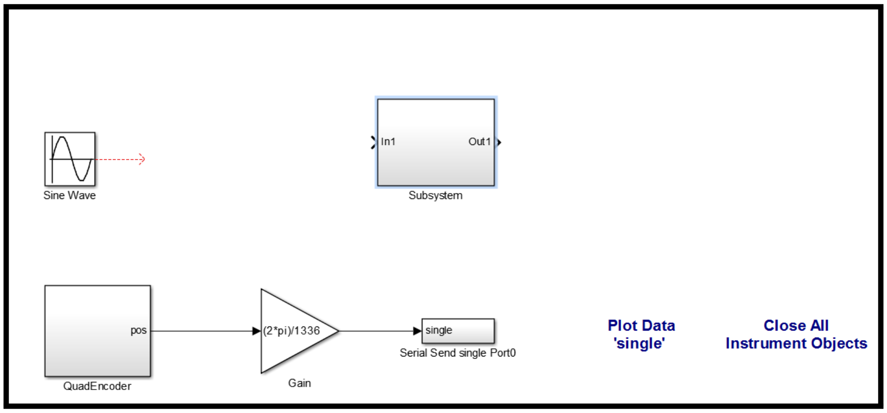


Figure 7.12: Current Model

18. Double-click the Sum block and change "List of signs:" to |+-. Now add the

following blocks to the model:

- Another Sum block
- 3 Gain blocks: Simulink → Commonly Used Blocks → Gain
- A Difference block: Simulink → Discrete → Difference
- A “2 - Serial Send single Port0” block: Take Home Labs Arduino Support Package → 2 - Serial Send single Port0

19. Connect the blocks together, as in Figure 7.13.

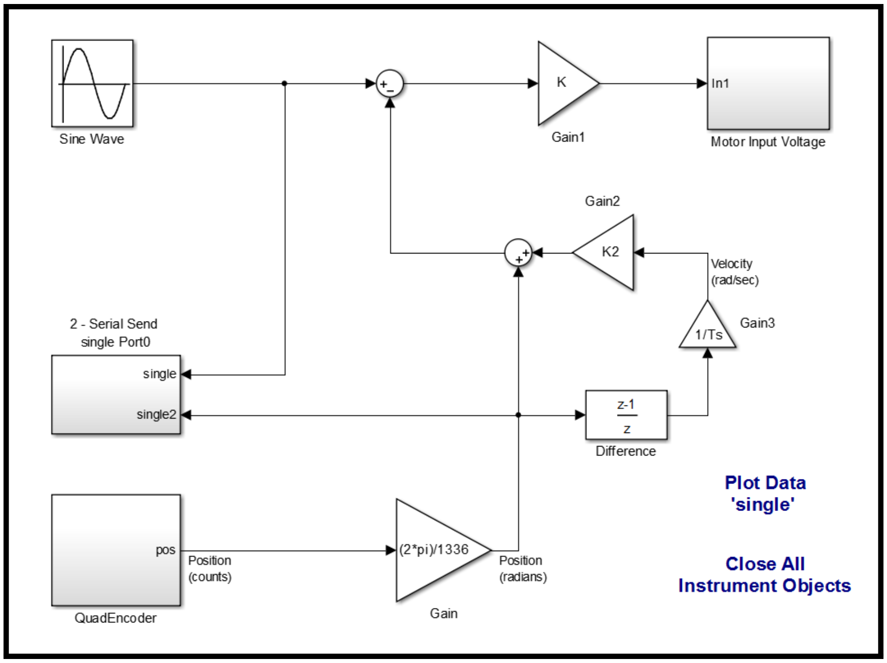


Figure 7.13: Final Model

20. Change the value of the gain block before the subsystem to *K*. Change the value of the gain blocks after the difference block to *1/Ts* and *K2*.

First Set of Gains

21. Double-click the sine wave block and change the “Amplitude:” to *pi/2*, and the “Frequency (rad/sec):” to _2*pi*fs_.

22. Create the constant variables you need in the workspace by typing in the following expressions in the MATLAB Command Window and pressing enter after each one:

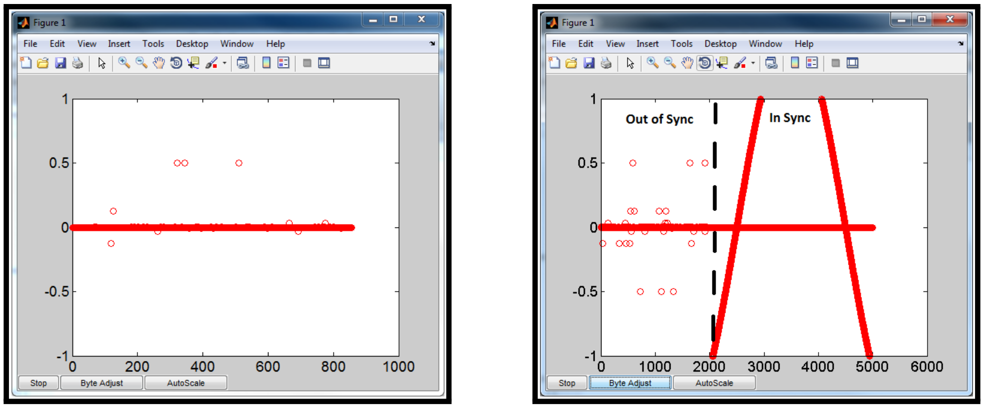
- *Ts = 0.01;*
- *fs = 0.05;*
- *K = 2;*
- *K2 = 0;*

23. Connect the Arduino to the PC, and download the Simulink model to the Arduino.

24. Once the model is successfully downloaded to the Arduino, select the "Plot Data 'single'" text.

25. In the pop-up window, change the COM port to the port your Arduino is using, and set the number of samples to plot to *7500*. Press "OK" to start plotting the data. The input sine wave and output position are both being sent over the serial port, so there will be two graphs showing in the plot window.

26. If the data is in sync, you should see a sine wave plotting, and a constant 0 being plotted. If the data looks like Figure 7.14a, it is out of sync so press the "Byte Adjust" button. When the data becomes in sync, it will resemble the right half of Figure 7.14b. If your plot does not look like this, press "Byte Adjust" a few more times, if needed. After a few attempts, if the data is still out of sync, press "Stop" and close the plot window. Initiate the plotting process again by selecting the "Plot Data 'single'" text and entering the number of samples to plot. Repeat this step until the data is correct. Additional techniques for troubleshooting the serial data can be found in steps 79-82 of the *Sampling and Data Acquisition* experiment.



(a) Data Out of Sync

(b) Data In Sync

Figure 7.14: Serial Plot Data

27. Once the data is in sync, plug in the power supply.

28. Press "Autoscale" once the motor starts rotating, if the values are not visible. Let the experiment run until you see at least 2 cycles of the output sine wave before pressing "Stop".

29. After pressing stop in the serial plot window, open the *SerialPlotData.m* file in the MATLAB Current Folder section, and run ▷ the file. The file will extract the reference input and output position, and plot them vs time on the same graph.

30. Find the steady state peak to peak magnitude of the output position, and record the value in the third column of Table 7.1. In the fourth column, record the peak to peak magnitude of the input sine wave. In the final column, record the ratio of the output magnitude vs input magnitude $\frac{|y|}{|r|}$ in decibels (dB), using $20log_{10}$.

Frequency Response				
Number of Points	Frequency (Hz)	$\lvert r \rvert$ (rad)	$\lvert y \rvert$ (rad)	$\frac{\lvert y \rvert}{\lvert r \rvert}$ (dB)
7,500	0.05			
5,000	0.1			
2,000	0.25			
1000	0.5			
750	1			
500	5			

Table 7.1: Closed Loop Frequency Response

31. In the MATLAB Command Window, type *clear all; close all; clc;* and press enter.

32. Repeat steps 22-31 for each frequency provided in Table 7.1. When returning to step 22, always set *fs* equal to the frequencies listed in the table. The extra rows in the table can be used for additional frequencies. Take as many readings as you need to get an accurate frequency response.

    ⋆ The experimental data found for the load using zero pennies is shown in Table 7.2. The load weighed 17 grams, and seven additional frequencies were used.

Frequency Response				
Number of Points	Frequency (Hz)	$\lvert r\rvert$ (rad)	$\lvert y\rvert$ (rad)	$\frac{\lvert y\rvert}{\lvert r\rvert}$ (dB)
5000	0.1	π	2.25	-2.9
2500	0.5	π	3.72	1.47
2000	0.7	π	4.21	2.54
1000	1	π	5.715	5.2
1000	1.3	π	6.2	5.9
500	2	π	21.8	16.8
200	5	π	0.68	-13.29
750	1.75	π	14.8	13.46
350	3.5	π	1	-9.94
500	2.5	π	1.76	-5.03
500	2.1	π	18.4	15.4
500	2.2	π	15.9	14.08
500	2.3	π	14.4	13.22
500	2.4	π	4	2.1

Table 7.2: Experimental Results - Underdamped w/ No Pennies

⋆ The experimental data found for the load using 10 pennies is shown in Table 7.3. The load weighed 46.7 grams, and two additional frequencies were used.

Frequency Response				
Number of Points	Frequency (Hz)	$\lvert r \rvert$ (rad)	$\lvert y \rvert$ (rad)	$\frac{\lvert y \rvert}{\lvert r \rvert}$ (dB)
5000	0.1	π	2.29	-2.75
2500	0.5	π	4.88	3.8
2000	0.7	π	8	8.12
1000	1	π	15.55	13.9
1000	1.3	π	15.35	13.8
500	2	π	1.5	-6.42
200	5	π	.15	-26.42
750	1.1	π	26.1	18.39
750	1.2	π	19.25	15.75

Table 7.3: Experimental Results - Underdamped w/ 10 Pennies

Second Set of Gains

33. Double-click the sine wave block and change the "Amplitude:" to *3*pi/2*, and the "Frequency (rad/sec):" to *2*pi*fs*.

34. Create the constant variables you need in the workspace by typing in the following expressions in the MATLAB Command Window and pressing enter after each one:

    - *Ts = 0.01;*
    - *fs = 0.05;*
    - *K = 2;*
    - *K2 = 0.5;*

35. Connect the Arduino to the PC, and download the Simulink model to the Arduino.

36. Once the model is successfully downloaded to the Arduino, select the "Plot Data 'single'" text.

37. In the pop-up window, change the COM port to the port your Arduino is using, and set the number of samples to plot to *7500*. Press “OK” to start plotting the data.

38. Once the data is in sync, plug in the power supply.

39. Press “Autoscale” once the motor starts rotating, if the values are not visible. Let the experiment run until you see at least 2 cycles of the output sine wave before pressing “Stop”.

40. After pressing stop in the serial plot window, open the *SerialPlotData.m* file in the MATLAB Current Folder section, and run ▶ the file.

41. Find the steady state peak to peak magnitude of the output position, and record the value in the fourth column of Table 7.4. In the third column, record the peak to peak magnitude of the input sine wave. In the final column, record the ratio of the output magnitude vs input magnitude $\frac{|y|}{|r|}$ in decibels (dB), using $20log_{10}$.

Frequency Response				
Number of Points	Frequency (Hz)	$\|r\|$ (rad)	$\|y\|$ (rad)	$\frac{\|y\|}{\|r\|}$ (dB)
7,500	0.05			
5,000	0.1			
2,000	0.25			
1000	0.5			
750	1			
500	5			

Table 7.4: Closed Loop Frequency Response (2)

42. In the MATLAB Command Window, type *clear all; close all; clc;* and press enter.

43. Repeat steps 34-42 for each frequency provided in Table 7.4. When returning to step 34, always set *fs* equal to the frequencies listed in the table. The extra rows in the table can be used for additional frequencies. Take as many readings as you need to get an accurate frequency response.

    ⋆ The experimental data found for the load using zero pennies is shown in Table 7.5. The load weighed 17 grams, and one additional frequency was used.

Frequency Response				
Number of Points	Frequency (Hz)	$\|r\|$ (rad)	$\|y\|$ (rad)	$\frac{\|y\|}{\|r\|}$ (dB)
7,500	0.05	3π	8.3	-1.10
5,000	0.1	3π	7.87	-1.57
2,000	0.25	3π	6.3	-3.5
1000	0.5	3π	4.2	-7.02
750	1	3π	2.4	-11.9
500	5	3π	0.6	-23.9
1500	0.35	3π	5.3	-5

Table 7.5: Experimental Results - Overdamped w/ No Pennies

⋆ The experimental data found for the load using 10 pennies is shown in Table 7.6. The load weighed 46.7 grams, and no additional frequencies were used.

Frequency Response				
Number of Points	Frequency (Hz)	$\lvert r\rvert$ (rad)	$\lvert y\rvert$ (rad)	$\frac{\lvert y\rvert}{\lvert r\rvert}$ (dB)
7,500	0.05	3π	8.35	-1.05
5,000	0.1	3π	7.88	-1.56
2,000	0.25	3π	6.35	-3.43
1000	0.5	3π	4.52	-6.38
750	1	3π	2.63	-11.09
500	5	3π	0.45	-26.4

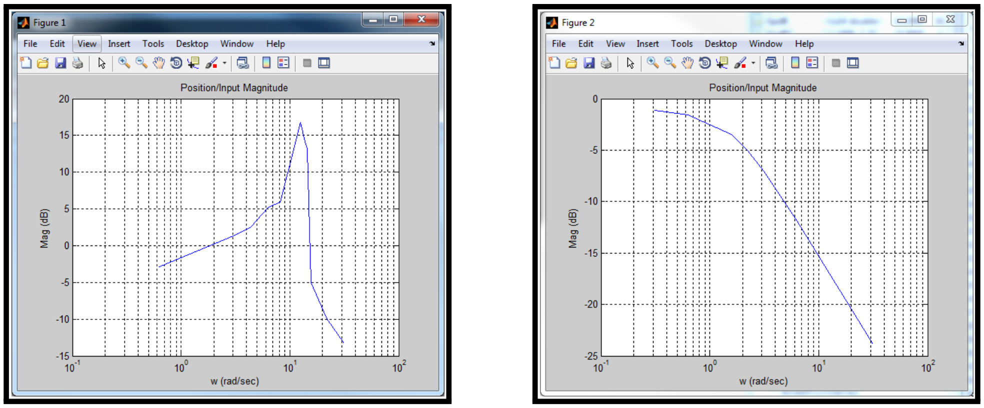
Table 7.6: Experimental Results - Overdamped w/ 10 Pennies

44. Open the *BodePlotData.m* from the MATLAB "Current Folder" location. Input the $\frac{|y|}{|r|}$ (dB) values from Table 7.1 inside the brackets of the *GpdB* variable. Add any additional frequencies to the variable *f*. The frequency variable should be arranged in numerical order, from smallest to largest.

45. Input the $\frac{|y|}{|r|}$ values from Table 7.4 inside the brackets of the *GpdB2* variable. Add any additional frequencies to the variable *f2*. The frequency variable should be arranged in numerical order, from smallest to largest.

46. Run the file to generate the two bode plots. Save them for you lab report, and leave them open. Discuss the relationships between your theoretical sketches of the Bode plot and your experimental results.

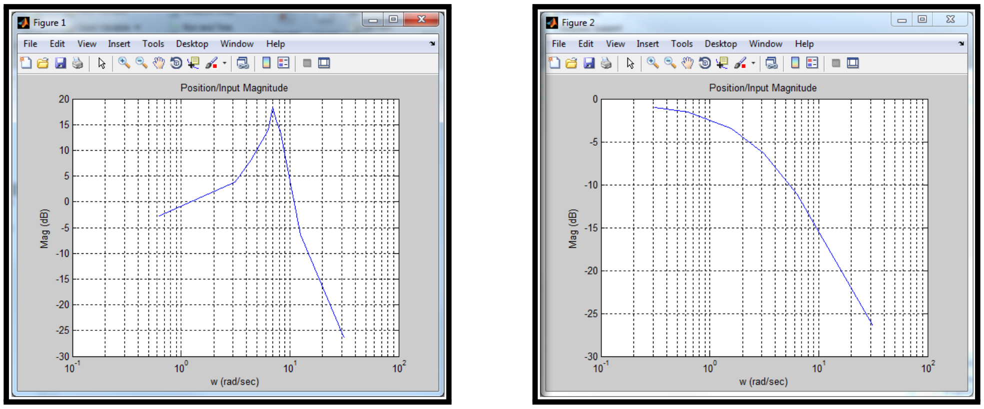
    ⋆ The frequency response data for the load with no pennies produced the bode plots shown in Figure 7.15a and Figure 7.15b.



(a) Underdamped (b) Overdamped

Figure 7.15: Bode Plots - Load with No Pennies

⋆ The frequency response data for the load with no pennies produced the bode plots shown in Figure 7.16a and Figure 7.16b.



(a) Underdamped (b) Overdamped

Figure 7.16: Bode Plots - Load with 10 Pennies

7.3.3 Exercise 3: Simulation and Experimental Comparison

In this subsection, you will use MATLAB to compute the bode plots for the motor's closed loop transfer function you found in exercise 1, using both sets of gains, and compare them to the experimental bode plots.

47. In MATLAB, using the command tf, create the closed loop motor transfer function you found and sketched in Exercise 1. Set the transfer function equal to the variable *Gcl*.

48. Create your K_m and τ_m constant variables in the workspace. Then calculate the bode plot for the gains $K = 2$ and $k_2 = 0$ on the same figure as your experimental plot, by entering the following in the MATLAB Command Window:

   - *figure(1);bodemag(Gcl,w,'--r');grid on;*
   - *legend('Experimental','Simulated','location','Northwest')*

   ⋆ The experimental response vs the simulated response for the load with no pennies, and using these gains is shown in Figure 7.17.

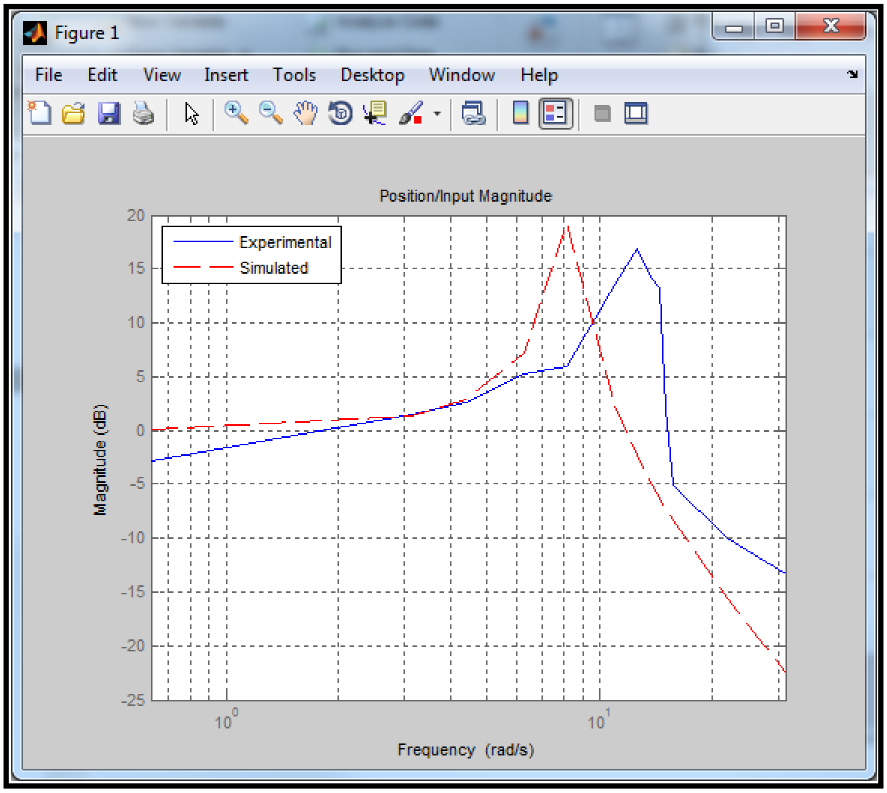


Figure 7.17: Bode Comparison No Pennies - Underdamped

   ⋆ The experimental response vs the simulated response for the load with 10 pennies, and using these gains is shown in Figure 7.18.

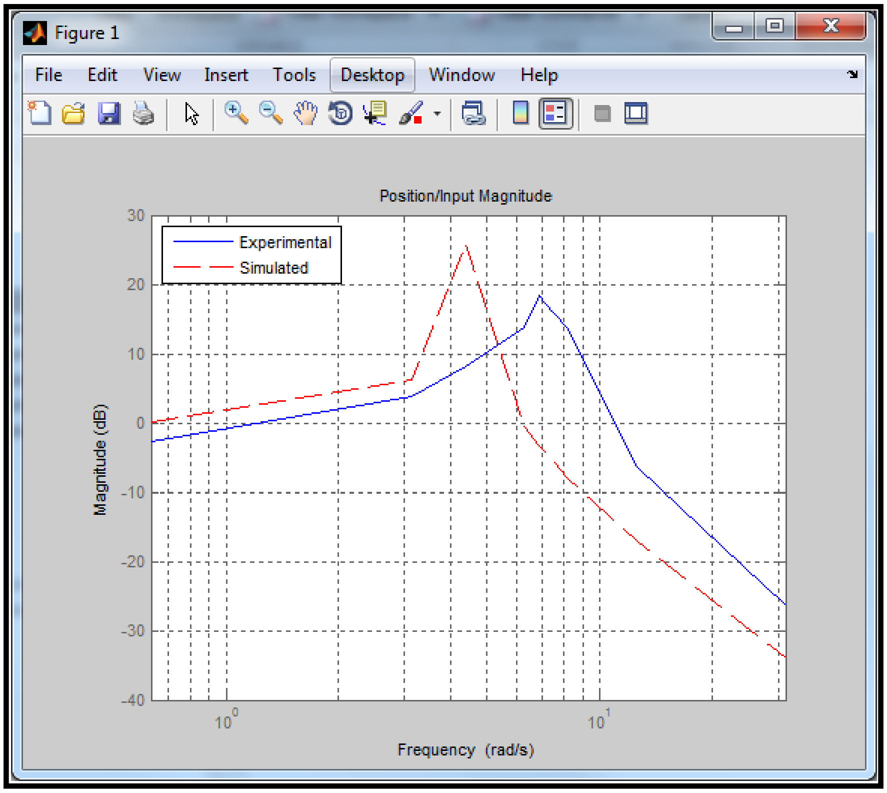


Figure 7.18: Bode Comparison 10 Pennies - Underdamped

49. Save the figure for your lab report.

50. How well do the experimental and simulated responses match? Explain any differences in bandwidth, low frequency magnitude, resonant frequency, resonant peak magnitude, etc.

   ⋆ The underdamped experimental responses, for the load with no pennies and with 10 pennies, both follow the same shape as the simulated responses. However, the experimental responses have lower magnitudes than the simulated responses at lower frequencies, and the experimental response's resonant peaks are not located at the same frequency as the simulated response's resonant peaks. This is mainly do to the nonlinearities of the motor, which are not being accounted for in the theoretical model being simulated.

51. Again, create the closed loop motor transfer function you found and sketched in Exercise 1, but this time set the transfer function equal to the variable *Gcl2*.

52. Simulate the bode plot for the gains $K = 2$ and $k_2 = 0.5$ on the same figure as

your experimental plot, by entering the following in the MATLAB Command Window:

- *figure(2);bodemag(Gcl2,w2,'--r');grid on; legend('Experimental','Simulated')*

⋆ The experimental response vs the simulated response for the load with no pennies, and using these gains is shown in Figure 7.19.

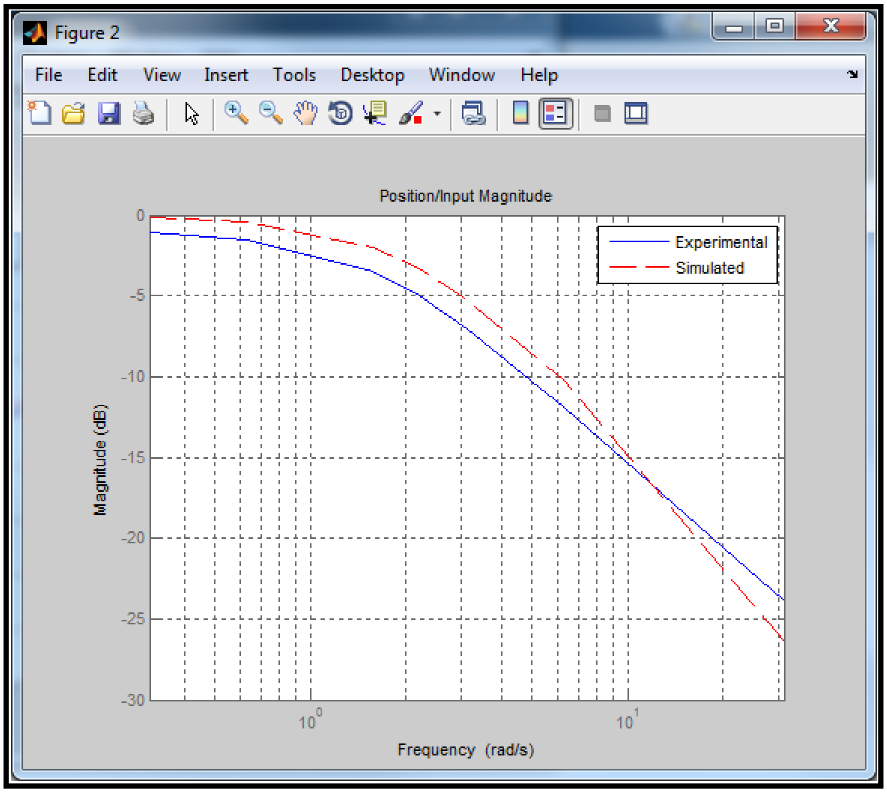


Figure 7.19: Bode Comparison No Pennies - Overdamped

⋆ The experimental response vs the simulated response for the load with 10 pennies, and using these gains is shown in Figure 7.20.

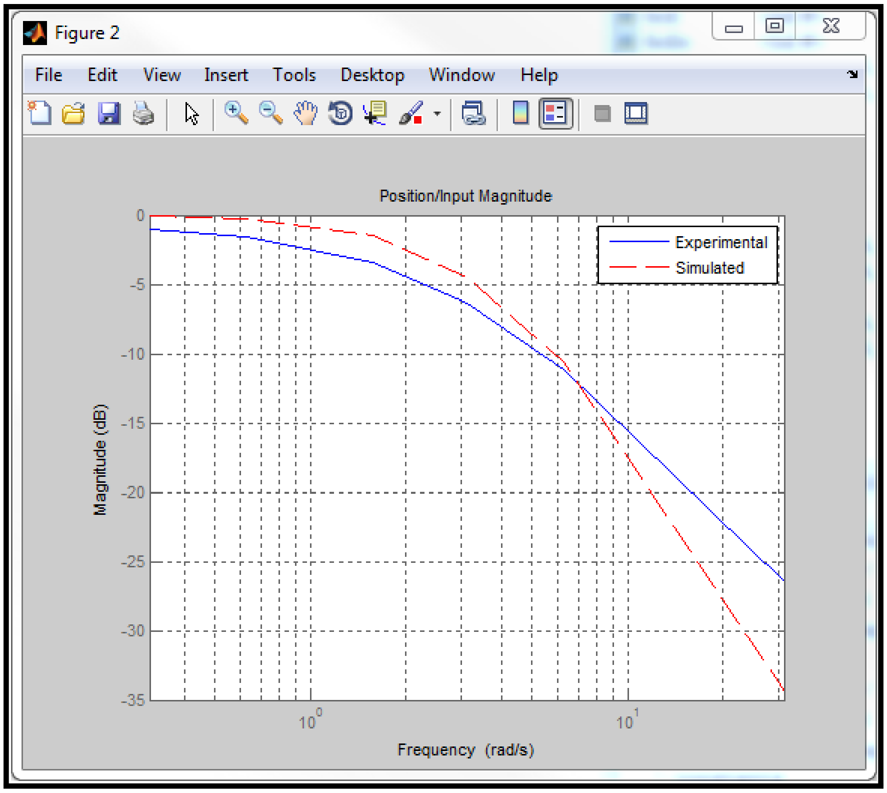


Figure 7.20: Bode Comparison 10 Pennies - Overdamped

53. Save the figure for your lab report.

54. How well do the experimental and simulated responses match? Explain any differences in bandwidth, low frequency magnitude, resonant frequency, resonant peak magnitude, etc.

   ⋆ The overdamped experimental responses, for the load with no pennies and with 10 pennies, both follow the same shape as the simulated responses. However, the experimental responses have lower magnitudes than the simulated response until the higher frequencies. The experimental responses have lower bandwidths than the simulated responses. This is also do to the nonlinearities of the motor, which are not being accounted for in the theoretical model being simulated.

7.4 Conclusion/Student Feedback

In this experiment the closed loop transfer function of a DC motor was derived. The frequency response of the closed loop model was found experimentally using various types and levels of feedback gains. Bode plots of the closed loop model were then simulated and compared to the experimental bode plots. The responses demonstrated the effects the different types and levels of feedback had on the system's response, and how well the theoretical response matched the actual response.

CHAPTER 8

OPTIMAL STATE FEEDBACK CONTROL (BALL AND BEAM)

8.1 Objective

In this experiment you will build and control a ball and beam system. Based on the equations of motion, you will build the theoretical nonlinear model in Simulink. The open loop response will be verified through simulation. Using MATLAB, you will linearize the nonlinear model, and simulate the open loop response for the linear model as well. Once the linear and nonlinear models have been verified, an optimal state feedback controller will be designed and simulated. Then you will test your optimal controller on the real system, and compare with the simulation results.

8.2 Setup

8.2.1 Required Materials

Hardware

All of the materials used in the *Sampling and Data Acquisition* experiment will be used in this experiment, except for the 3-D printed load. Additionally, the hardware listed below, and shown in Figure 8.1 and Figure 8.2 will be used.

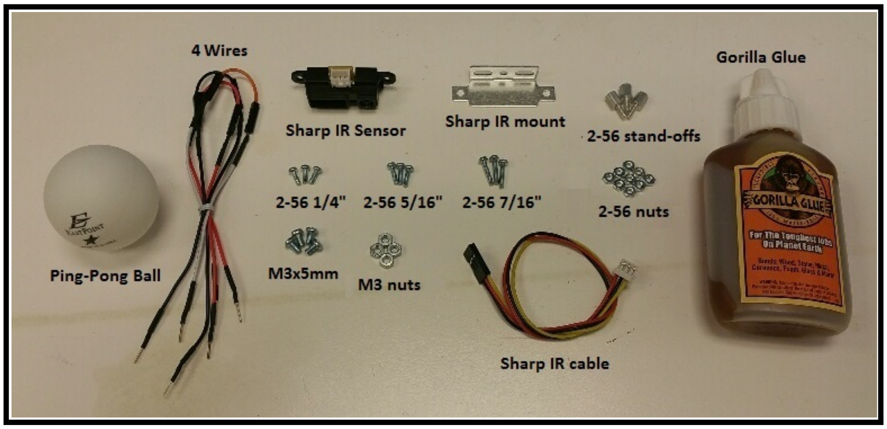


Figure 8.1: Additional Hardware

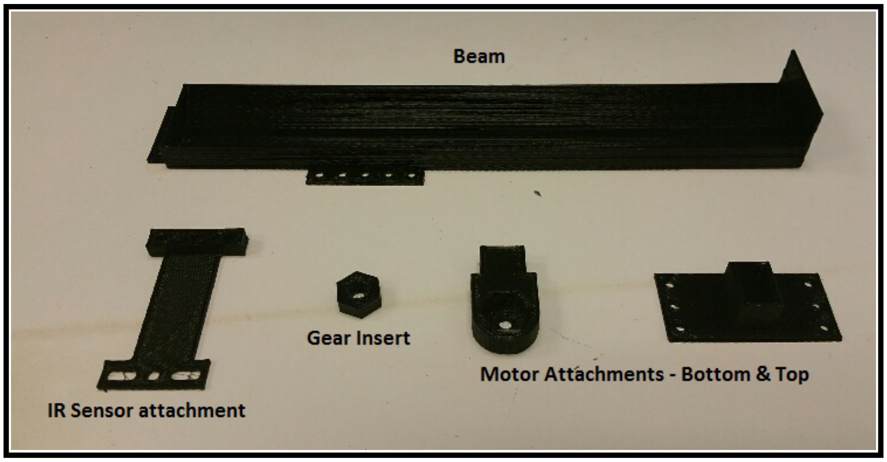


Figure 8.2: 3-D Printed Hardware

- Ping Pong Ball
- Sharp GP2Y0A41SK0F IR Sensor
- Sharp IR Mounting Bracket with 3 to 4 M3x5mm screws and nuts
- Sharp IR Cable
- 4 wires
- 9 - 2-56 Screws (3 - 1/4”, 3 - 5/16”, and 3 - 7/16”)
- 9 - 2-56 nuts
- 3 - 2-56 Stand offs
- 3-D Printed Beam
- 3-D Printed IR attachment
- 3-D Printed Gear insert
- 3-D Printed Motor attachments (Top and Bottom)
- Gorilla Glue

Software

- MATLAB/Simulink 2013a or later

  ⋆ *The steps and images related to MATLAB/Simulink for this experiment were created using MATLAB/Simulink 2013a. Therefore some steps and images may be a little different if you are not using this version. If you are in fact using a different version, make sure you know the steps for running models onto the Arduino for your version of Simulink.*

- MATLAB/Simulink files
- 3-D Printer files

Prerequisite Experiments

- Sampling and Data Acquisition

8.2.2 Software Setup

The necessary software files that are needed for this experiment can be downloaded from the Take Home Labs webpage. One method for downloading the files is shown in the steps below.

1. Open your internet browser and navigate to `thl.okstate.edu`
2. On the left side of the Homepage, select "All Experiments"
3. In the middle section of the All Experiments page select "Optimal State Feedback Control (Ball on Beam)"
4. On the Optimal State Feedback Control (Ball on Beam) page select "Software/Code" in the rightmost section. A zipfile named *Experiment_OSFC* should download.
5. Right-click the file and choose "Extract All...", or any other method of extracting files on your PC.
6. Extract this folder somewhere convenient, and remember the location. This will be the folder where all of the files and plots created for this experiment are saved.

7. Navigate back to the Optimal State Feedback Control (Ball on Beam) page, and select "3-D Printer Files" in the rightmost section.

8. Select the Optimal State Feedback Control (Ball on Beam) link to download the files. Open and 3-D print all of the ".STL" files. *Note:* Before printing each part, be sure to orient them correctly in the 3-D printing software. You should not need any support material. The correct orientations are shown in Figure 8.2, but you may have to orient the beam diagonally to make it fit on the print bed.

8.2.3 Hardware Setup

Beam Construction

9. The base hardware for this experiment is shown in Figure 8.3. If you do not have the hardware set up this way, follow the hardware setup steps provided in the *Sampling and Data Acquisition* experiment.

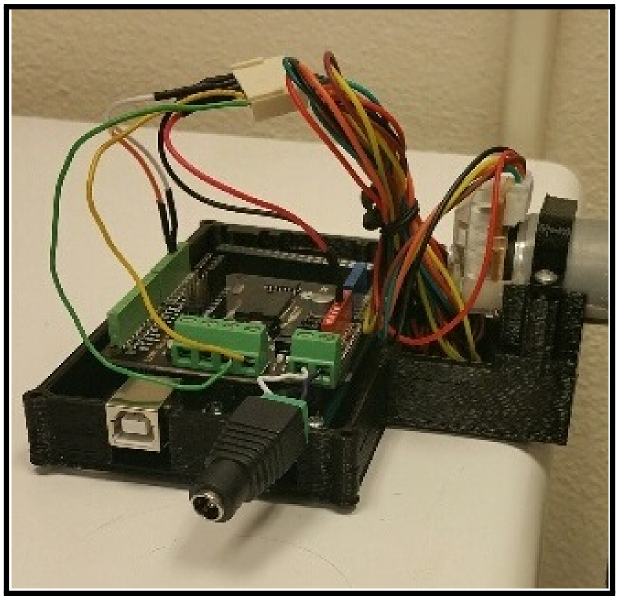


Figure 8.3: Base Assembly

10. Take the gear insert and make sure it fits on the motor's gear, as in Figure 8.4. Then remove it from the gear. *Note:* You may need to use some force to fit the gear insert onto the gear.

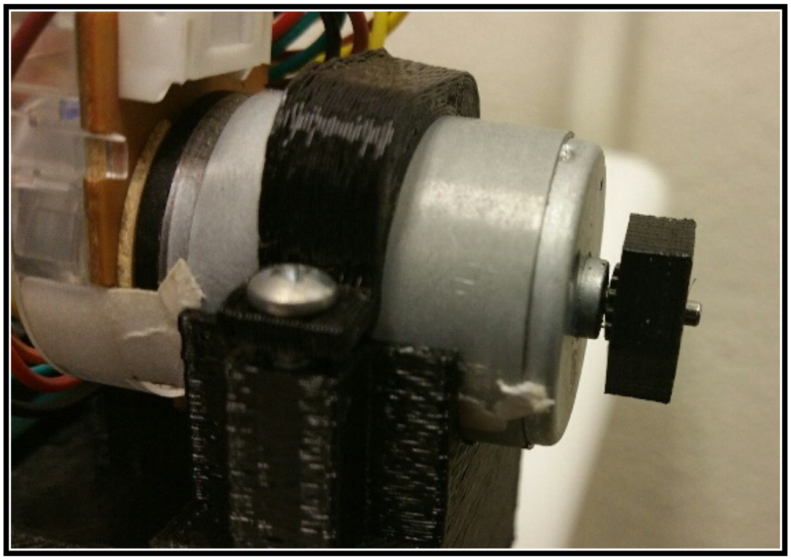


Figure 8.4: Gear Insert

11. Place the gear insert inside the bottom motor attachment, as in Figure 8.5. *Note:* You may need to sand the sides of the gear insert to make it to fit inside the attachment.

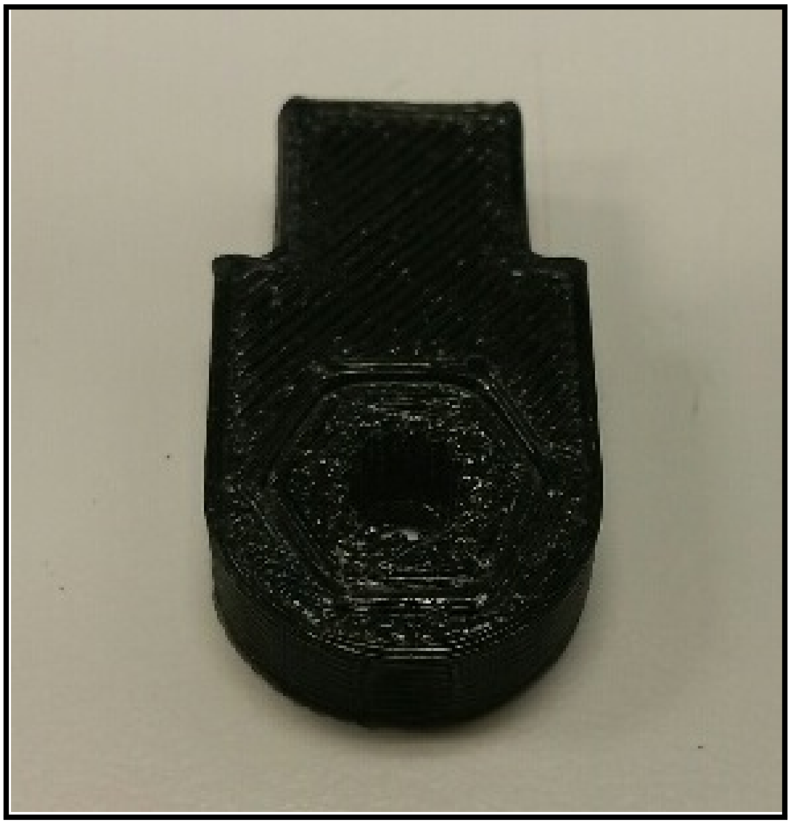
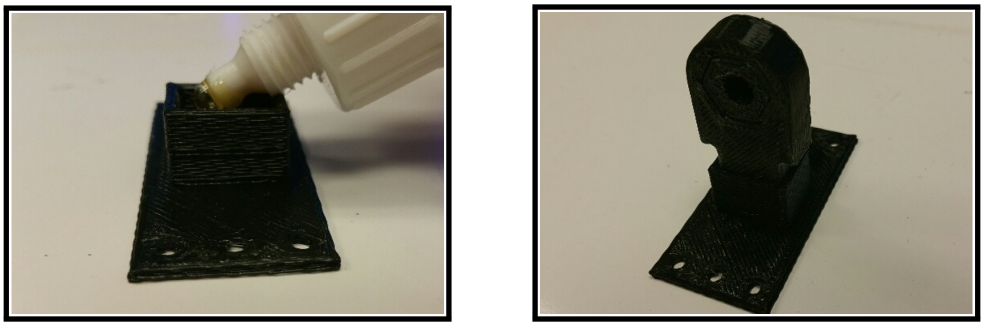


Figure 8.5: Gear Insert inside Bottom Motor Attachment

12. Place gorilla glue inside the top piece of the motor attachment, as in Figure 8.6a. Place the bottom motor attachment piece inside the top motor attachment piece, as in Figure 8.6b. Give the glue time to dry before moving the piece. *Note:* You may need to sand the sides of the bottom motor attachment piece to make it to fit inside the top motor attachment piece.



(a) Motor Top Attachment (b) Connecting Motor Attachments

Figure 8.6: Motor Attachments

13. Connect the standoffs to the end of the beam, as in Figure 8.7.

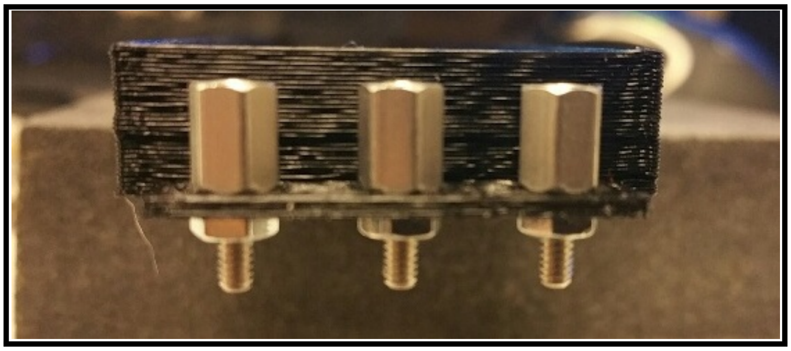


Figure 8.7: Beam Standoffs

14. Connect the IR sensor attachment to the standoffs using three 2-56 5/16" screws, as in Figure 8.8.

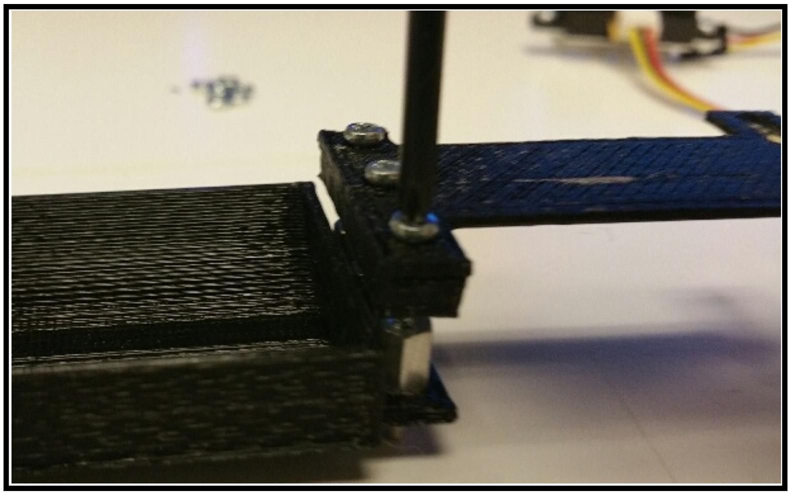
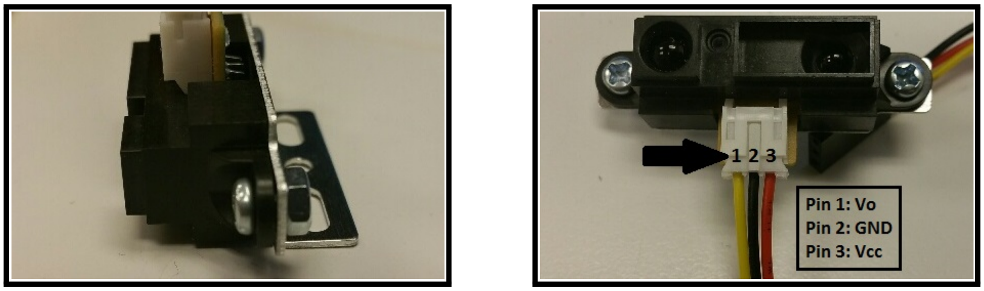


Figure 8.8: IR Attachment

15. Connect the IR sensor to the IR sensor mounting bracket using two M3x5mm screws, as in Figure 8.9a. Connect the IR sensor cable to the IR sensor, as in Figure 8.9b. *Note:* Remember the pin configuration of the IR sensor.



(a) IR Sensor Mount

(b) IR Sensor Cable

Figure 8.9: IR Sensor Connections

16. Connect the IR sensor mounting bracket to the IR attachment, using one M3x5mm screw and nut in the middle hole, as in Figure 8.10. *Note:* You can also mount the IR sensor to the attachment using the two adjustable slots, which would require using two M3x5mm screws and nuts.

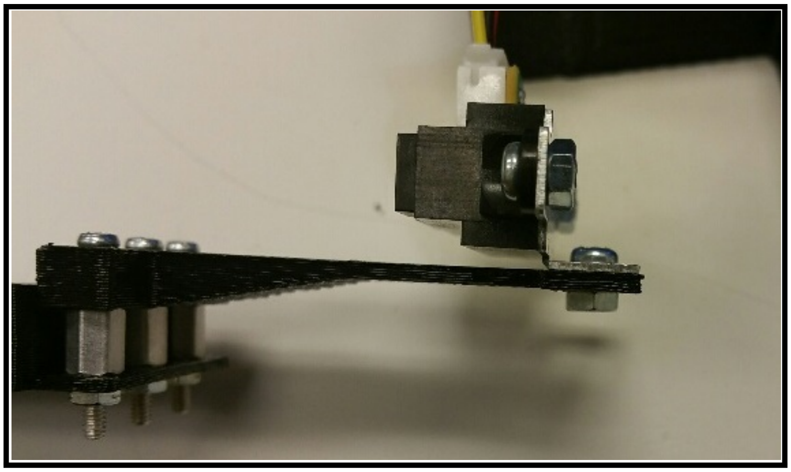


Figure 8.10: IR Connection to the Beam

17. Place the motor attachment, that you glued together, onto the motor's gear, as in Figure 8.11.

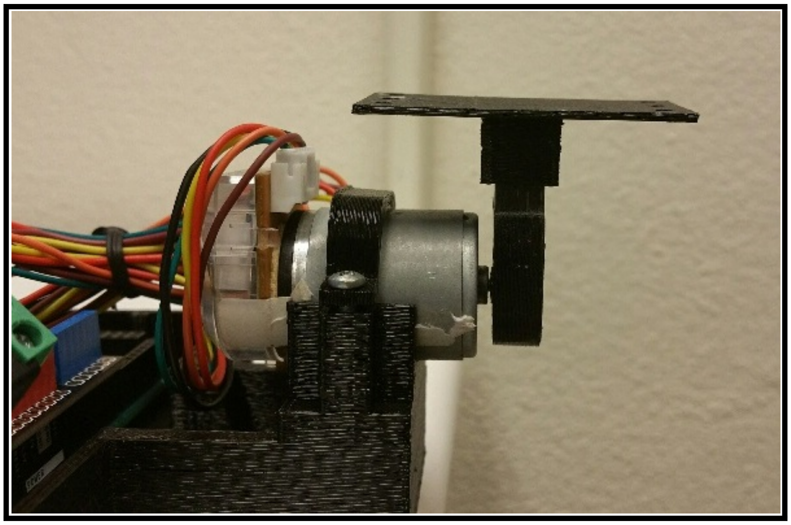


Figure 8.11: Motor Attachment to Motor

18. Take the beam and place it onto the motor attachment. Adjust the beam over the motor attachment's holes, so that the beam balances on the motor attachment, as in Figure 8.12.

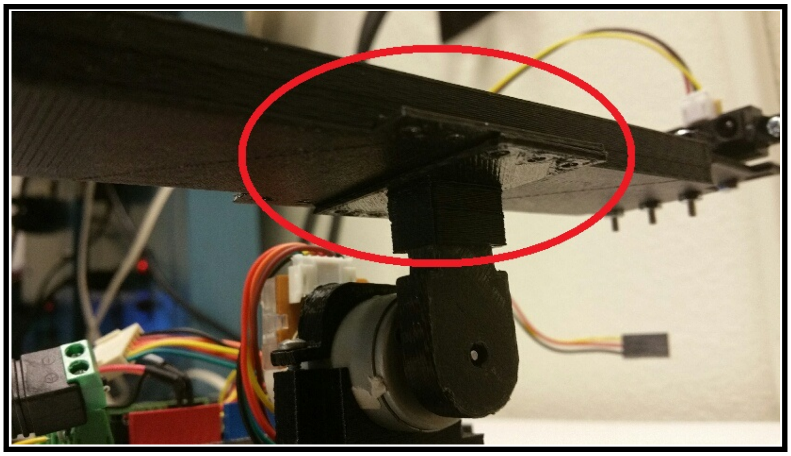
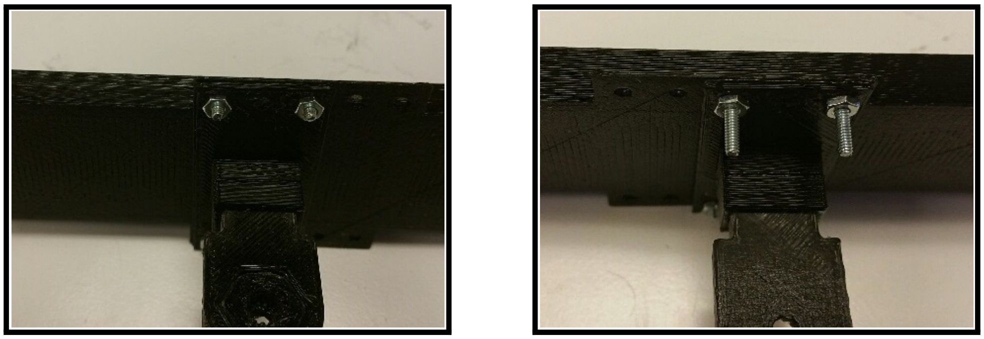


Figure 8.12: Attaching the Beam

19. When you find a position where the beam is able to balance on the motor attachment, screw the beam and motor attachment together, as in Figure 8.13a and Figure 8.13b. Use as many screws as you desire, but make sure to use the 2-56 1/4" screw on the side closest to the motor, and use the 2-56 7/16" screw on the opposite side.



(a) Shorter Screw Side

(b) Longer Screw Side

Figure 8.13: Beam Connection

20. Stop and ensure your setup resembles Figure 8.14.

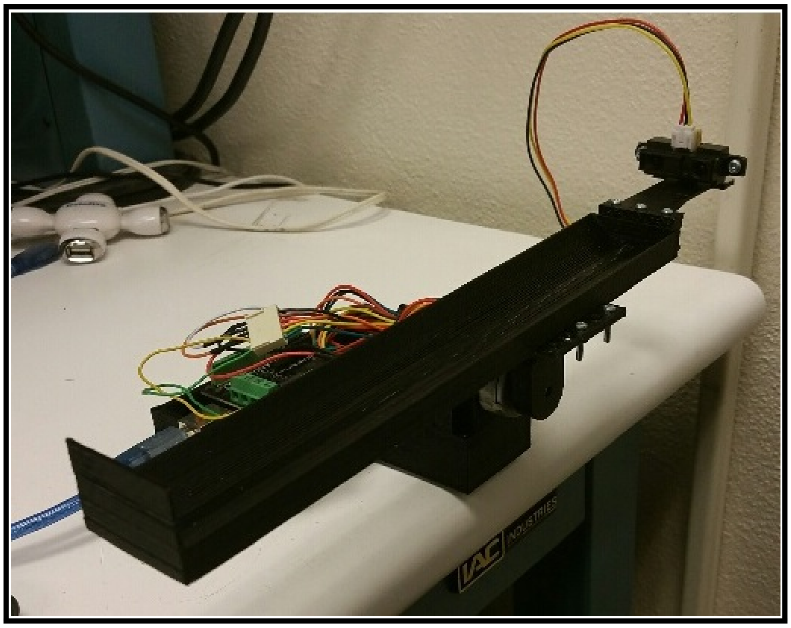


Figure 8.14: Complete Beam Construction

IR Sensor Connection

21. Place 3 wires into the end of the IR connector, as in Figure 8.15.

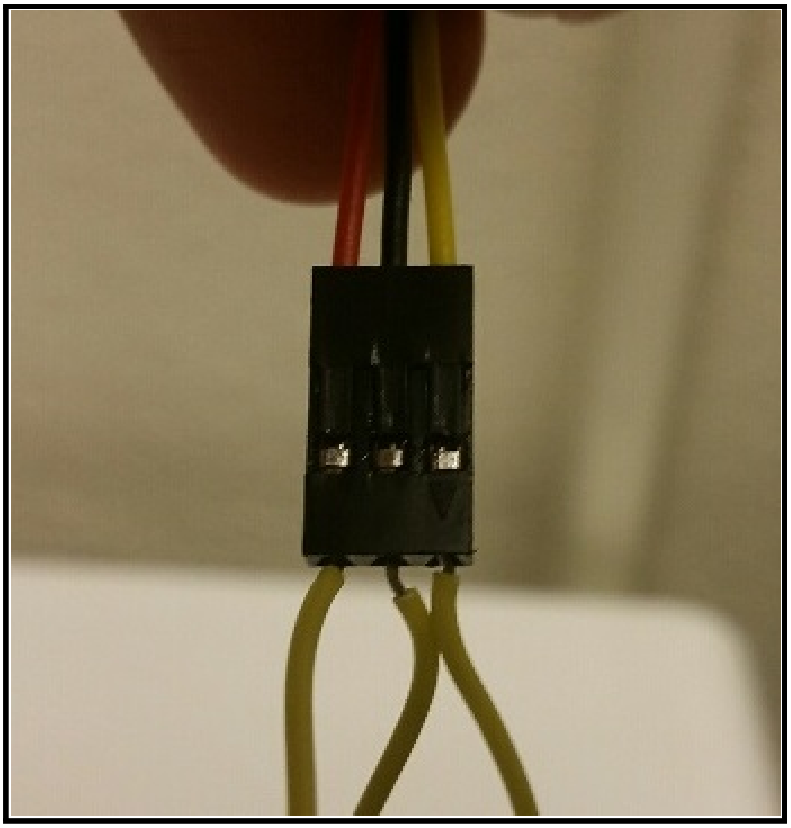
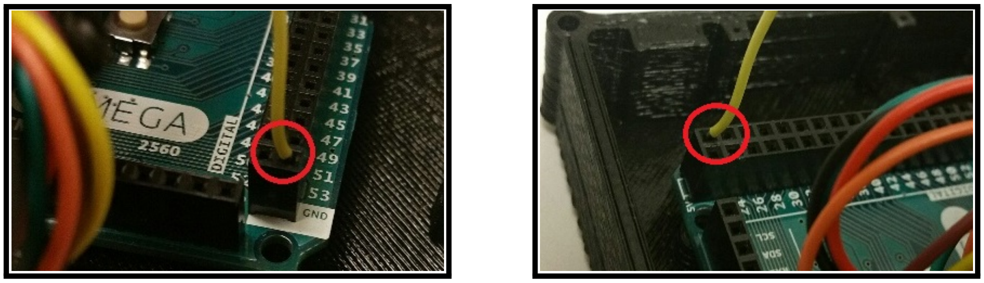


Figure 8.15: Wires for IR Connector

22. Connect pin 2 of the IR sensor to a GND pin on the Arduino, as in Figure 8.16a. Connect pin 3 of the IR sensor to a 5V pin on the Arduino, as in Figure 8.16b.



(a) GND Connection

(b) Vcc Connection

Figure 8.16: IR Sensor Connections

23. Connect pin 1 of the IR sensor to the Analog IN 0 pin of the Arduino, as in Figure 8.17.

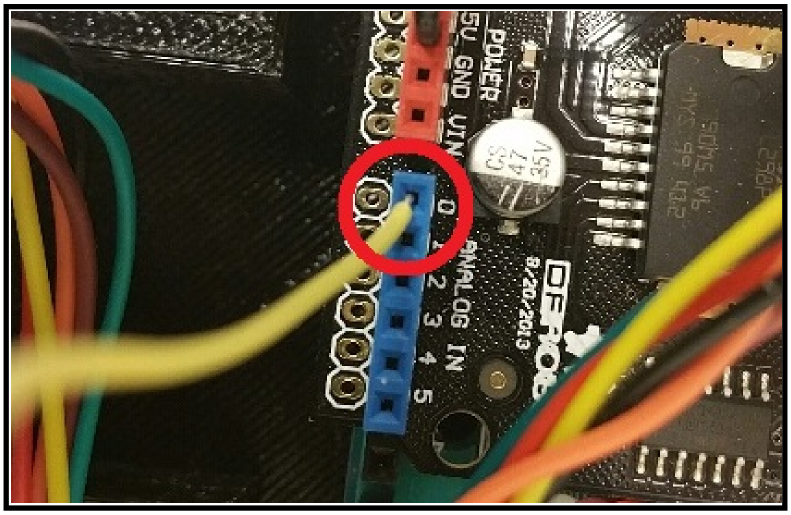
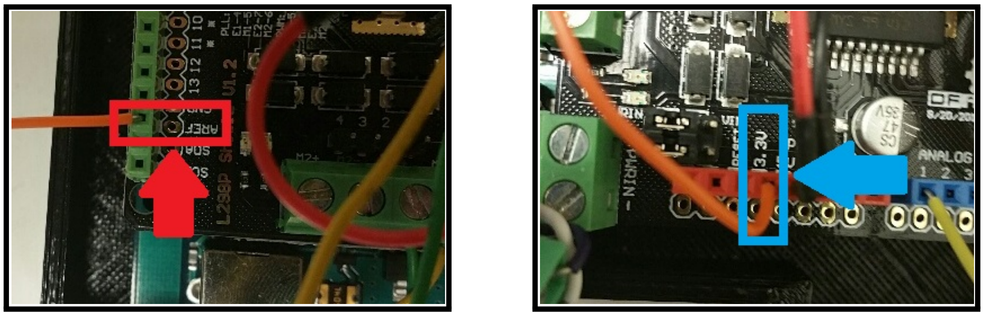


Figure 8.17: IR Sensor Vo Connection

24. Take another wire and connect it from the AREF pin on the Arduino to the 3.3V pin on the Arduino, as in Figure 8.18a and Figure 8.18b.



(a) AREF Pin

(b) 3.3V Pin

Figure 8.18: Arduino Analog Reference Voltage

25. Place the ball on the beam above the motor attachment, as in Figure 8.19.

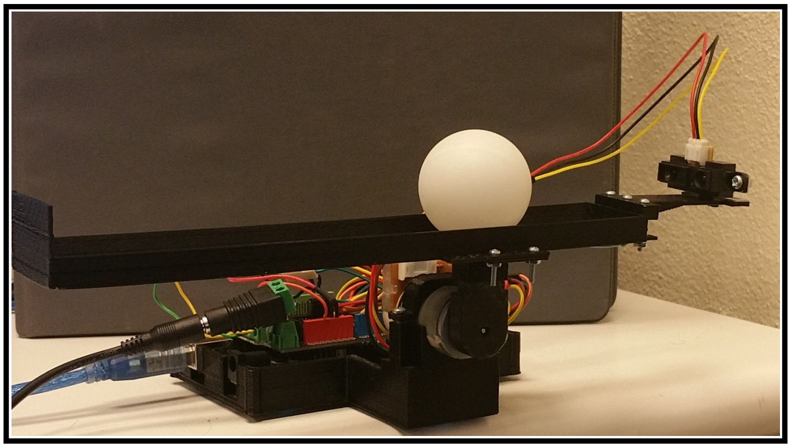


Figure 8.19: Ball and Beam

26. After adding the IR connection wires and the ping pong ball, the beam's center of mass point may be off. If this happens, you can add sticky tack to the bottom of the beam to adjust the beam's weight on either end, as in Figure 8.20.

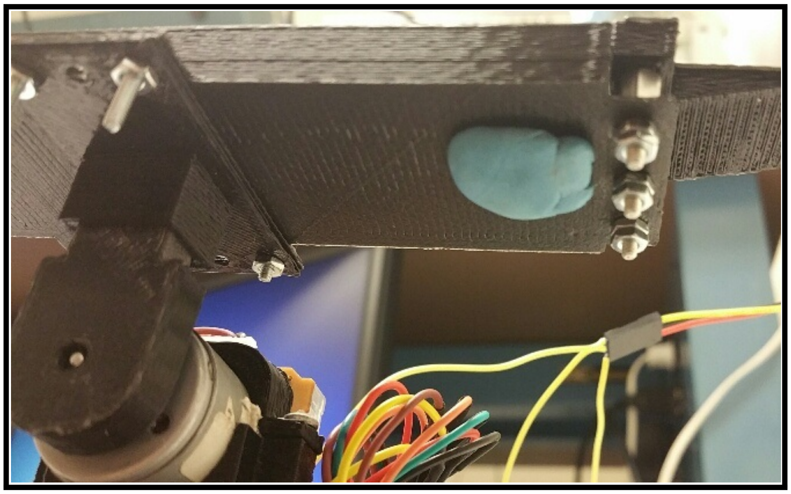


Figure 8.20: Adding Sticky Tack

27. If you added sticky tack to make the beam balance, with the ball above the motor attachment, the final assembly should look like Figure 8.19.

8.3 Experimental Procedures

In this section, you will build a Simulink model of the system, and simulate the open loop response to verify performance. Then you will develop a linear model of your system, simulate the open loop response, and compare the open loop response of the linear model to the open loop response of the nonlinear model. After verifying the open loop responses for both the linear and nonlinear model, you will design an optimal state variable feedback controller for the linear model. You will simulate the linear and nonlinear system responses with the added optimal state variable feedback controller. Then you will use the optimal state variable feedback controller to control the physical ball and beam system you built, by using the Arduino and Simulink.

8.3.1 Exercise 1: System Model

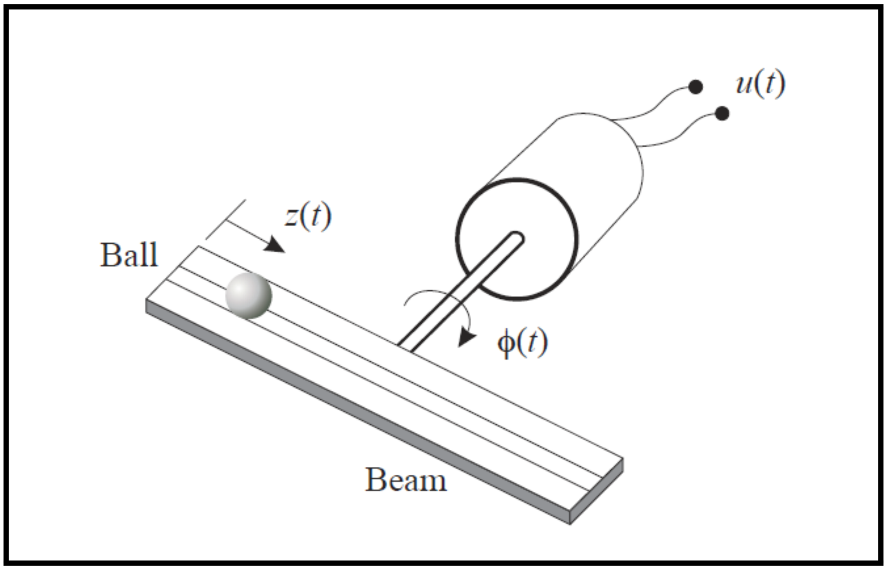


Figure 8.21: Ball and Beam Model

The equations of motion for the Ball and Beam are shown below.

$$\dot{x_1} = \frac{\tau - mgcos(x_2)x_4 - 2mx_1x_3x_4}{J_{beam} + J_{ball} + mx_4^2} \tag{8.1}$$

$$\tau = \frac{K_t}{R_a}(u - K_b x_1) \tag{8.2}$$

$$\dot{x_2} = x_1 \tag{8.3}$$

$$\dot{x_3} = \frac{mx_1^2x_4 - mgsin(x_2)}{\frac{J_{ball}}{r^2} + m} \tag{8.4}$$

$$\dot{x_4} = x_3 \tag{8.5}$$

where

$$x_1 = \text{Beam Velocity}$$
$$x_2 = \text{Beam Position}(\phi)$$
$$x_3 = \text{Ball Velocity}$$
$$x_4 = \text{Ball Position}(z)$$
$$u = \text{Motor Voltage}$$
$$\tau = \text{Torque}$$

The constants are

$$K_b = 0.01 \text{ (Back EMF)}$$
$$K_t = 0.01 \text{ (Torque Constant)}$$
$$R_a = 30 \text{ ohms (Armature Resistance)}$$
$$J_{beam} = 6{\times}10^{-5} \text{ Kg-}m^2 \text{ (Beam Inertia)}$$
$$m = 0.0027 \text{ Kg (Ball Mass)}$$
$$r = 0.02 \text{ m (Ball Radius)}$$
$$J_{ball} = 7.2\ {\times}10^{-7} \text{ Kg-}m^2 \text{ (Ball Inertia)}$$
$$g = 9.81\ m/s^2 \text{ (Gravitational Constant)}$$

28. Make a Simulink model of the system.

    ⋆ The model of the Ball and Beam is shown in Figure 8.22.

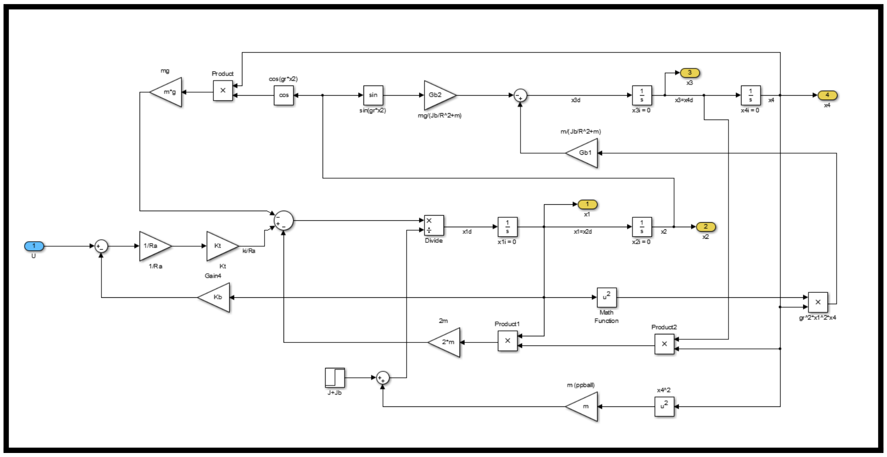
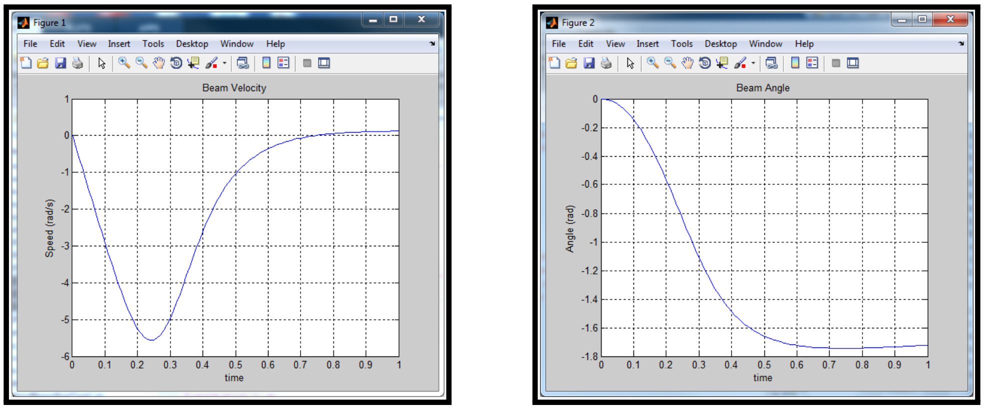


Figure 8.22: Ball and Beam Simulink Model

29. Simulate the open loop response for 1 second, using an initial ball position of *0.1* m.

30. Plot each system state vs time.

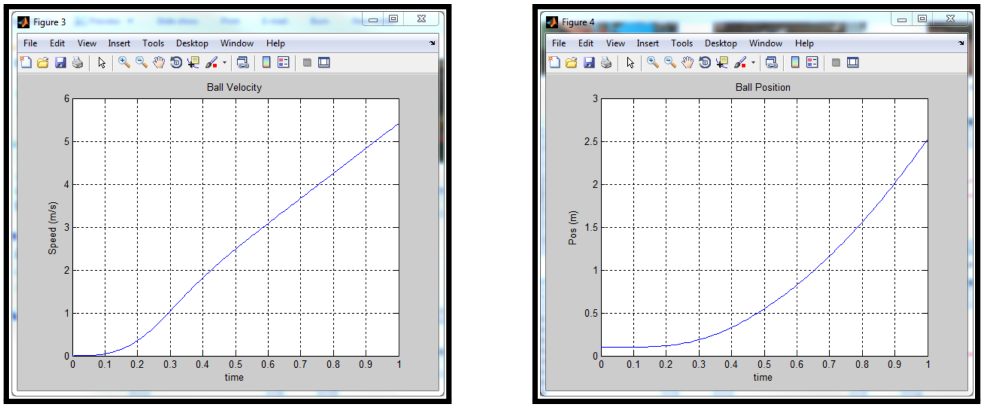
    ⋆ The open loop simulation results are provided below.



(a) Beam Velocity

(b) Beam Position

Figure 8.23: Nonlinear System Open Loop Response



(a) Ball Velocity

(b) Ball Position

Figure 8.24: Nonlinear System Open Loop Response

31. Is the response what you expected? Explain.

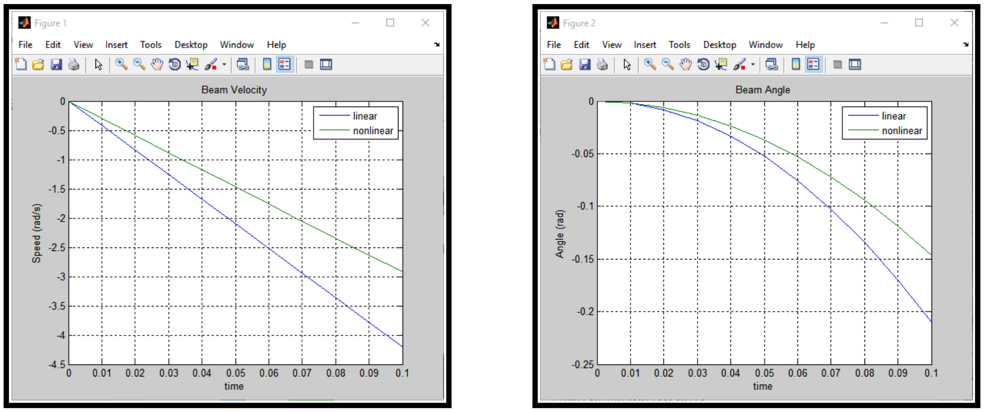
    ⋆ The results provided in the previous step verify that the open loop model is correct. The beam will fall over when the ball is not at the equilibrium point, which is indicated by the results for the beam angle. The ball would fall off of the beam and the position would become large as shown in the ball position results. The beam would move to an angle of approximately $\frac{\pi}{2}$.

8.3.2 Exercise 2: System Linearization

32. Create a linear model of the system using the MATLAB command "linmod". Linearize the model about the equilibrium point where all states equal 0.

33. Simulate the open loop response of the linear model, for 0.1 second and using initial ball positions of *0.1*m and *0.01*m. Compare with the responses of the nonlinear Simulink model. Plot each state.

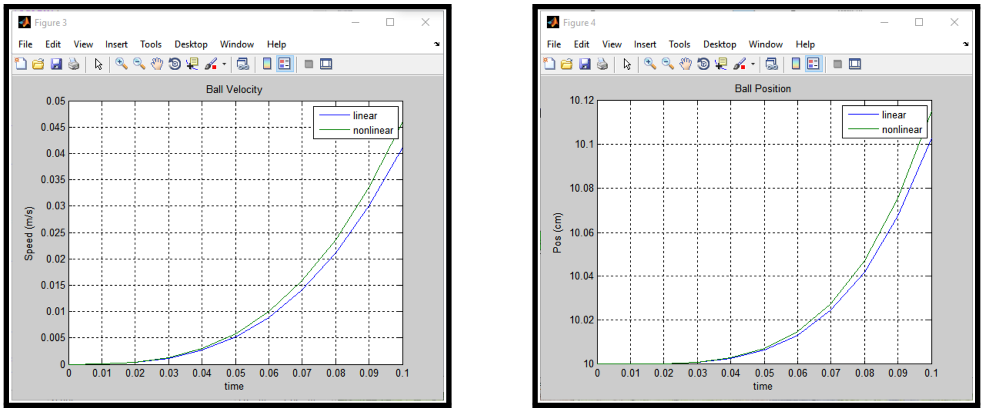
    ⋆ Comparisons between the linear and nonlinear open loop responses are provided below.



(a) Beam Velocities

(b) Beam Positions

Figure 8.25: Open Loop Response Comparisons ($x_4(0) = 0.1$m)

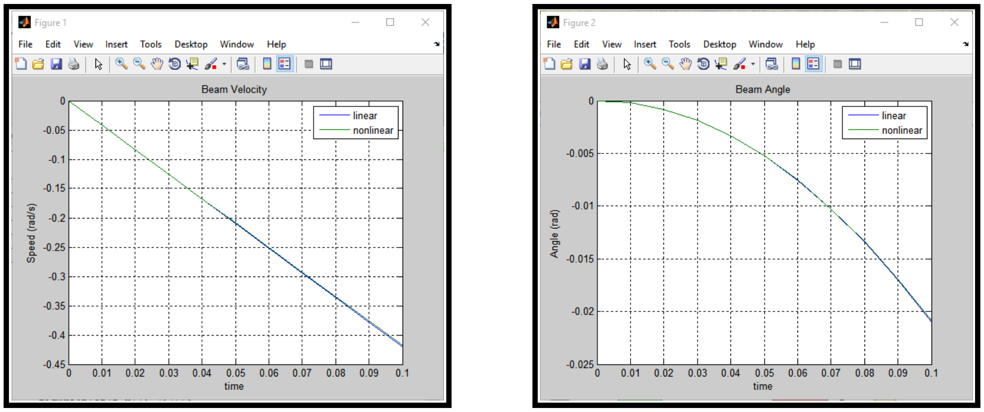


(a) Ball Velocities

(b) Ball Positions

Figure 8.26: Open Loop Response Comparisons ($x_4(0) = 0.1$m)

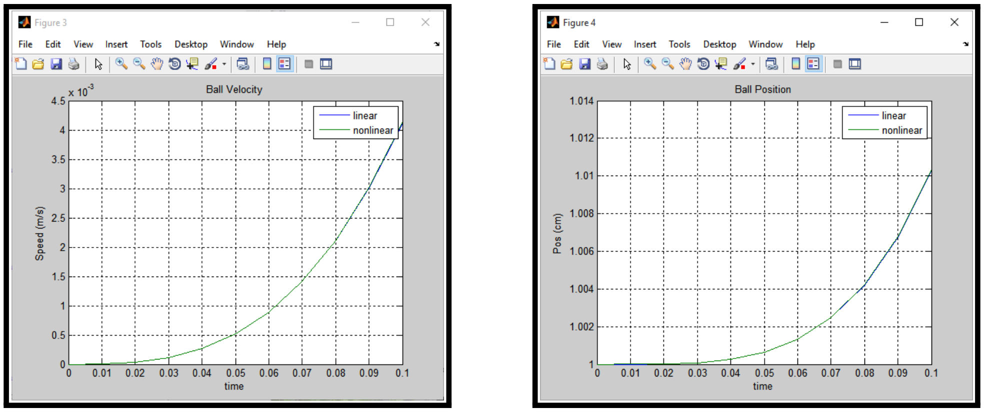
⋆ The open loop simulation results are provided below.



(a) Beam Velocities

(b) Beam Positions

Figure 8.27: Open Loop Response Comparisons ($x_4(0) = 0.01$m)



(a) Ball Velocities

(b) Ball Positions

Figure 8.28: Open Loop Response Comparisons ($x_4(0) = 0.01$m)

34. How well do the linear model's responses resemble the nonlinear model's responses? How do they differ as the initial condition moves further away from the equilibrium point?

    ⋆ The linear and nonlinear models perform similarly for some period of time, then start to deviate from one another. From the results of the two simulations, the linear and nonlinear responses deviate from each other sooner as the initial condition is moved further away from the equilibrium point.

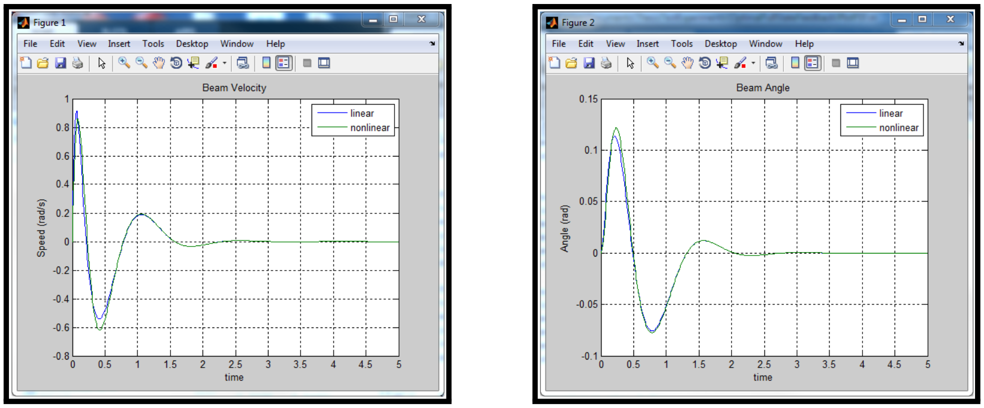
8.3.3 Exercise 3: Optimal State Variable Feedback - Simulation

35. Assuming there is no noise in the system, and all states are measurable, design an optimal state variable feedback controller for your linear ball and beam model using MATLAB's "lqr" command. You will need to select the weighting matrices in the performance index. Your intial settings are not critical, as you can make changes later.

36. Simulate the linear and nonlinear closed loop systems for 5 seconds, with the added controller. Use an initial ball position of *0.1*m.

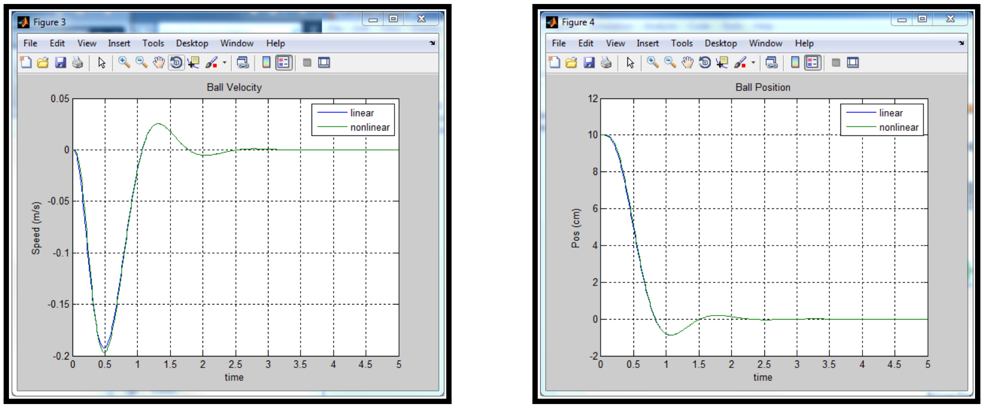
37. How does the response for the linear model differ from the response for the nonlinear model?

    ⋆ Setting $Q = I$ and $R = 1$ produced an unstable response, and setting $Q = I$ and $R = 0.1$ also produced an unstable response. The results for $Q = I$ and $R = 0.05$ are shown in the figures below.



(a) Beam Velocities (b) Beam Positions

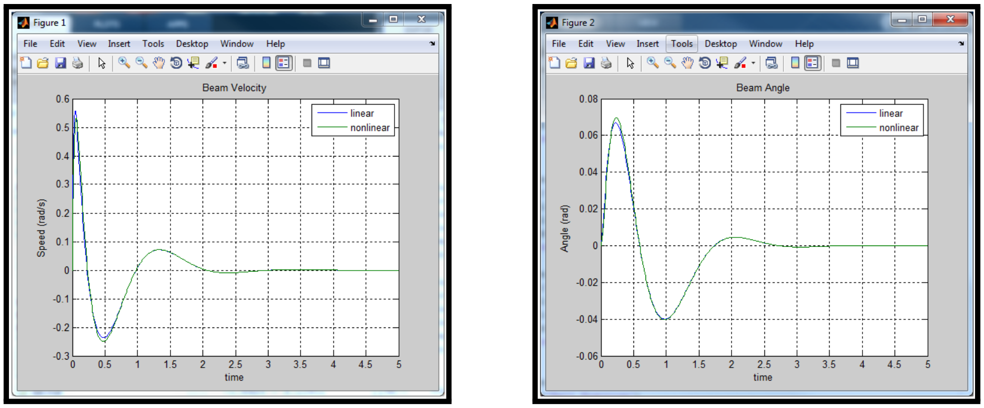
Figure 8.29: Closed Loop Response Comparisons - Beam



(a) Ball Velocities (b) Ball Positions

Figure 8.30: Closed Loop Response Comparisons - Ball

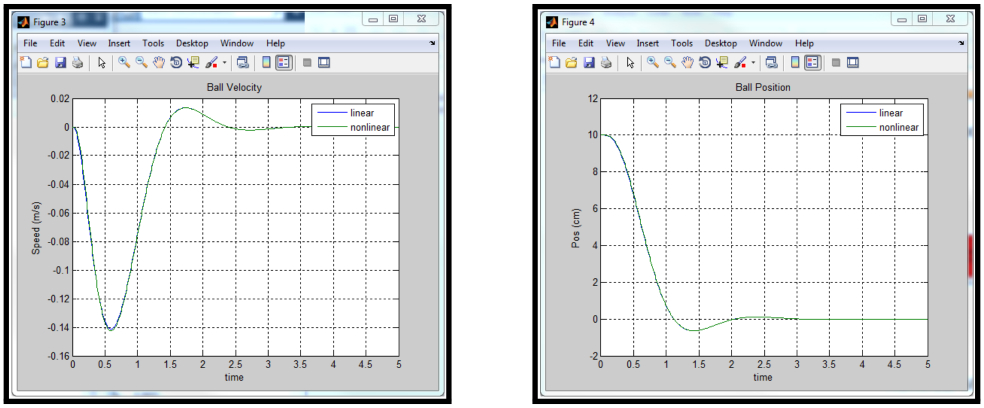
⋆ The results for $Q = I$ and $R = 0.01$ are shown in the figures below. These are the values that are used in my experimental closed loop response.



(a) Beam Velocities

(b) Beam Positions

Figure 8.31: Closed Loop Response Comparisons - Beam (2)



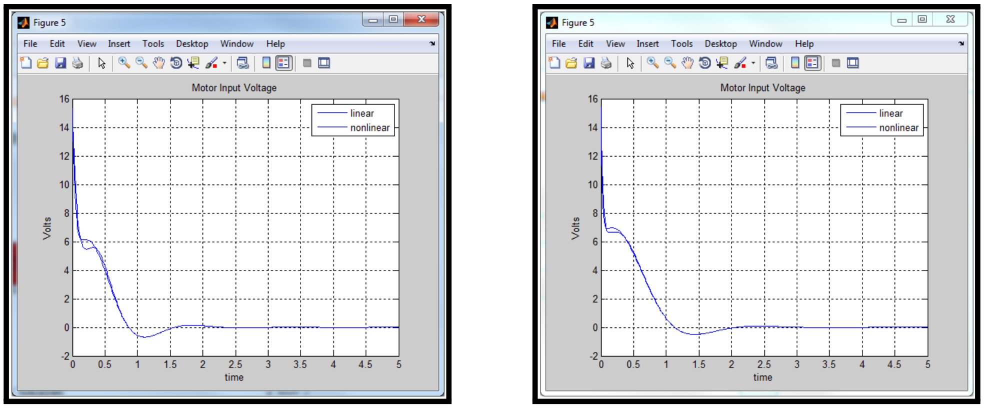
(a) Ball Velocities

(b) Ball Positions

Figure 8.32: Closed Loop Response Comparisons - Ball (2)

38. Do the input voltages for each response seem reasonable?

⋆ The input voltages of the linear and nonlinear closed loop model using state gains found with $Q = I$ and $R = 0.05$ are shown in Figure 8.33a. The input voltages of the linear and nonlinear closed loop model using state gains found with $Q = I$ and $R = 0.01$ are shown in Figure 8.33b. These are the unsaturated voltages, and both sets of gains produce reasonable voltages since the input voltage being used is 12 volts. Although the results being shown are the unsaturated voltages, a saturation block is placed in the simulation to ensure the response would still perform correctly for the rated power supply voltage that will be used in the experiment.



(a) Input Voltages - R = 0.05

(b) Input Voltages - R = 0.01

Figure 8.33: Closed Loop Response Comparisons - Input Voltages

39. If needed, make any adjustments to your controller design by changing the weighting matrices to provide a fast response without exceeding physical limitations.

40. If all of the states were still exactly measurable, but there were disturbances in the system, how would the optimal gains change? Explain.

    ⋆ The gains would be the same. Since each state of a system is measurable, the same optimal gains would be produced for both the deterministic and stochastic case.

8.3.4 Exercise 4: Optimal State Variable Feedback - Experiment

41. Open MATLAB and set the MATLAB "Current Folder" location to the *Experiment_OSFC* folder you extracted in the software setup section. The files *SerialPlotData.m*, *OSFC_Constants.m*, and *OSFCExpModel.slx* should show in the "Current Folder" section.

42. Open *OSFCExpModel.slx*. The model should now resemble the model in Figure 8.34.

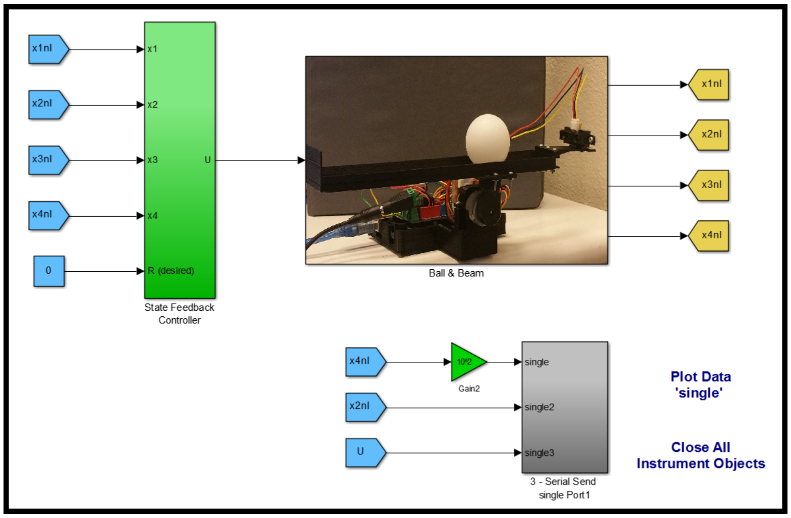


Figure 8.34: Experimental Model

43. Open the *OSFC_Constants.m* file, and add your optimal gain values into the variable “K”.

44. Run ▶ the file.

45. Set up the system so that it is balanced, with the beam parallel to the floor, and with the ball at the center of mass point, as in Figure 8.35.

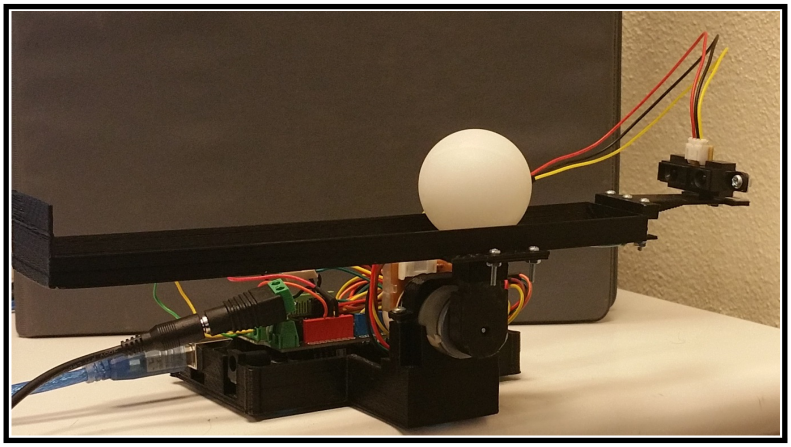


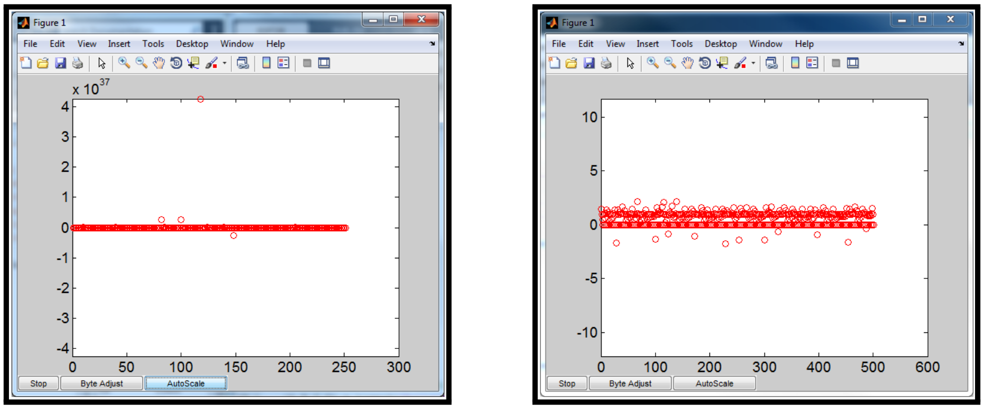
Figure 8.35: Ball and Beam

46. Connect the Arduino to the PC, but do not plug in the power supply. Download the Simulink model to the Arduino. The model will already be set up to run on the Arduino. If you receive an error about being unable to connect to the Arduino, you may need to set the COM port manually in the Configuration Parameters ⚙.

47. Once the model is successfully downloaded to the Arduino, select the "Plot Data 'single'" text.

48. In the pop-up window, change the COM port to the port your Arduino is using, and set the number of samples to plot to *500*. Press "OK" to start plotting the data. The beam position, ball position, and input voltage are all being sent over the serial port, so there will be three plots showing in the figure.

49. Press "Autoscale". The beam position and ball position should be close to zero initially, and if the data is in sync, a small input voltage may be showing in the plot. If the data looks like Figure 8.36a, it is out of sync so press the "Byte Adjust" button. After the out of sync data passes, press "Autoscale" again. When the data becomes in sync it will be similar to Figure 8.36b. If your plot does not look similar to this, press "Byte Adjust" and "Autoscale" a few more times, if needed. After a few attempts, if the data is still out of sync, press "Stop" and close the plot window. Initiate the plotting process again by selecting the "Plot Data 'single'" text and entering the number of samples to plot. Repeat this step until the data is correct. Additional techniques for troubleshooting the serial data can be found in steps 79-82 of the *Sampling and Data Acquisition* experiment.



(a) Out of Sync Data

(b) In Sync Data

Figure 8.36: Experimental Serial Data

50. Once the data is in sync, ensure that the initial ball position is close to 0. If the ball position is not close to 0, rotate the IR sensor left or right, as in Figure 8.37, until the ball position is close to 0. After adjusting the sensor, be sure to tighten the screw again, and ensure the system is still balanced.

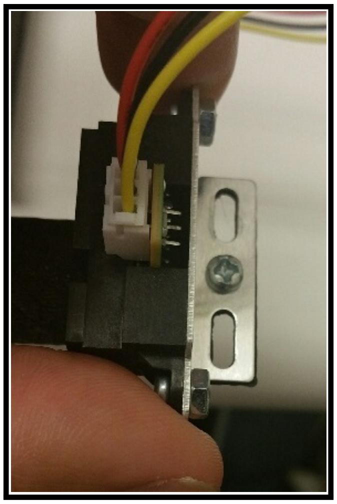


Figure 8.37: Adjust IR Sensor

51. Plug in the power supply.

52. The beam will likely not move since the ball is in the desired position initially, so lightly tap the ping pong ball in either direction. Try not to block the IR sensor as it may read the distance to your hand instead of the ball. *Note:* Tapping the ball with too much force may cause it to bounce off either end of the beam, which may cause the system to become unstable.

53. If the ball does not balance, press "Stop" and redo your optimal state variable feedback simulation using different choices of the performance index. Repeat this until you are satisfied with the response.

54. Press "Autoscale" once the ball and beam start moving, if the values are not visible. Let the experiment run until the ball comes to rest after tapping it in either direction. Do this a few times.

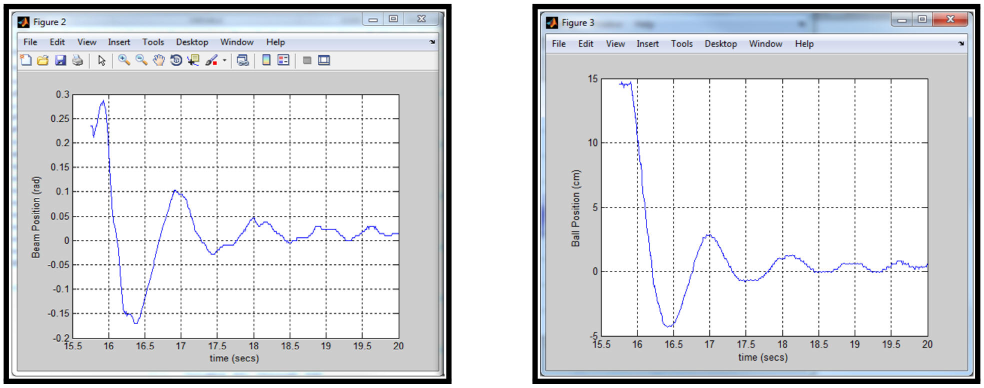
55. Manually move the ball to the longer end of the beam, and then let the ball go. Press "Stop" once the ball balances.

56. After pressing stop in the serial plot window, open the *SerialPlotData.m* file in the MATLAB Current Folder section.

57. Press Run ▶.

58. Describe the response of the system. Does the ball settle at the desired position exactly every time? If not, explain possible reasons for this.

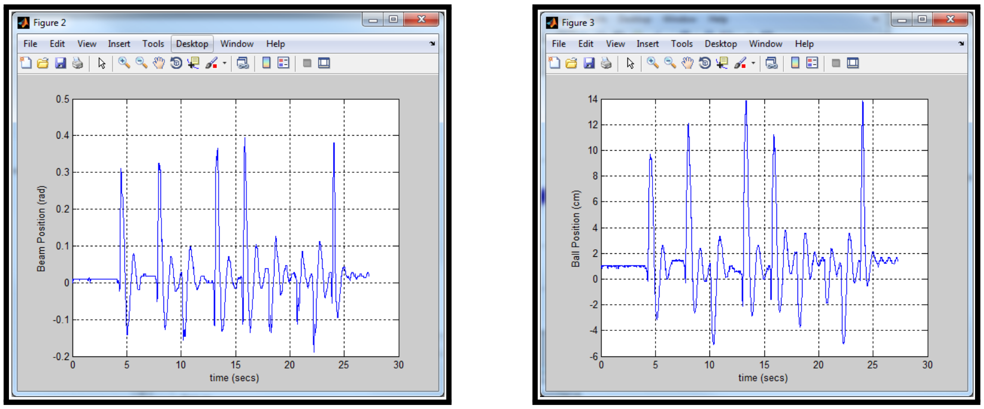
- ⋆ The experimental results for holding the ball at the end of the beam are shown in Figure 8.38a and Figure 8.38b. The experimental results, while tapping the ball in either direction, are shown in Figure 8.39a and Figure 8.39b.



(a) Beam Position (1)

(b) Ball Position (1)

Figure 8.38: Experimental Output Positions (1)



(a) Beam Position (2)

(b) Ball Position (2)

Figure 8.39: Experimental Output Positions (2)

- ⋆ For both cases, the system responds as it should. The ball balances well every time, but not in the exact desired location. Potential causes for this are: the model that is being simulated is not exactly the same as the real model, since there are frictions, design differences, and other factors not being taken into consideration.

59. Run the simulation again, but before plugging in the power, set the ball position close to the initial position used in the simulations (0.1m). Keep the beam

around 0 radians, and hold it as steady as possible. An example of this is provided in Figure 8.40.

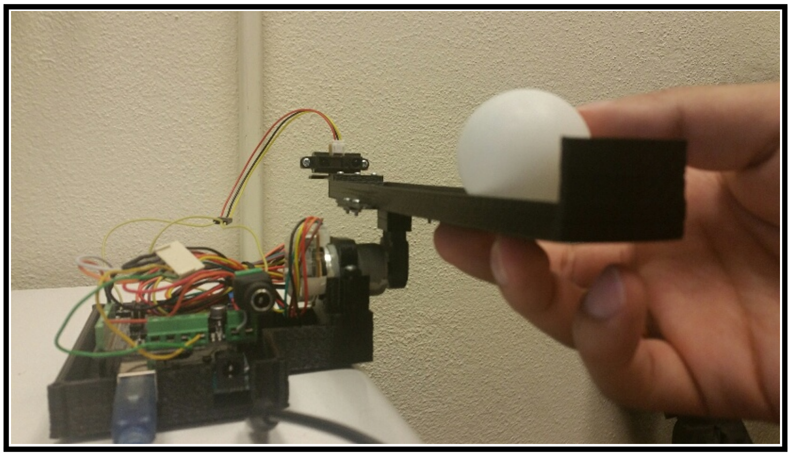
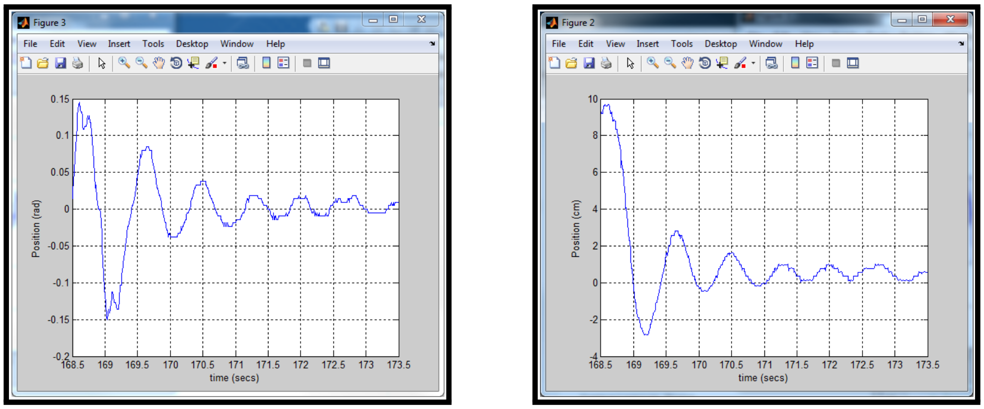


Figure 8.40: Ball Position Example

60. Plot and compare the output positions with the simulation results.

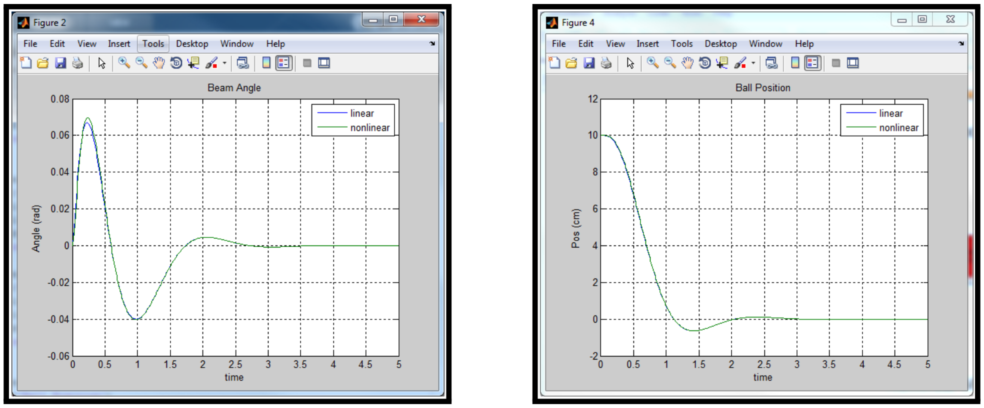
    ⋆ The experimental results while holding the ball at 0.1m initially are shown in Figure 8.41a and Figure 8.41b. The simulation results, found in Exercise 2, are shown in Figure 8.42a and Figure 8.42b.



(a) Beam Position (3)

(b) Ball Position (3)

Figure 8.41: Experimental Output Positions (3)



(a) Beam Position

(b) Ball Position

Figure 8.42: Simulation Output Positions

⋆ The general shape of the experimental and simulation outputs follow each other. Noticeable differences are: the experimental response shows more oscillations and does not settle exactly at the desired positions. Potential causes for this are: the model that is being simulated is not exactly the same as the real model since there are frictions, design differences, and other factors not being taken into consideration.

8.4 Conclusion/Student Feedback

This experiment consisted of designing an optimal state feedback controller for a ball and beam system. Before testing anything on the actual system, the open loop response was verified for the nonlinear model. The linear model was found from the nonlinear model using MATLAB, and the open loop response for the linear model was found and compared to the nonlinear open loop response. An optimal state feedback controller was designed and simulated for the linear and nonlinear models. Then the controller was tested on the actual system and compared to the simulations.

CHAPTER 9

CONCLUSIONS

In this thesis, the limited availability of laboratory experiments for engineering courses was discussed. Laboratory experiments are a hands-on learning tool that help students understand theoretical concepts, thus take home labs were designed to provide an affordable, small, simple, and adaptable solution. Existing options were analyzed and used to create a complete solution.

The Take Home Labs presented in this paper were designed around the Arduino, which is an affordable open source microcontroller, and MATLAB/Simulink. This hardware software combination is one of the major factors in the Take Home Lab's simplicity and low cost. Simulink's support for the Arduino allows for the experiments to be designed or modified by students without being strong programmers. The ability to access the functions of a power microcontroller without the need to do much programming is the key to making the Take Home Labs flexible and potentially useful for many engineering courses. There are other lab kits that combine the use of the Arduino and MATLAB/Simulink, but they are mainly confined to one system, or one course. The contribution of 3-D printing in this report adds the ability to design systems, that are used with the Arduino and MATLAB/Simulink, in a much more flexible manner. The website adds another element to the overall framework because it makes the experiments available to anyone. The student handouts, hardware parts, and software files are all provided and easily accessible because of the website. The website also allows for others to contribute by submitting comments, asking questions, or submitting new experiments. Control systems labs that can be used within a course have been around for a long time. In the past, these labs needed specific lab space, the systems were large, and they were confined to one or two courses. As technology has evolved, the size, price, and accessibility of these types of labs has improved, but the ability to use these lab kits for many courses has been limited. The Take Home Labs in this report have provided an overall framework design that is small, affordable, simple, and adaptable, which has not been done before.

The Take Home Labs concept was developed jointly in this thesis and in [11]. Five experiments were described in this thesis, and five complementary experiments are described in [11]. Together, these documents represent the current state of the Take Home Labs concept. The first experiment that was described in this thesis was an introductory experiment for setting up the main hardware and software. The second experiment was also an introductory experiment that reviewed the concept of sampling, and sampling and data acquisition with the Arduino. The third experiment was a Systems Dynamics dedicated experiment that explored the concept of frequency response by finding the open loop frequency response of a DC motor with an attached load. The fourth experiment was also a Systems Dynamics dedicated experiment that further explored the concept of frequency response by using the results from the open loop frequency response, and finding the closed loop frequency response of the DC motor with an attached load. The final experiment was designed for a graduate Optimal Control course. The experiment required that the student design an Optimal State Feedback Controller for a Ball and Beam system. These experiments were the beginning of many potential experiments.

9.1 Future Work

The addition of more experiments will be one future improvement to the Take Home Labs. The Take Home Labs can also be improved in the future by working with better 3-D printers as the prices continue to decrease, experimenting with other microcontrollers and hardware, and adding elements to enhance the simplicity and accessibility of the website. More experiments will help the Take Home Labs expand to other areas of engineering, and it will create a more diverse learning platform. The 3-D printer is a major factor in making the designs adaptable, thus continuing to improve current designs, and make new designs, as the functionality of 3-D printers improves, will allow more flexible and complex systems be designed easily. Adapting the experiments to work with the Raspberry Pi and the Beaglebone in the future will make the experiments more attractive to students who may have not have experience with the Arduino. The micrcontrollers are also affordable, and even more powerful than the current Arduino, therefore they may be better options for more advanced future experiments. To add to the simplicity and accessibility, adding videos to the website or making a YouTube series of videos is a potential option for helping students with common problems they may encounter. Some students are more confident when they hear someone explain things to them, and videos could help them feel more comfortable attempting the experiments. This would be a potential solution for not having a TA since it provides the student with some assistance.

BIBLIOGRAPHY

[1] M. T. Hagan and C. D. Latino, "A modular control systems laboratory," *Computer Applications in Engineering Education*, vol. 3, no. 2, pp. 89–96, 1995.

[2] M. T. Magan, C. A. Hernandez, W.-C. Yeung, and M. Hozhabri, "A control systems laboratory with microcomputer supervision," *SIMULATION*, vol. 1, p. 1, 1984.

[3] W. Durfee, P. Li, and D. Waletzko, "At-home system and controls laboratories," in *Proceedings of the American Society of Engineering Education Annual Conference & Exposition*, 2005.

[4] J. P. Oliver and F. Haim, "Lab at home: Hardware kits for a digital design lab," *Education, IEEE Transactions on*, vol. 52, no. 1, pp. 46–51, 2009.

[5] M. K. Jouaneh and W. J. Palm III, "Control system experiments at home," in *Frontiers in Education Conference (FIE), 2011*, pp. T2G–1, IEEE, 2011.

[6] J. Sarik and I. Kymissis, "Lab kits using the arduino prototyping platform," in *Frontiers in Education Conference (FIE), 2010 IEEE*, pp. T3C–1, IEEE, 2010.

[7] Y. S. Lee, Y. S. Park, B. E. Jo, and S. Seo, "Low-cost rcp system for control courses using matlab and the open-source hardware," in *World Congress on Mechanical, Chemical, and Material Engineering (MCM 2015)*, MCM, 2015.

[8] J. Crist, J. Littlefield, K. Kim, P. Kruse, B. Sohn, K. Strauss, and W. Durfee, "Me3281 take-home lab kit me4054w senior design," tech. rep., University of Minnesota, 2013.

[9] Quanser, "Quanser - real-time control experiments for education and research," 2015.

[10] MinSeg, "The miniature balancing robot: A low-cost mobile lab experiment kit for education," 2015.

[11] S. Hendrix, “Take home labs and the furuta pendulum,” Master’s thesis, Oklahoma State University, 2015.

[12] Ceat-its.okstate.edu, “Laptop specification recommendation for CEAT | CEAT Information Technology Services,” 2015.

[13] Mathworks.com, “Simulink - Simulation and Model-Based Design,” 2015.

[14] C. G. Bolivar-Vincenty and G. Beauchamp-Baez, “Modelling the Ball-and-Beam System From Newtonian Mechanics and from Lagrange Methods,” *LACCEI*, 2014.

[15] W. Wang, “Control of a ball and beam system,” Master’s thesis, University of Adelaide, Australia, 2007.

VITA

Carion Pelton

Candidate for the Degree of

Master of Science

Thesis: TAKE HOME LABS AND THE BALL AND BEAM

Major Field: Electrical Engineering

Biographical:

Personal Data: Born in Ardmore, Oklahoma, United States of America on May 9, 1990.

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 at Oklahoma State University in December, 2015.