MODELING AND CONTROL OF A MOTOR SYSTEM USING THE LEGO EV3 ROBOT

Ashley C. Mitchell, B.S.A.E

Thesis Prepared for the Degree of

MASTER OF SCIENCE

UNIVERSITY OF NORTH TEXAS

August 2015

Mitchell, Ashley C. *Modeling and Control of a Motor System Using the Lego EV3 Robot*. Master of Science (Electrical Engineering), August 2015, 52 pp., 32 figures, 8 numbered references.

In this thesis, I present my work on the modeling and control of a motor system using the Lego EV3 robot. The overall goal is to apply introductory systems and controls engineering techniques for estimation and design to a real-world system. First I detail the setup of materials used in this research: the hardware used was the Lego EV3 robot; the software used was the Student 2014 version of Simulink; a wireless network was used to communicate between them using a Netgear WNA1100 wifi dongle. Next I explain the approaches used to model the robot's motor system: from a description of the basic system components, to data collection through experimentation with a proportionally controlled feedback loop, to parameter estimation (through time-domain specification relationships, Matlab's curve-fitting toolbox, and a formal least-squares parameter estimation), to the discovery of the effects of frictional disturbance and saturation, and finally to the selection and verification of the final model through comparisons of simulated step responses of the estimated models to the actual time response of the motor system. Next I explore three different types of controllers for use within the motor system: a proportional controller, a lead compensator, and a PID controller. I catalogue the design and performance results – both in simulation and on the real system – of each controller. One controller is then selected to be used within two Controls Systems Engineering final course projects, both involving the robot traveling along a predetermined route. The controller's performance is analyzed to determine whether it improves upon the accumulation of error in the robot's position when the projects are executed without control.

ACKNOWLEDGEMENTS

I would like to express my gratitude to Dr. Yan Wan for her support during the pursuit of my graduate degree; thank you for your encouragement and guidance during my research for this thesis and for providing me with opportunities to further my professional development.

I would also like to thank Dr. Shengli Fu and Dr. Hyoung Soo Kim for participating in my defense committee and for sharing their professional expertise with me during my education at the University of North Texas.

Thank you Holden Mitchell for being my motivation.

Finally, and always, my sincerest gratitude goes to Chris Mitchell: for everything.

TABLE OF CONTENTS

LIST OF FIGURES

Page

CHAPTER 1

INTRODUCTION

1.1 Motivation

The design of controllers for real-world systems is a complicated task at best. Knowledge of internal dynamics, external disturbances, and the interplay of different parameters in the give-and-take of accomplishing performance goals are all necessary to effectively translate a system into a mathematical model for accurate simulation and controller design. Accurate modeling is crucial to a controls engineer; the alternative of attempting to design within the system itself can be both costly and time-consuming. The mathematical theories of system modeling and controller design taught in engineering courses make several assumptions and simplifications (for example, linearity) for the purpose of educational clarity, but how well do these theories and approaches hold up when applied to a real system? In my research for this paper, I explored the applications of modeling and control theory in a practical context. I have highlighted how the methods taught in systems modeling and control systems engineering courses can be applied to the modeling and control of a motor system; along the way, I evaluated the difficulties encountered when working within the realities of noise, disturbance, and nonlinearity, as well as how they can be dealt with. The goal of my efforts was to provide a helpful starting point to students in these courses looking to understand the realities of the modeling and control of a motor system, as well as to provide an introductory tutorial on working with the Lego EV3 robot.

1.2 Research Objectives

Previous controls system courses in the electrical engineering department at the University of North Texas have included a final semester project focused on the control of a Lego Mindstorms NXT robot. These projects were one of the following:

Project 1: The robot travels one meter, turns around, and returns to where it started.

Project 2: The robot travels in a square (one meter on each side) and returns to where it started.

Implementation of these projects was carried out using either Robot-C or Labview programming software, and the programs were either downloaded directly onto the robot's intelligence brick or the robot remained hardwired to the computer using a USB connection.

Recently, the controls lab received the newer generation of this robot: the Lego EV3. One of my objectives in this research was to learn how to use this new EV3 robot in conjunction with Simulink software and a wireless communication network, either wifi or Bluetooth. This work would help prepare future students in the lab to work with this new hardware in contexts that are more widely used in the engineering industry.

In order to explore the application of modeling and controller design theories onto a real-world motor system, my research objectives were as follows:

1. Find a model for the system using experimentation and parameter estimation,
2. Utilize that model to design an effective controller, and finally,
3. Test the effectiveness of my design by implementing the controller within the two course projects. An ideal controller, in both scenarios, will be one which allows the robot to end at exactly the position where it started.

1.3 Organization of the Thesis

The remainder of this thesis is organized as follows. Chapter 2 details the setup of materials used in this research project: the hardware and software, the wireless communication connecting them, and more details about the two Controls Systems course projects on which the final controller designs will be tested. It provides a tutorial so that the research may be reproduced for future work. Chapter 3 explains the approaches used to model the robot’s motor system, from a description of the basic system components, to data collection, to parameter estimation, and finally to verification of the final model by comparison of the simulated step response to the actual time response of the motor system. Three different modeling results are detailed: one is ideal, one incorporates friction, and one acknowledges a saturation limit on the power allowed into the motor system. The benefits and shortcomings of each of these models are discussed. Chapter 4 describes the design of a controller for the motor system. Three different types of controllers are explored: a proportional controller, a lead compensator, and a PID controller. It also catalogues the results of the performance of each of these controllers, and selects one for implementation within the course projects. The performance of the motor system with the addition of control is analyzed and compared to the performance without control. Finally, Chapter 5 contains a discussion of the results of the research; some suggested next steps to further this research; and a reflection on the contribution of this paper to the practical application of controller design theory.

CHAPTER 2

SETUP

2.1 Introduction

The setup for this research project included the following components. The hardware utilized was the Lego EV3 robot. The software used to program the robot was the 2014 Student Version of Simulink; this was chosen because Mathworks software is widely used in the engineering industry, and the student version could be used on my personal computer. The hardware and software communicated through wifi. In the next section I will describe in detail the steps taken to prepare for experimentation with the Lego motor system: the preparation of the hardware, software, and wifi; the steps taken to connect the three; and some of the issues encountered along the way. The final section of this chapter then provides more information on the final course projects, as well as some explanation of how they were initially programmed in Simulink.

2.2 Preparation of Components for Research

Although the Lego EV3 robot comes with a plethora of attachments (including different types of sensors and other mechanical components), only the basic build of the robot (using the instructions provided in the robot kit) was necessary to execute the travel routes of the two course projects: two large wheels were attached to the two servo motors connected to ports A and B for power inputs, and a third rolling ball was attached to the brick for stability. Because distance and heading changes were measured in terms of the rotation of the wheels, the only output of concern was the rotation of the motors; therefore, the only sensor needed was each motor's internal encoder, which measures rotation in terms of whole degrees.

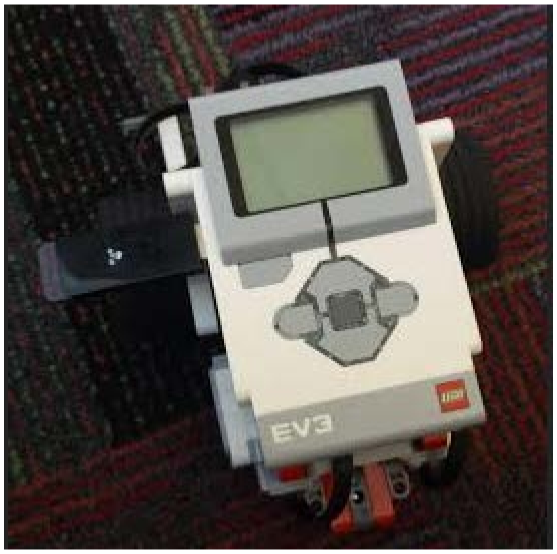


Figure 1: Assembled Lego EV3 Robot with Wifi Dongle

In order to access Simulink blocks specific to the Lego EV3 robot's input and output ports, it was necessary to download the Support Package for EV3 off of the Mathworks website and follow the instructions for connecting through wifi; this included updating the software on the brick. [1]

Although the Lego intelligence brick can be controlled with wifi, it does not have built-in wireless capabilities on its own nor is an attachment for connecting to wifi included in the robot kit. A separate USB wifi dongle had to be purchased and attached to the brick in order to connect it to a wireless network. After trying several different dongles and researching robotics forums, the Netgear WNA1100 was found to be the only one compatible with the Lego.

When initially attempting to connect to the wifi network in our controls lab, some limitations of the Lego intelligence brick became apparent. While the brick would allow a password to be entered for connecting to a network, it would not allow both a username and a password, which was necessary for logging in as a student to the University of North Texas (UNT) network. Also, the keyboard is limited to only numbers and letters, with no other characters offered. Another issue is that the network must either be WPA2 encrypted or have no encryption, so knowledge of the specific network the Lego is connecting to was critical; for example, to connect to my home wifi network I first had to change the settings on my network to match the encryption specifications of the Lego. It was also problematic to be dependent on the signal strength of available wifi. To circumnavigate these network issues, and to ensure that I could connect the Lego brick to the Simulink software on my computer regardless of my location, I connected both the hardware and software to a mobile hotspot network on my Samsung Galaxy S4 mobile phone. The hotspot allowed for some consistency in the availability of a useable wifi network for experimentation with the Lego -- I could always connect to it, as well as reset it or update the password if necessary. Even with this control, however, connectivity with the robot could be very inconsistent. Sometimes it would connect automatically, sometimes it would recognize the network but require the password to be re-entered, and sometimes it would give an error message for no clear reason and require the network name and information to be entered manually.

Once connected, the next step was to find the Lego's IP address on the intelligence brick under Settings, and manually input this address into Simulink under "Simulation -> Model

Configuration Parameters -> Run on Target Hardware" in order to run my Simulink models on the Lego robot.

For communication between the encoder and Simulink, it was also important to make sure the program was running in external mode. Without this step, the Scope block in Simulink would not display the output data from the Lego encoder. I also selected the option to "Save Data to Workspace" under the settings in the Scope block in my programs, so that I could collect quantitative time-response output data during my experimentations with the robot.

Some issues encountered while working with the Lego EV3 robot included: unclear error graphics, freezing during start-up or shut-down, an automatic shut-down based purely on length of time passed versus inactive time, delays in encoder outputs, and general lack of precision. If I were to continue with this research, I would be inclined to look into a more reliable, but equally cost-effective, option (for example, an Arduino-based robot).

2.3 More Details Concerning Course Projects

The only tangible output value that could be collected from the Lego motor's internal encoder was the rotation of the motor in degrees. To translate the travel distances and heading changes of the robot at different steps within the course projects (whether one meter straight, 180 degrees about-face turns in project 1, or 90 degree turns around the square in project 2) to their equivalent motor rotation, the Lego robot's physical dimensions had to be measured. The following measurements were taken: the diameter of the wheels was 2.25 inches and the distance between the outer edges of the two large wheels was 5.75 inches. Once found, these measurements could be used in conjunction with standard geometric relationships to convert distance and heading change requirements into their equivalent wheel rotations in degrees [2]. The resulting rotation requirements were then used as the set points, or desired rotations, within the corresponding steps of the course projects.

Because the course projects were executed on the Lego motor system in real-time, it was important to keep in mind certain inevitable communication delays between the software and hardware. For example, to avoid an accumulation of error, it was necessary to reset the encoder at each step in the projects; in other words, as each rotation - whether for forward motion or a change in heading - was met, the encoder needed to reset back to zero before beginning to track the next rotation. Without this reset, I risked an accumulation of error: if the first step had settled with a steady-state error, this error would undoubtedly lead to an error in the next step also, and so forth. However, if the encoder was programmed to reset at, for example, 6 seconds, the output of the encoder did not register a value of zero degrees until at least one sample time later, i.e. at least until 6.1 seconds. So it was important to include a delay within the Simulink program to ensure that the output had time to reset to zero in between steps.

Ideally, the course projects would be programmed to move between steps once the rotation for the current step has settled to its steady-state value (i.e. once the robot has traveled one meter, it then begins its heading change). For this project, however, the movement between steps was coded based on the clock: the time it took to complete a step in the project was observed, and the model was coded to move on to the next step after that completion time for the current step had passed. A more elegant coding based on settling times would be a valuable endeavor for future work on this topic.

CHAPTER 3

MODELING

3.1 Introduction

In order to design an effective controller to be used within the Control Systems final course projects, the first step was to find an accurate mathematical model for the Lego EV3 motor system; this would allow the controller to be designed using simulation instead of experimentation, thereby saving valuable time and costs. To model this system, the following steps were taken:

1. Understand the basic components of the system
2. Collect input and output data for the system through experimentation
3. Estimate the system parameters from the data
4. Enter the parameters into the system model to verify its response through simulation in both frequency and time domains

The goal of this process was to find a model that accurately mimicked the Lego motor system's performance, so that the model could be used to formally design an optimal controller for the system.

3.2 Basics of the Lego Motor System

To begin modeling, first the basic components and structure of the system needed to be understood. The input to, or actuation of, the system is power. The output is rotation in degrees.

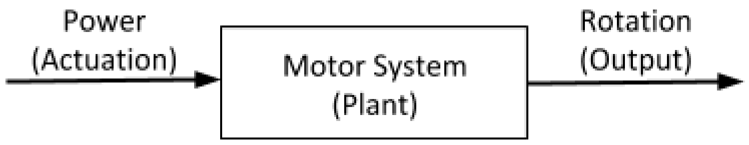


Figure 2: Basic Block Diagram of the Lego Motor System

The Lego EV3 robot has a wide range of capabilities, but the focus of this research is on its distance or motion control; therefore, a reasonable starting point for modeling was to assume that the Lego motor system could be modeled by a dynamic second-order system. The basic form for a second-order system's transfer function (or frequency response model) is shown below.

$$H(s) = \frac{\theta(s)}{P(s)} = \frac{k w_n^2}{s^2 + 2\xi\, w_n s + w_n^2}$$

. . . (1)

H(s) depicts the ratio of system outputs to inputs, θ(s) is the system output (rotation), and P(s) is the system input (power). The three parameters k (gain), w_n (natural frequency), and ξ (damping ratio) are the parameters that needed to be estimated through experimentation in order to find a precise model for the system [3].

When verifying the model for the system, I needed to observe the response of the system to a step input in both frequency and time domain to verify that it matched the output of the actual Lego system to the same input. The time-domain model was found by taking the inverse laplace transform of the frequency-domain model. The resulting second-order step response in time domain is shown in Equation 2.

$$\theta(t) = k\left(1 - e^{-\xi w_n t} * \left(\cos\left(w_n\sqrt{1-\xi^2} * t\right) + \frac{\xi}{\sqrt{1-\xi^2}}\sin\left(w_n\sqrt{1-\xi^2} * t\right)\right)\right) * u(t)$$

… (2)

Because the rotation data from the motor's encoder will be collected in time domain, this format of the system model was helpful during parameter estimation.

3.3 Data Collection

In order to gain insight into the internal dynamics of a system, experimentation must be performed to collect input and output data: the relationship between the two can thus be observed and then modeled through mathematics using parameter estimation techniques. Previous literature on the modeling of earlier Lego robot generations estimated the system parameters by analyzing the output of the motor system to a pulse input in an open loop, i.e. without feedback or reference tracking [4]. For my research, however, I used an alternative technique for experimentation: I constructed a simple feedback loop with a proportional controller to collect data on the closed-loop response of the system to a step input. I programmed the loop to track an arbitrary reference angle and assumed the controller had a unit gain; therefore the actuation into the system would simply be the error between the current encoder output and the desired set point. A screenshot of this Simulink model is shown in Figure 3.

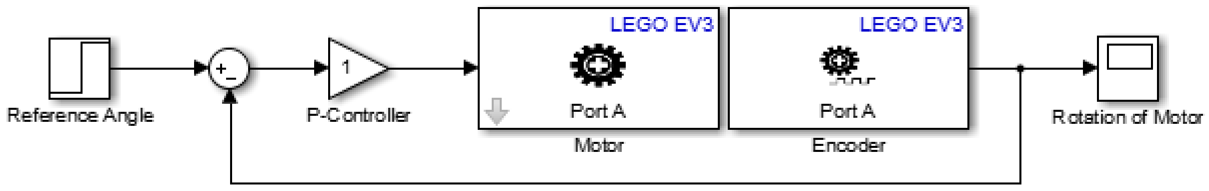


Figure 3: Simulink Model for Single-Wheel Data Collection using a Feedback Loop

It is important to note here that because of this feedback loop format, the time-response output data collected was indicative of the response of the closed-loop system. So when verifying the closed-loop step response through simulation, the plant model itself needed to be the open-loop model for the motor system; therefore the transfer function used for the model was different than the basic form mentioned above. If the gain through the sensor on the negative feedback branch is one, then the relationship between the open-loop and closed-loop models is:

$$H(s) = \frac{D(s)G(s)}{1 + D(s)G(s)}$$

. . . (3)

where D(s) is the transfer function for the controller and G(s) is the transfer function for the plant [3]. The closed loop second-order model is shown in equation 1; from this relationship, the open-loop model for the plant is then:

$$G(s) = \frac{kw_n^2}{s^2 + 2\xi\, w_n s + w_n^2(1-k)}$$

. . . (4)

3.4 Parameter Estimation

3.4.1 Introduction

The following section details how the parameters for the Lego motor system's model were estimated. After collecting data through experimentation, the transient and steady-state performance characteristics of the system's step response were observed. Using these observations, three different techniques for parameter estimation were explored: time-domain specification equations, Matlab's curve-fitting toolbox, and a formal least-squares estimation approach.

The first parameter estimation technique - using time-domain specification equations - is the application of relationships taught in class between system parameters (linked to the system's pole locations) and transient performance characteristics. These relationships are as follows [3]:

1. Rise time, defined as the time the system takes to increase from ten percent of its steady-state value to ninety percent of its steady state value, is related to the natural frequency of the system by the equation $t_{rise} = {}^{2.2}/_{w_n}$.

   . . . (5)

2. Peak time, defined as the time the system takes to reach its maximum output value, is related to damped natural frequency by the equation $t_{peak} = {}^{\pi}/_{w_d}$.

   . . . (6)

3. Percent overshoot is found by dividing the difference between maximum output value and steady-state value by the steady-state value. The relationship between

percent overshoot and damping ratio is defined using the following equation:

$$M_p\ \% = 100 * e^{-\xi\pi / \sqrt{1-\xi^2}}$$

. . . (7)

4. Settling time, defined as the time the system takes to settle to within one percent of its steady-state value, is related to the real component of the system's pole by the equation $t_{settling} = {}^{4.6}/_{\sigma}$.

. . . (8)

The system model's parameters can be estimated using an observation of the time-response characteristics of the system in conjunction with knowledge of the above relationships. The relationships described in equations 5 and 7 proved to be the most useful in finding my motor system parameters.

The second parameter estimation technique used was a formal least squares estimation approach [3]. This approach aims to minimize the difference between the output of the system (from data) and the expected output (from simulation). The initial function for this approach is shown below, where f(x) is the model for the output, y, and (x_i,y_i) is the real data collected from the system.

$$\min_{k,\xi,w_n} \sum_{i=1}^{n} (y_i - f(x_i))^2$$

. . . (9)

To complete the estimation, equation 9 is derived with respect to each of the parameters and set equal to zero (which is the mathematical process for finding the critical point, or in this case the minimum, of the function); then the resulting system of equations is solved for the parameter values.

The third parameter estimation technique utilized in my research was Matlab’s curve-fitting toolbox, called by entering “cftool” in the command window. To use this toolbox, the time-domain step response equation (see Equation 2) was entered as the custom equation; time and rotation output data collected from the Simulink scope were entered as the x- and y-variables respectively. The toolbox then automatically estimated the parameter values based on a regression analysis of the time-response data.

From the results of these parameter estimations, as well as due to the uncovering of more information about the realities of the system along the way, three different models were developed for the motor system: an ideal linear model, a linear model that incorporated the effects of frictional disturbance, and a nonlinear saturation model. The benefits and shortcomings of each are discussed in the following subsections before one is chosen as the best representation of the motor system.

3.4.2 Ideal Model

In my initial parameter estimation attempt, the Lego EV3 motor was treated as a pure black-box system: it was assumed that I had no knowledge of the internal dynamics of the motor (for example torque, power limits, angular velocity capabilities, etc.) and that the motor was a linear, time-invariant system (in order to directly apply introductory systems modeling techniques). The data for the estimation was collected without friction: the feedback loop was run on only one motor, with the robot lying on its back with its wheels in the air. Therefore, the response and rotation output were not affected by the surface the robot was traveling on. An arbitrary step input of 720 degrees yielded the time response output shown in Figure 4. Based on this plot, the rise time for the system was approximately 0.8 seconds indicating a natural frequency of about 2.8 (as calculated from Equation 5), the percent overshoot was 5.4% indicating a damping ratio of about 0.68 (as calculated from Equation 7), and there was no steady-state error which implied that the gain for the system was 1.

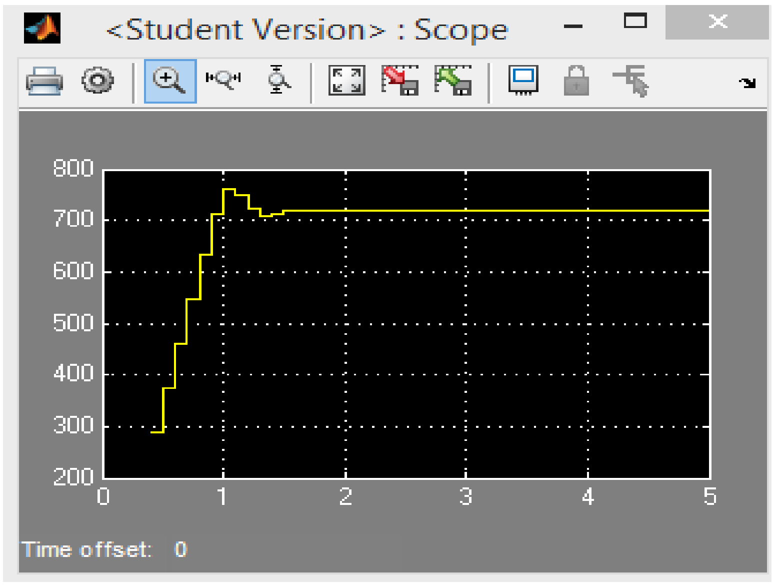


Figure 4: System Response of the Lego Motor with a 720 Degree Set Point

Theoretically, these parameters should have given me a fair model for the system. However, upon changing the set point to a different reference angle value, the performance characteristics - and therefore the system parameters - changed.

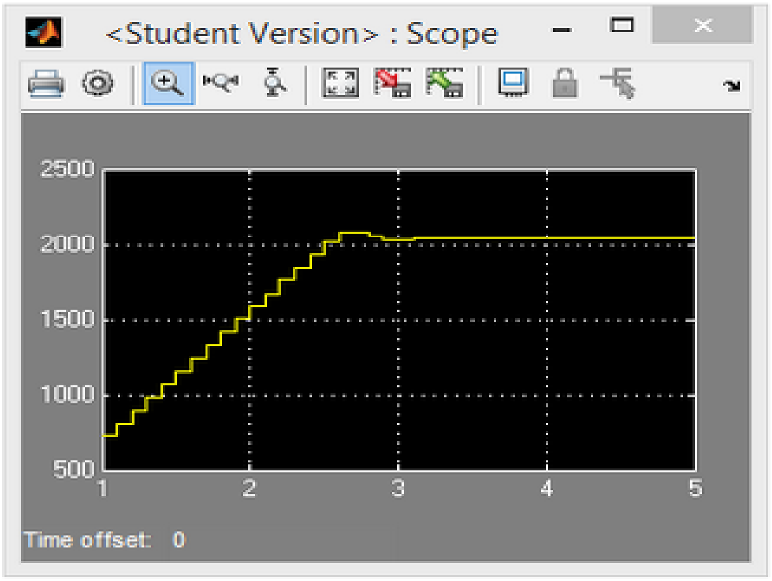
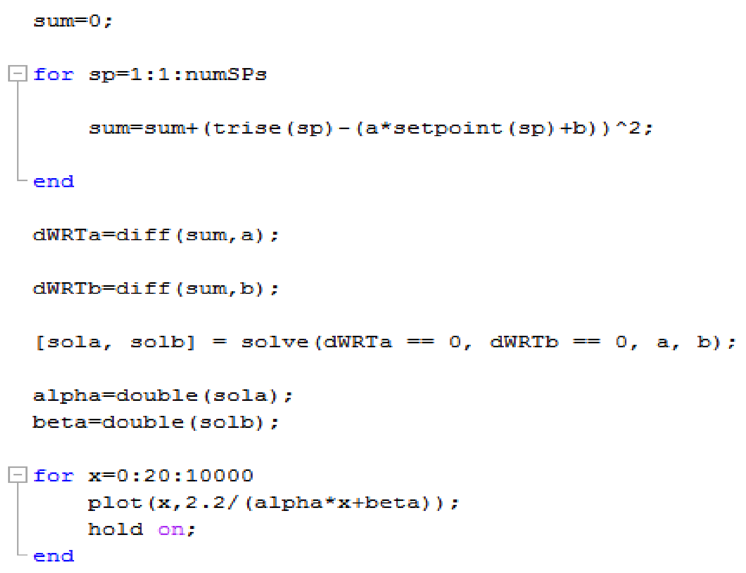


Figure 5: System Response of the Lego Motor with a 2046 Degree Set Point

Figure 5 shows the time response plot for another arbitrarily chosen set point. With this new reference angle of 2046 degrees, the rise time for the system was approximately 2.2 seconds indicating a natural frequency of about 1, and the percent overshoot was 1.8% indicating a damping ratio of about 0.79. These values differ greatly from those obtained with the initial 720 degree set point.

The differences between the parameters of these two set points required me to reflect on my assumptions about how to approach parameter estimation for this motor system. Initially, I had assumed that the parameters estimated would be consistent for the system regardless of the reference angle being tracked. However, my observations from experimentation with the system indicated that the performance characteristics were dependent on the set point. It did seem logical, after all, that in order for the motor to rotate to a larger angle it would take a larger amount of time. This suggested that the motor system's parameters themselves were dynamic, versus fixed like I had originally assumed, and therefore each parameter needed its own model to explain its dependence on the set point. To explore this trend further, a wide set of data was needed; this was collected by running the previous Simulink feedback loop model for a variety of set points. The parameters for each set point were calculated based on observations of the time-domain responses. Then, to find the relationships between set point and parameter values, my second parameter estimation technique – a formal least-squares parameter estimation – was used.

The first parameter I modeled using this approach was the natural frequency, or w_n. The data used for estimation was $(x_i, y_i) = (set\ point, t_{rise})$ and the model for the rise time was $f(x) = \alpha x + \beta$; this model was chosen because the rise time appeared to increase at a relatively constant slope with respect to an increase in the desired rotation. A portion of the Matlab code used to execute this estimation is shown in Figure 6.



```
sum=0;

for sp=1:1:numSPs

    sum=sum+(trise(sp)-(a*setpoint(sp)+b))^2;

end

dWRTa=diff(sum,a);

dWRTb=diff(sum,b);

[sola, solb] = solve(dWRTa == 0, dWRTb == 0, a, b);

alpha=double(sola);
beta=double(solb);

for x=0:20:10000
    plot(x,2.2/(alpha*x+beta));
    hold on;
end
```

Figure 6: Matlab Code Used to Implement the Least-Squares Parameter Estimation

The results from this program provided a model which allowed me to calculate the rise time for the system based on the set point. This rise time was then used to find the natural frequency, based on the time-domain specifications detailed previously. The relationship between set point and natural frequency is illustrated in Figure 7.

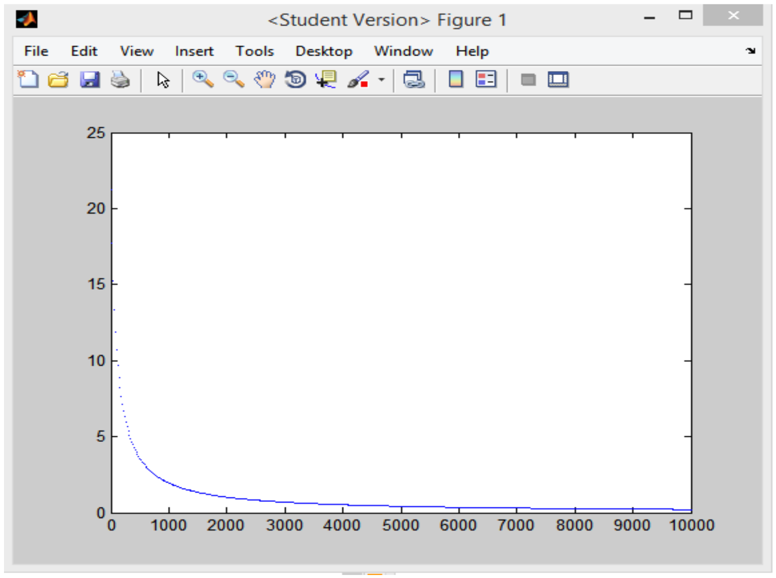
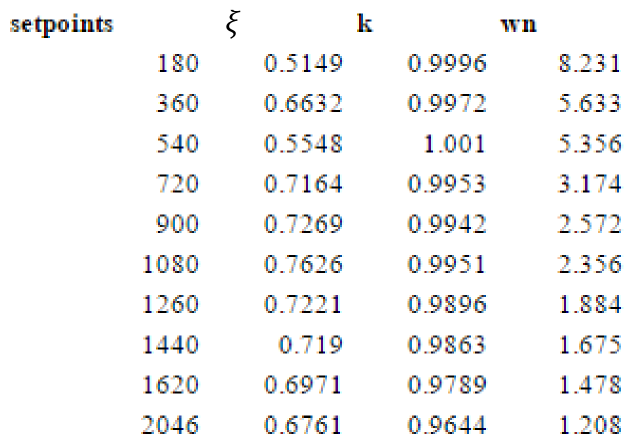


Figure 7: Set Point vs. Natural Frequency

The same approach was used to estimate a model for the damping ratio. An odd observation here, though, was that although the percent overshoot decreased with an increase in set point, the degrees of overshoot remained constant. This suggested that some physical characteristic of the Lego motor system itself was resulting in a fixed overshoot after the desired rotation was met. This discovery is revisited in Section 3.4.4, when more complexity and understanding is uncovered about the internal dynamics of the Lego system; for now the damping ratio trend was modeled as if only affected by the change in set point.

The formal least-squares parameter estimation was also repeated using data on the observed gains of the Lego motor system for the different set points. The results of the parameter estimations according to set point are shown in Figure 8. These parameters were then entered into the frequency-domain second-order system model to simulate the response of the motor system for a specific desired rotation output (see Section 3.5 for the results of the simulation).



setpoints	ξ	k	wn
180	0.5149	0.9996	8.231
360	0.6632	0.9972	5.633
540	0.5548	1.001	5.356
720	0.7164	0.9953	3.174
900	0.7269	0.9942	2.572
1080	0.7626	0.9951	2.356
1260	0.7221	0.9896	1.884
1440	0.719	0.9863	1.675
1620	0.6971	0.9789	1.478
2046	0.6761	0.9644	1.208

Figure 8: Parameter Values for a Selection of Set Points

From the table, it is clear that as set point increased, damping ratio increased (percent overshoot decreased) and natural frequency decreased (rise time increased). Both of these trends confirm what was observed through experimentation.

3.4.3 Friction Model

The issue with the ideal models found so far was that they were built from data collected without friction. In reality, the robot would be traveling on a surface, in contact with whatever friction that surface provided, and this would introduce some disturbance to the response of the system. Too much friction and the robot's wheels may have difficulty traversing the terrain; too little friction and slippage could occur. Either way the encoder's output may be misleading: the rotation of the motors may not accurately match the distance the robot actually traveled. Therefore, to improve the accuracy of the system model, the effects of friction needed to be considered. Rather than incorporate friction as a separate input into the motor system's feedback loop, which would require either an existing understanding of the friction coefficients of the travel surfaces or another set of experiments aimed solely at uncovering and quantifying this parameter, I decided instead to recollect the input and output data using the previous experiment, but altering it to deliver power to both of the motors, allowing the robot to travel on the ground. Thereby I could collect the time-response data of the motor system while it was subjected to the effects of friction. This would theoretically allow me to incorporate the effects of friction into the model itself, versus needing to define it separately as a disturbance to the system. For this to be reasonable, I needed to select one type of travel surface and then use the same surface for both the data collection and the testing of the controller design, since one would be based on the other. The travel surface selected for modeling was a low-pile carpet, which could be found at any of the locations where I would be doing my research: school, home, and the library. While each location surely did not have the exact same carpet and therefore would have slight differences in frictional coefficients, this allowed me to be relatively consistent. A screenshot of the adjustment made to the data collection Simulink model – to send power to both of the robot's wheels – is shown in Figure 9.

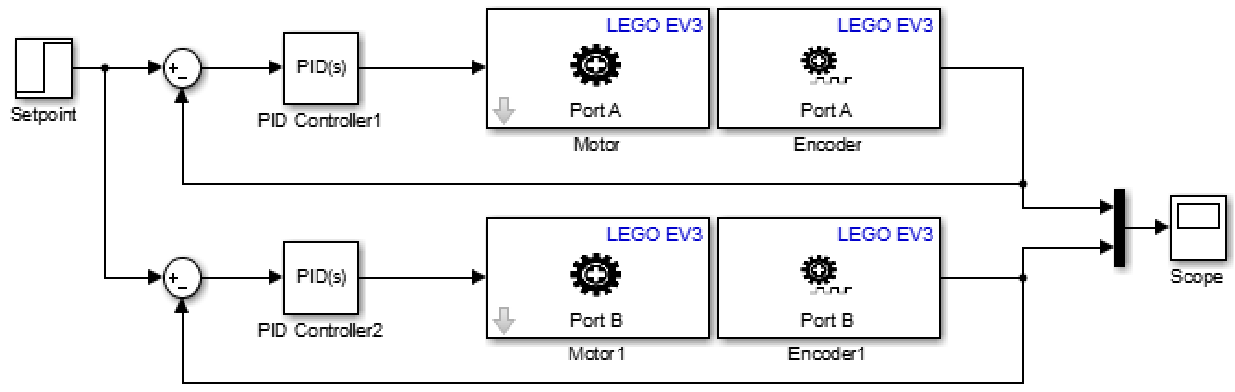
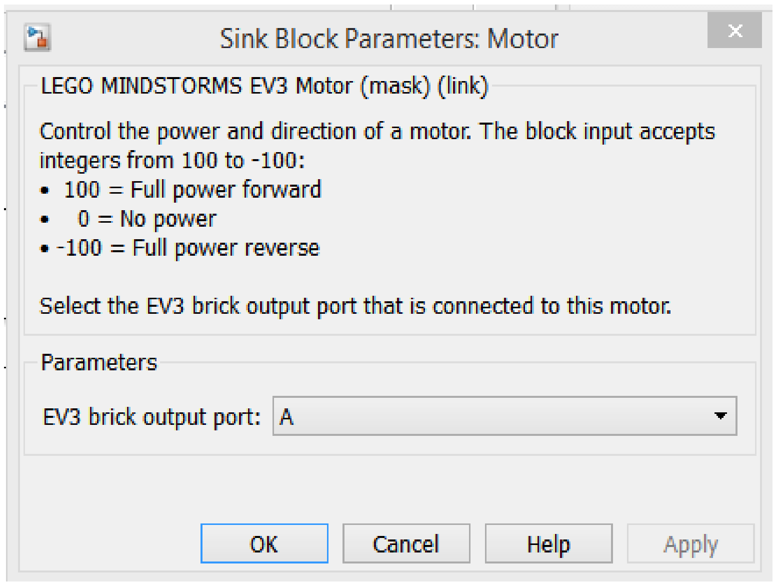


Figure 9: Simulink Data Collection Model for Friction Parameters

I used the same set points for this friction model as I had used before, which allowed me to compare the changes in parameters values for the ideal system directly to those found for the system when affected by friction. Because both wheels were being given power, double the data points were collected for each set point. The parameters were still dependent on the set point, as before, so the same least-squares estimation code was run again with the new data to estimate the model parameters again, this time acknowledging the effects of friction. The most notable change was that natural frequency parameters were much lower for the friction model: for example, the natural frequency for a set point of 360 degrees with friction dropped to 1.7179, which was considerably lower than the 5.633 estimated for the ideal model. A lower natural frequency translated to a longer rise time: this result seemed logical since the motor should have taken longer to rotate to a desired set point if it was working against friction.

3.4.4 Saturation Model

So far, I worked under the assumption that the internal dynamics of the Lego motor system were completely unknown: however, this was not entirely true. In actuality, the Simulink motor block for the Lego motor told me that there is a limitation set on the power input to the motor (see Figure 10).



LEGO MINDSTORMS EV3 Motor (mask) (link)

Control the power and direction of a motor. The block input accepts integers from 100 to -100:
- 100 = Full power forward
- 0 = No power
- -100 = Full power reverse

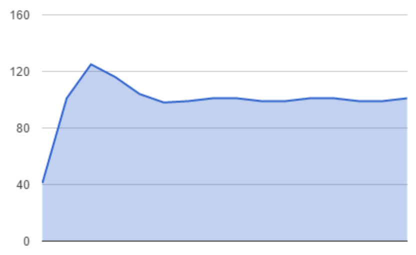
Select the EV3 brick output port that is connected to this motor.

Parameters

EV3 brick output port: A

Figure 10: Simulink Block for the Lego Motor

The description within the Simulink block for the Lego motor shows that the block only accepts integer values ranging from negative to positive 100, indicating full power in reverse to full power forward. Therefore, even though the error between set point and encoder output was modeled to feed into the Lego motor, the actuation of the system was actually being cut off at an upper limit of 100 degrees due to the physical specifications for the system. This saturation meant that the motor system was actually nonlinear; basically it had two parts: the first assumed a constant maximum power and the second was a step response based on the remaining 100 degrees. So essentially, all the data I had collected before only acted as a true second order system response for that last 100 degrees of error; in other words, the system was only responding linearly when it had a set point of 100 degrees, even though many other higher reference angles were tested. To confirm this discovery, I isolated my data to begin where the encoder output was 100 degrees below the set point: when comparing these isolated outputs for all of the different set points, the resulting curves looked nearly identical (see Figure 11).



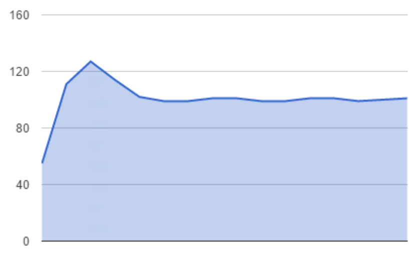
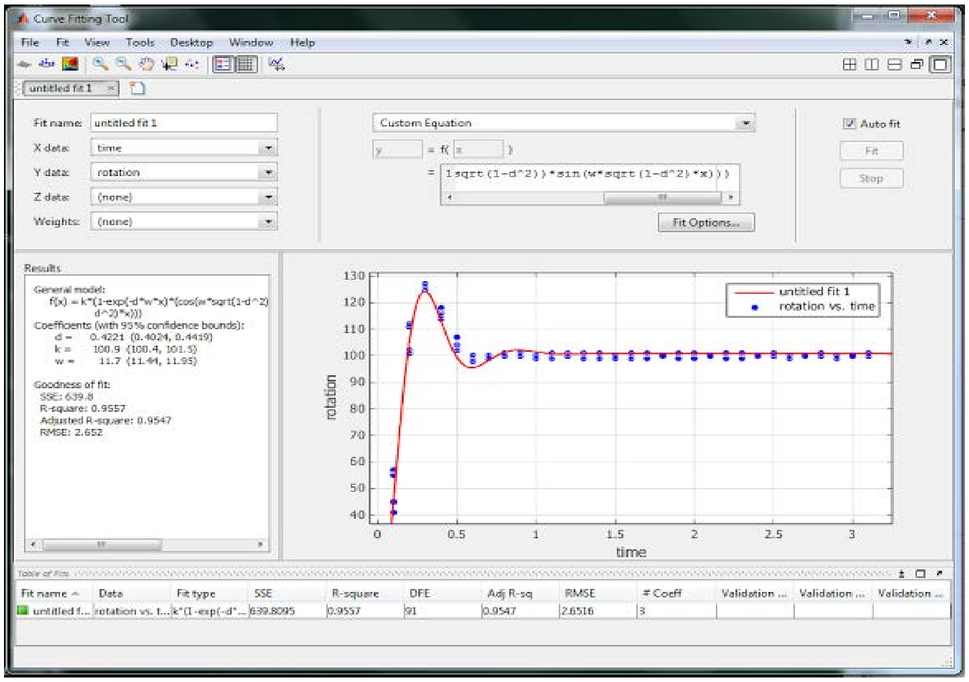


Figure 11: Isolation of the Top 100 Degrees of the 370 degree Friction Response Data (left) and the 720 Degree Friction Response Data (right)

This saturation limit explained why the overshoot values in the previous data collection experiments were always the same regardless of the reference angle: the set point in the range that behaved as a linear second-order system was always 100 degrees, so the overshoot value was always the same value. This new development also meant that the system parameters were no longer dependent on set point. Because 100 degrees is such a small rotation, it was difficult to collect a reasonable amount of data with the step input set at this value. Therefore, instead of recollecting data, the isolated friction data was adjusted so that the times and outputs began back at zero. This allowed me to use all the data that had already been collected to represent a set point of 100 degrees. With this set point, the system response exhibited a natural frequency of 11 and a damping ratio of 0.39, according to time-domain characteristics. Next, a formal parameter estimation was needed to fine-tune the system model. To estimate these new fixed parameters, I utilized my final parameter estimation technique: the curve-fitting toolbox in Matlab. My adjusted time-response data was entered in for x and y, and the time-domain second order system step response equation was entered as the custom equation for regression analysis. The parameters found here closely matched the ones found by observation of the time-domain characteristics (see Figure 12).



Fit name: untitled fit 1
X data: time
Y data: rotation
Z data: (none)
Weights: (none)

y = f(x)
= 1 sqrt(1-d^2))*sin(w*sqrt(1-d^2)*x)))

```
Results
General model:
   f(x) = k*(1-exp(-d*w*x)*(cos(w*sqrt(1-d^2)
                 d^2)*x)))
Coefficients (with 95% confidence bounds):
    d =      0.4221  (0.4024, 0.4419)
    k =       100.9  (100.4, 101.5)
    w =        11.7  (11.44, 11.95)

Goodness of fit:
  SSE: 639.8
  R-square: 0.9557
  Adjusted R-square: 0.9547
  RMSE: 2.652
```

Table of Fits

Fit name	Data	Fit type	SSE	R-square	DFE	Adj R-sq	RMSE	# Coeff	Validation ...	Validation ...	Validation ...
🟩 untitled f...	rotation vs. t...	k*(1-exp(-d*...	639.8095	0.9557	91	0.9547	2.6516	3			

Figure 12: Curve-Fitting of the 100 Degree Set point Time-Response Data

For further confirmation that these parameters were estimated well, the values were inserted into the transfer function model for the motor system with a step input of 100 and the feedback response was verified in Simulink. The result is shown in Figure 13.

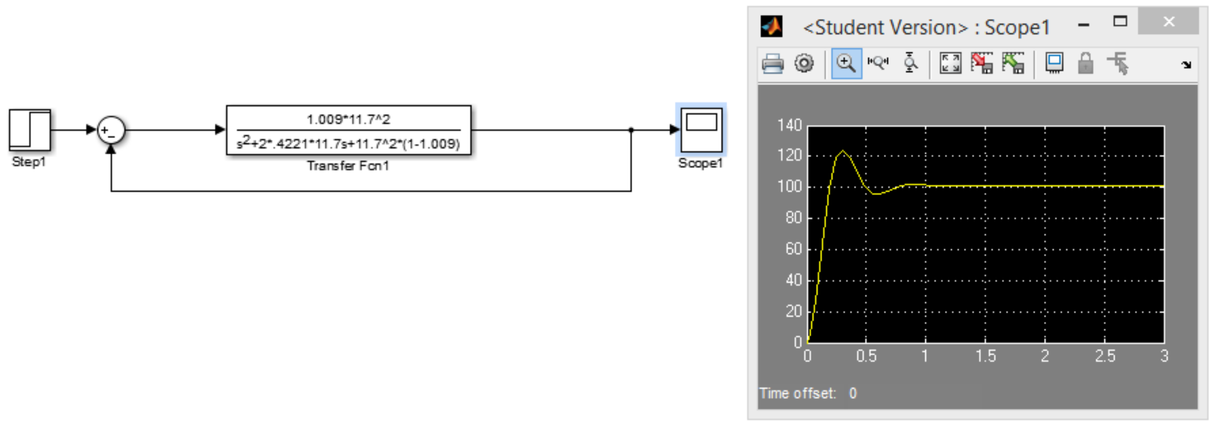


Figure 13: Simulated Response of the Lego Motor System within the Linear Range of the Saturation Model

While the time domain model of the closed loop system matches the actual encoder output well with no steady-state error, this simulated response of the frequency-domain model does show a slight error. This difference in steady-state values between the two responses could be due to the assumption (taken during calculation of the plant's open-loop transfer function) that the negative feedback through the sensor has a unit gain, which is likely not true for the real system. This could lead to some discrepancies between the Lego output and simulated response in controller design as well.

3.5 System Model Verification and Selection

To select the best system model to carry forward into controller design, the simulated responses of each of the three models estimated in this chapter were compared the actual encoder output of the Lego robot’s two motors running on low pile carpet with an arbitrarily selected reference angle of 360 degrees. The response of the real Lego motor system is shown in Figure 14; the simulated response of the ideal model, friction model, and saturation are shown in Figures 15, 16, and 17, respectively.

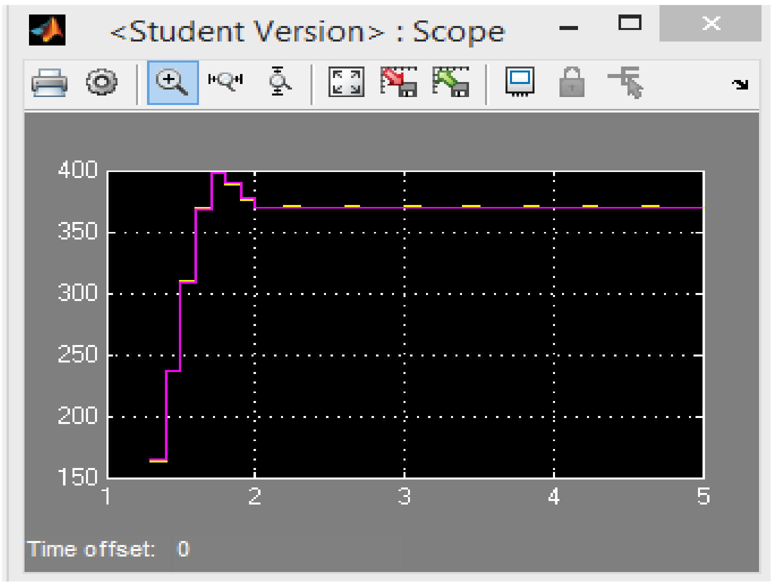
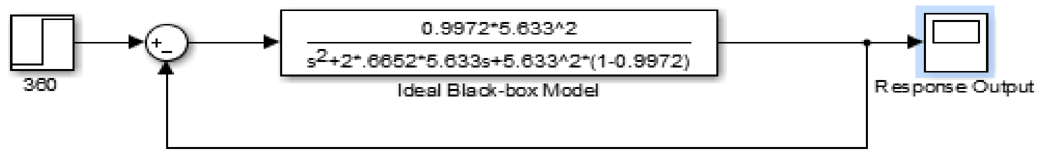


Figure 14: Lego Motor System Response on Low-Pile Carpet



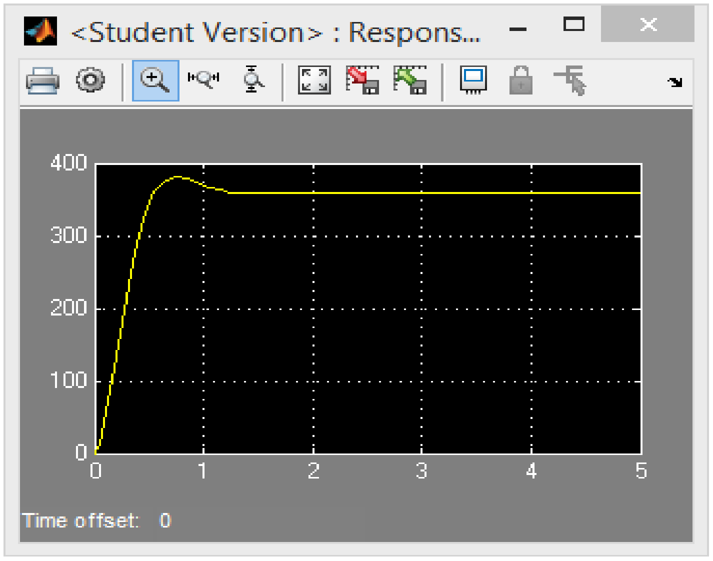
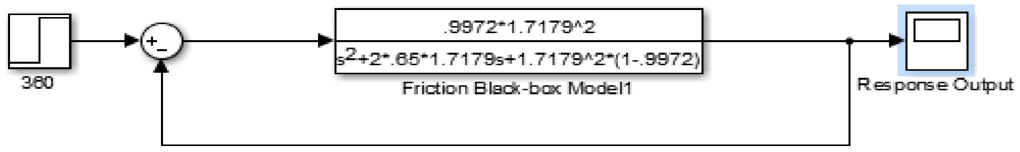


Figure 15: Simulink Feedback Loop and Resulting Step Response of the Ideal Model


360
.9972*1.7179^2
s2+2*.65*1.7179s+1.7179^2*(1-.9972)
Friction Black-box Model1
Response Output

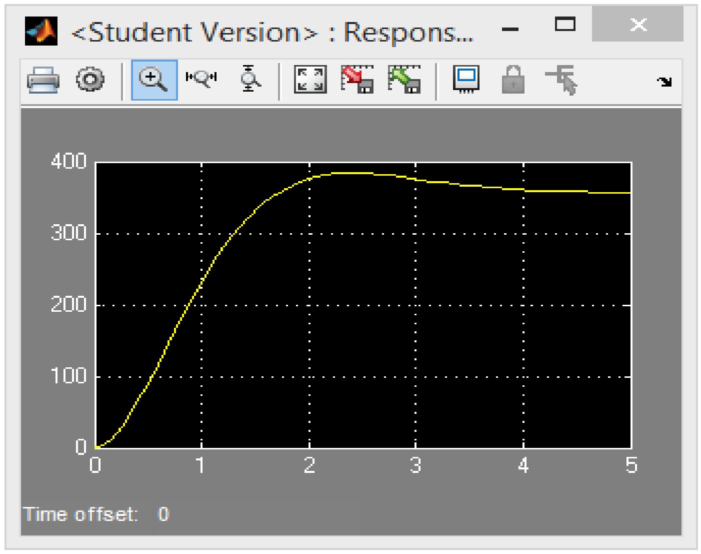


Figure 16: Simulink Feedback Loop and Resulting Step Response of the Friction Model

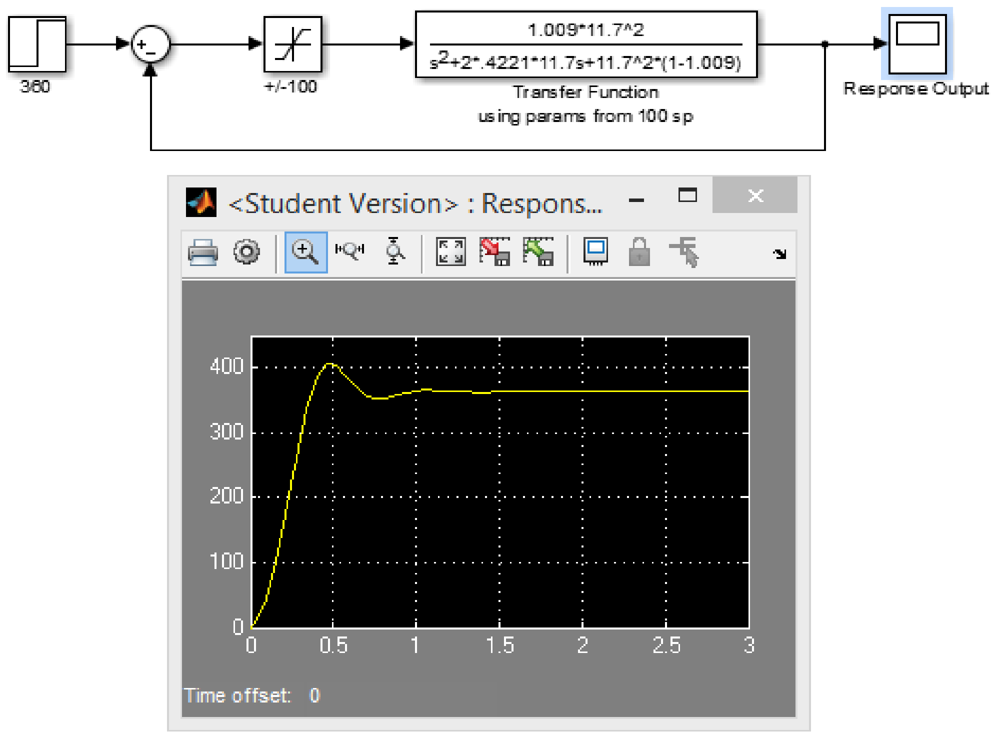


Figure 17: Simulink Feedback Loop and Resulting Step Response of the Saturated Model

Based on a comparison of the step response plots, the saturation model appeared to be the best representation of the internal dynamics of the Lego motor system. Therefore, this model was the only one utilized in the next chapter for controller design. However, exploring the other models (which were ignorant of this saturation limit) was still a valuable exercise in terms of understanding how to model a system through experimentation; because even without considering any details about the motor itself, I was still able to find a reasonable model for the system. Thus the estimation for the first two models was a useful illustrations of how the techniques of system modeling and parameter estimation could be carried out even with limited knowledge about the physical specifications of a system.

CHAPTER 4

CONTROL

4.1 Introduction

Feedback controllers are important to motor systems because they help to reject the effects of noise and disturbance, enabling the system to perform more accurately. In this chapter I describe my efforts in designing an optimal controller for use with the Lego EV3 motor system. I begin by explaining the adjustments I made to account for the nonlinearity of the system, follow with details on the performance goals for the motor, and then explore three different controller options: a proportional controller, a lead compensator, and a proportional-integral-derivative controller. Concepts, design details, and performance analysis are detailed for each controller before one is selected and tested with the two course projects.

4.2 Pivot Test Program

Because the most accurate model found for the Lego motor system was nonlinear, controller design for the system was more complicated than the relatively straightforward methods taught in a controls systems course. These introductory design methods assume that the system being controlled is linear, which was not true for the Lego motor because of the saturation limit. Therefore to begin designing a controller using these methods, I had to focus on adding a controller to just the linear portion of the system - i.e. to step responses from set points at or below 100 degrees. However, a 100 degree angle of rotation is very small and difficult to observe as a travel distance. Testing of the initial controller designs was therefore implemented on a very simple pivot model, where one wheel of the Lego robot remained still while the other rotated to 100 degrees. This movement was easier to observe for comparison of the system performance under different controllers than a program sending power to both of the wheels would have been.

4.3 Details on Performance Requirements

4.3.1 Basic Design Goals for the Controller

The design goals for control of the Lego motor system were as follows: the controller needed to minimize the system overshoot, minimize the steady-state error, and minimize the settling time. Within the course projects, the controller needed to enable the Lego robot to complete its trajectories as quickly and accurately as possible.

4.3.2 Time Performance Limitation

Knowing that there was an upper limit on the power allowed into the motor, the best peak response time for any reference angle could not be any better than the time it would take for the motor to rotate to that angle at maximum power. From the data collected previously, the slope of the system response during saturation indicated that at maximum power the Lego motor rotated approximately 72 degrees every 0.1 seconds. Therefore the fastest the motor could rotate to 100 degrees would be 0.1389 seconds. So in moving forward with controller design, this limitation needed to be kept in mind; if the simulated response of the controlled system showed a peak time less than 0.1389 seconds, then it was not actually a feasible response within the real physical system. Rather, the best possible peak time would be one that was as close as possible, without going under, this time limitation set by the specifications of the Lego motor.

4.3.3 Error Accumulation

Design goals for a controller can be subjective to the intent for the system: my chief concern in applying control to the Lego robot was to have it end exactly where it began when it ran the trajectories in the two course projects. Because each of the course projects consists of multiple steps, the biggest obstacle to accomplishing this goal was the system's steady-state error: if the steady-state error was too high, the error in position of the Lego robot along its course of travel would accumulate over the multiple steps of the projects. Even a low percentage of error in one rotational steady-state value could lead to a very large error between the starting and ending positions of the robot. Because of this threat of error accumulation, the design requirement that I assigned the most weight moving forward in controller design was minimizing the steady-state error.

4.3.4 Synchronization of the Wheels

Underlying the goal of precise travel along the robot's trajectories was the need for the robot to travel in a straight line. This required synchronization between the two wheels. Synchronization required an additional controller between the two motors, to ensure that any slight variations between the performance of the wheels - caused by internal friction and slight mechanical differences - would be accounted for and corrected. To optimize this controller, another set of experimentation and design would need to be conducted based solely on the output of error between the rotations of the motors over time. With a desired set point of zero error between the wheels, the techniques for modeling and design that I had utilized thus far in working with the individual motors are difficult to complete, from a mathematical standpoint. In order to move forward with controller design for the individual motors, I selected a reasonably well performing proportional controller with a gain of 0.1 to synchronize the performance of the two wheels. An excellent goal for future work would be to spend time formally designing a controller just for the synchronization.

4.4 The Need for Control

Before beginning to explore controller designs for the Lego motor system, the final course projects were executed without any controllers (either on the individual motors or between the wheels for synchronization) to highlight the need for control. The scope output from a trial run of the first course project is shown in Figure 18.

Without the presence of control, the performance of the Lego motor in course project 1 showed high overshoot and high steady-state error. The results on the error between its starting and ending positions were very inconsistent, ranging in values from 1.25 to 4.875 inches of error over several different trials. Additionally, there was a wide variability in the accuracy of the heading changes: even if the position error was fairly low, the Lego often ended facing in an odd direction. The same issues were true for the performance in course project 2: the steady-state errors for the heading change steps were especially apparent, with the robot failing to turn its full 90 degrees of rotation and thereby causing the four-legged journey to look decidedly un-square. The errors between starting and ending positions in course project 2 ranged from 3.625 to 18.875 inches over several trials.

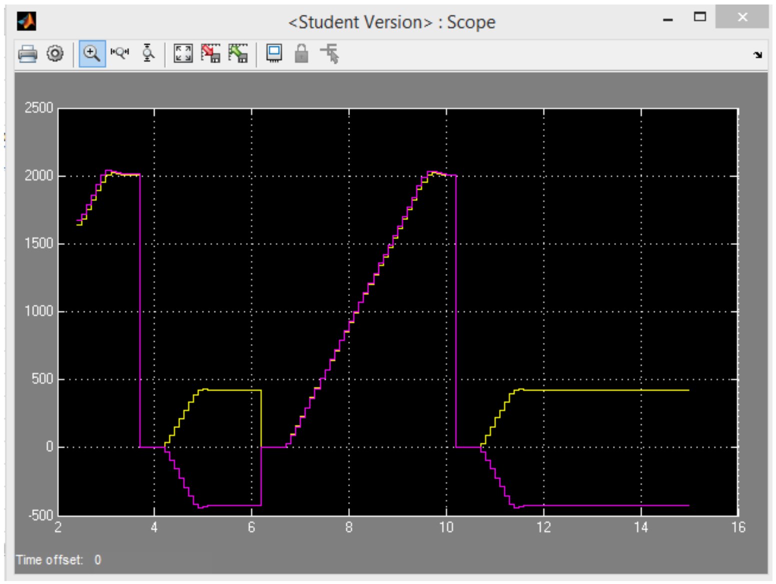


Figure 18: Course Project 1 Time-Domain Rotation Output Without Control

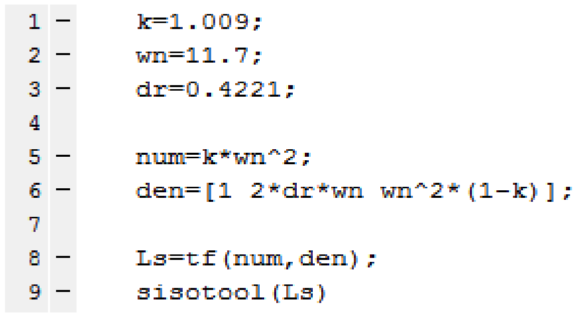
The issues in precision and accuracy of the course project executions illustrate the need for control of the Lego motor systems within the final Controls System course projects.

4.5 Proportional Controller

The first type of controller explored for use with the Lego motor system was a proportional controller. A proportional controller adjusts the actuation into a system in proportion with the current error between the system output and the desired set point. While a P-controller is theoretically simple to design, since there is only one parameter to find, performance under proportional control can be unsatisfactory due to tradeoffs between various aspects of the time-domain response characteristics: for example, a higher gain may reduce the steady-state error for the system, but the cost could be a higher overshoot.

4.5.1 Design of an Optimal P-Controller

To find an optimal value k_p for the proportional gain, the open-loop transfer function of the system is analyzed using a root locus plot; this plot illustrates the effects that changes in system gain have on the performance characteristics of the system, as represented by the system's pole locations. Figure 19 shows my Matlab code for simulating a root locus plot for the Lego motor system.



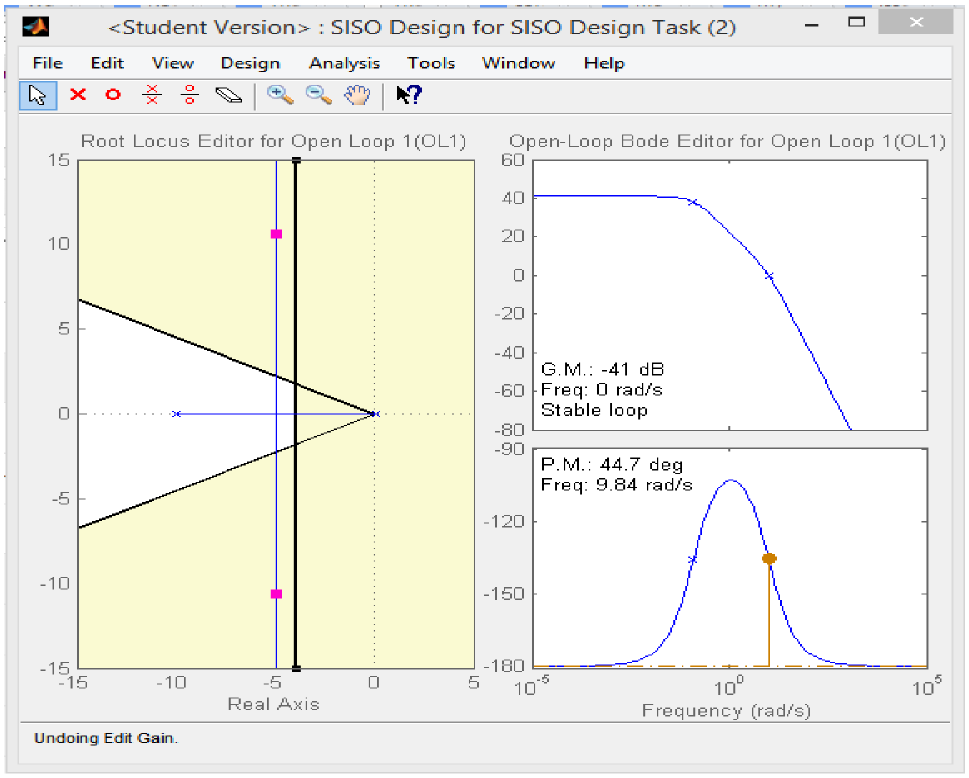
```
k=1.009;
wn=11.7;
dr=0.4221;

num=k*wn^2;
den=[1 2*dr*wn wn^2*(1-k)];

Ls=tf(num,den);
sisotool(Ls)
```

Figure 19: Matlab Code for Root Locus

I chose to use the "sisotool" command, because it allowed me to input my performance requirements for the system directly onto the plot. By right-clicking on the root locus, I added the following design requirements: an overshoot of less than one percent and a settling time of less than one second. The resulting design regions are shown on the root locus plot in Figure 20. The system's current pole locations lie outside of the desired performance region (the unshaded area), indicating that the poles need to be moved -- i.e. the system's gain needs to be changed -- in order to meet the desired design requirements. From a conceptual understanding of the inverse relationship between gain and steady-state error, the highest gain possible within this region would give the lowest possible steady-state error that could be achieved while still satisfying the overshoot and settling time requirements. By manipulating the pole locations of this plot, I found the optimal k_p value for meeting these requirements was 0.213; this was the gain I selected for my proportional controller.


<Student Version> : SISO Design for SISO Design Task (2)
File Edit View Design Analysis Tools Window Help
Root Locus Editor for Open Loop 1(OL1)
Open-Loop Bode Editor for Open Loop 1(OL1)
15
10
5
0
-5
-10
-15
-15 -10 -5 0 5
Real Axis
60 40 20 0 -20 -40 -60 -80
-90 -120 -150 -180
10^{-5} 10^{0} 10^{5}
Frequency (rad/s)
Undoing Edit Gain.

G.M.: -41 dB
Freq: 0 rad/s
Stable loop

P.M.: 44.7 deg
Freq: 9.84 rad/s

Figure 20: Root Locus Plot with Design Requirements

4.5.2 Analysis of the P-Controller's Performance

The simulated response of the motor system to a 100 degree step input with the addition of this proportional controller design is shown in Figure 21.

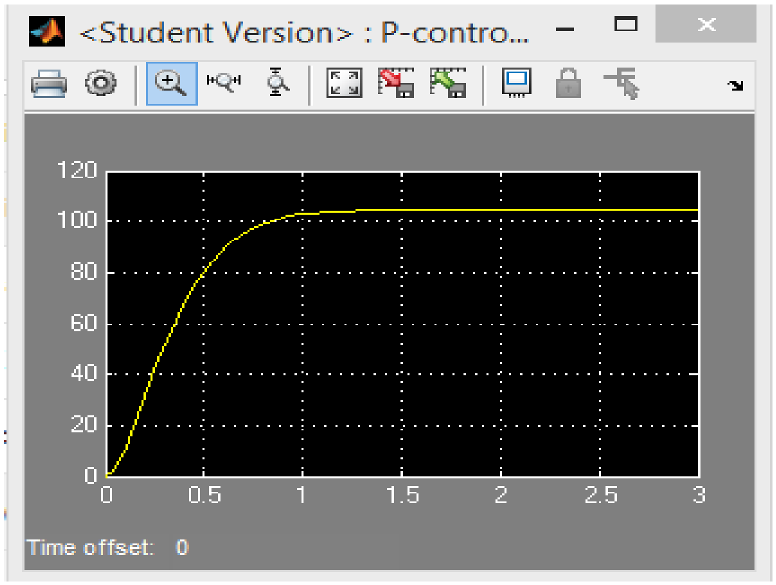


Figure 21: Simulated Response of the Lego Motor System with Proportional Control

When the same proportional controller was added to the pivot program for the Lego robot, the scope yielded the following response shown in Figure 22.

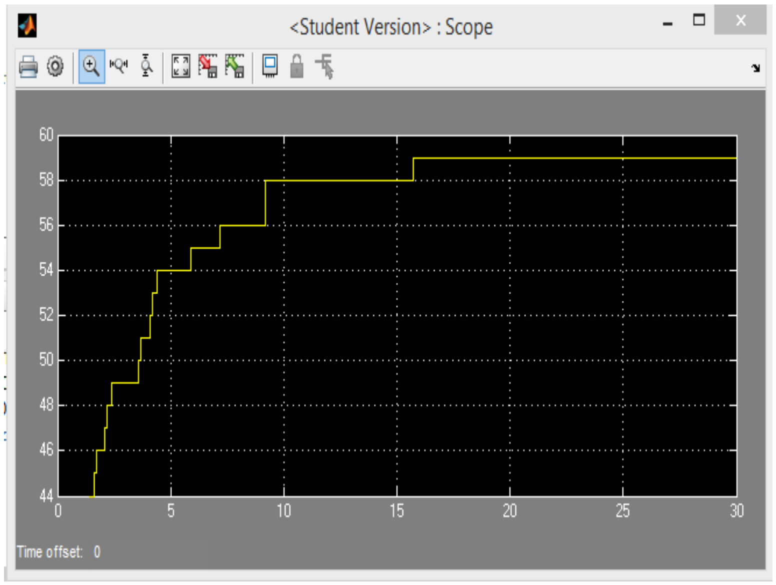


Figure 22: Actual Time-Response of the Lego Motor System with Proportional Control

While the response of the Lego motor under this controller was smooth and resulted in no overshoot, it exhibited a higher steady-state error than anticipated; this served as further proof of the tradeoffs between different performance characteristics under a proportional-controller, as well as an illustration of the differences between the ideal design and practical implementation of control.

4.6 Lead Compensator

The next type of controller design I attempted was a lead compensator, which acts as a type of modified proportional-derivative controller; actuation into the system is adjusted based on both the current error and the prediction of future error. The parameters for a lead compensator are designed using the equation for a system's steady-state error and the frequency response plots for the system.

4.6.1 Design of a Lead Compensator

My first step in lead compensator design was to choose a gain k for the compensator based on the desired steady-state error for the system, which is found using the following equation:

$$E_{ss} = \frac{1}{1 + kG(0)}$$

. . . (10)

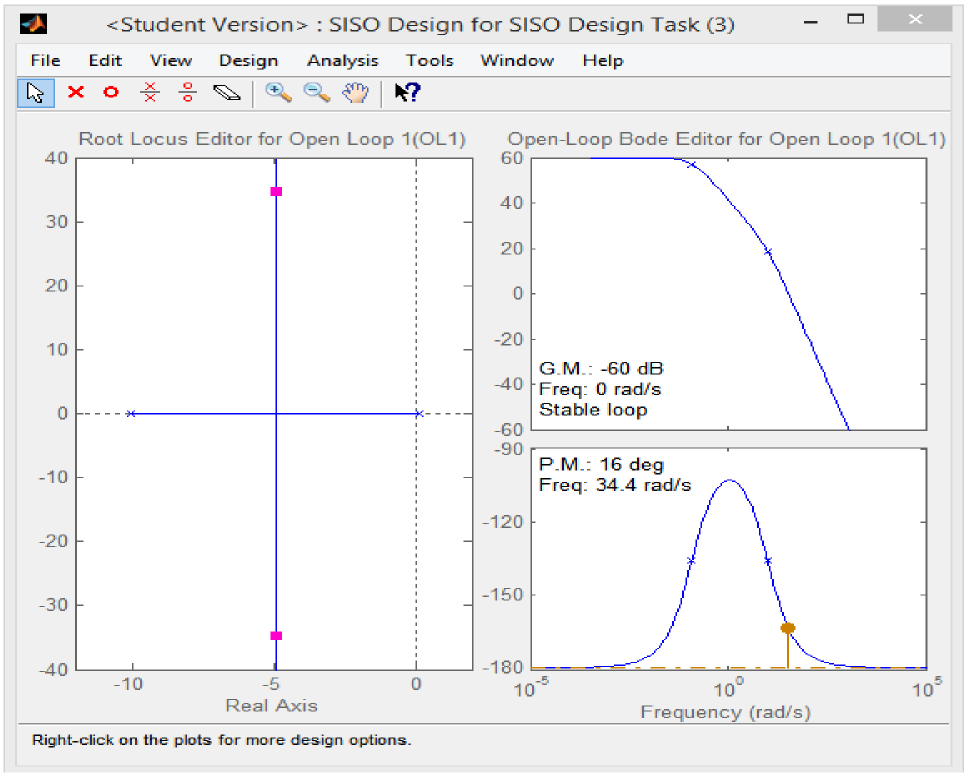
where G(0) is the transfer function of the plant evaluated at s equal to zero. I selected an aggressive desired error of less than 0.1 percent, due to my decision on the importance of a very low steady-state error. The result of this relationship was the design requirement that k needed to be greater than 8.9; I selected k=8.92, not wanting the gain to go too high and thereby risk instability.

The next step in lead compensator design was to decide on an allowable overshoot for the system, which translated into the phase margin requirement using the following equation [7].

$$\varphi_m = \tan^{-1}\frac{2\xi}{\sqrt{-2\xi^2+\sqrt{1+4\xi^2}}} \quad \ldots (11)$$

In an optimistic attempt to achieve both a desirable error and a low overshoot, I decided to design for zero overshoot: this meant a damping ratio of 1 and phase margin requirement of 76.3466. I now needed to look at the frequency response plot, or Bode plot, for the system, subject to the gain calculated in the previous step. Again, I chose to use Matlab's sisotool command because of its graphical user interface, which illustrates the interplay between the different system parameters; the Matlab code used was the same as in the p-controller design section except for the addition of the 8.92 gain.

Based on the Bode plot of the current system, shown in Figure 23, my phase margin without compensation was only 16 degrees. As illustrated in the plot, increasing my phase margin to meet my overshoot requirements would also impact the gain margin: to account for this tradeoff, I added a 10 degree margin to my phase margin requirement, resulting in a need for 70.3466 more degrees of phase in order to get the system overshoot I desired.



<Student Version> : SISO Design for SISO Design Task (3)
File
Edit
View
Design
Analysis
Tools
Window
Help
Root Locus Editor for Open Loop 1(OL1)
Open-Loop Bode Editor for Open Loop 1(OL1)
Real Axis
Frequency (rad/s)

G.M.: -60 dB
Freq: 0 rad/s
Stable loop

P.M.: 16 deg
Freq: 34.4 rad/s

Right-click on the plots for more design options.

Figure 23: Frequency Response Plot for Lead Compensator Design

This phase value was used to calculate the system parameters for the lead compensator, whose transfer function is

$$D(s) = K\frac{Ts+1}{\alpha Ts+1}$$

. . . (12)

Alpha is calculated by

$$\alpha = \frac{1-\sin\varphi}{1+\sin\varphi}$$

. . . (13)

where the phase is measured in radians. Alpha must be less than one in order to have lead compensation [3].

T is calculated by

$$T = {}^{1}/_{w_c\sqrt{\alpha}}$$

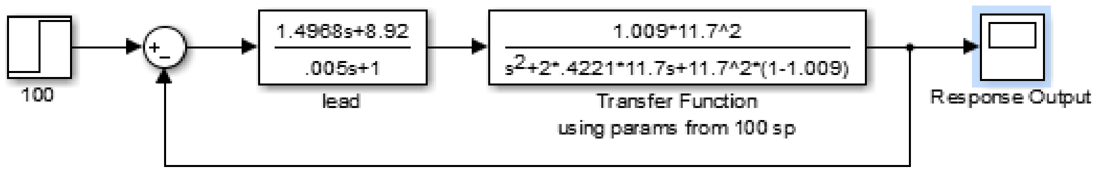
. . . (14)

where w_c is the crossover frequency found from the initial Bode plot (34.4 rad/sec).

When the resulting lead compensator transfer function was applied to the motor system's model, the frequency response plot showed that I had not yet reached my desired phase margin. This design process can be repeated recursively until a design matches well enough to the desired controller performance; instead I chose to accept the higher overshoot as a tradeoff for the lower steady-state error and observe how the performance of the lead compensator compared to that of the proportional controller when applied to the actual Lego motor system.

4.6.2 Analysis of Lead Compensator Response

The simulated response of the system with lead compensation is shown in Figure 24. The performance of the Lego motor when my lead compensator design was added the pivot program is shown in Figure 25. As predicted, the system had some overshoot, but settled at the desired rotation of 100 degrees.


100
$\frac{1.4968s+8.92}{.005s+1}$
lead
$\frac{1.009*11.7^2}{s^2+2*.4221*11.7s+11.7^2*(1-1.009)}$
Transfer Function
using params from 100 sp
Response Output

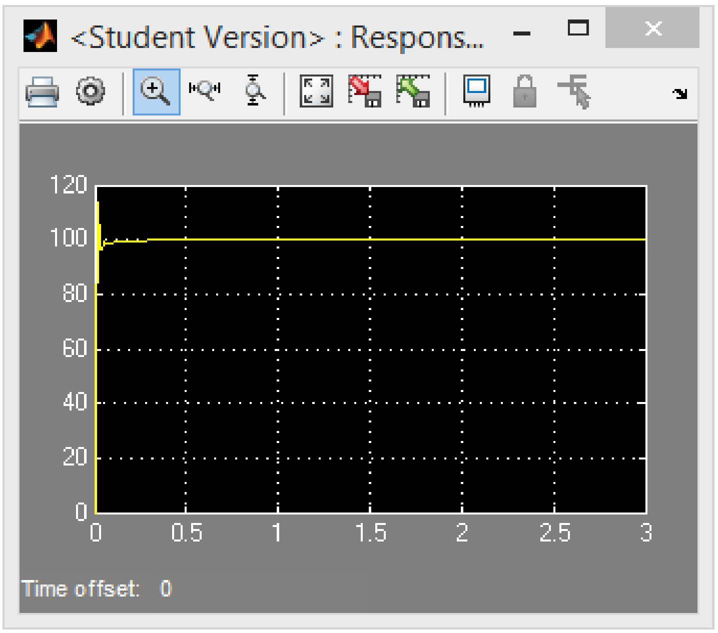


Figure 24: Simulated Response of the Lego Motor System with Lead Compensation

The discrepancies between the simulated response and the real-time motor system response could be attributed to the fact that the settling time for the simulated system under the lead compensation was very fast: because of the time limitations discussed previously, the simulated response was not technically feasible on the real system. This was a side effect of the design requirements of steady-state error and overshoot that I used when determining the parameters for my compensator. Again, this discrepancy illustrates the disconnect that can occur between conceptual design and practical implementation, as well as the tradeoffs and realities of physical limitations that a controls engineer must keep in mind during design.

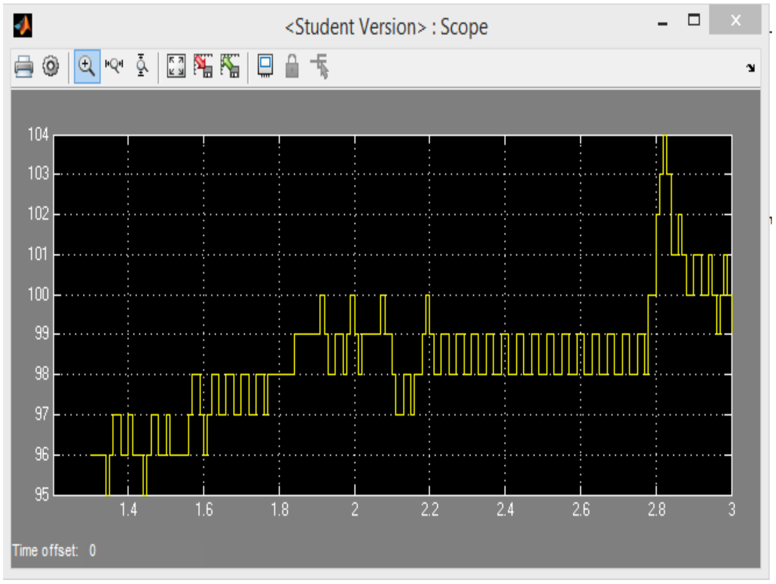


Figure 25: Actual Time-Response of the Lego Motor System with Lead Compensation

4.7 Proportional-Integral-Derivative Controller

The basic transfer function for a proportional-integral-derivative controller is

$$D(s) = k_p + {}^{k_i}/_{s} + k_d s.$$

. . . (15)

Design for this controller can be difficult, because there are three different parameters to design, each affecting the actuation into the plant based on a different component of the error: k_p is based on the current error, k_i on the accumulation of past error, and k_d on the prediction of future error, based on the rate of change of error [3]. The effects of the values chosen for each of the three parameters are not independent; as such, the tuning of these parameters is a recursive process.

4.7.1 Design of a PID Controller

Because a key goal of this research was the exploration of controller design concepts from class in a practical context, I decided to use a manual tuning approach for the design of this PID controller: this approach would allow me to observe the effects of changes to the system when adjusting the value of each parameter individually. The steps I took to approach this manual tuning were outlined in [5].

The response of the system model to a unit proportional controller with a 100 degree set point was already simulated in Figure 13. My first step in manual tuning was to increase the value of k_p to improve the rise time. A gain of 3 cut the rise time nearly in half, while still remaining within the physically feasible time limit discussed earlier (see Figure 26).

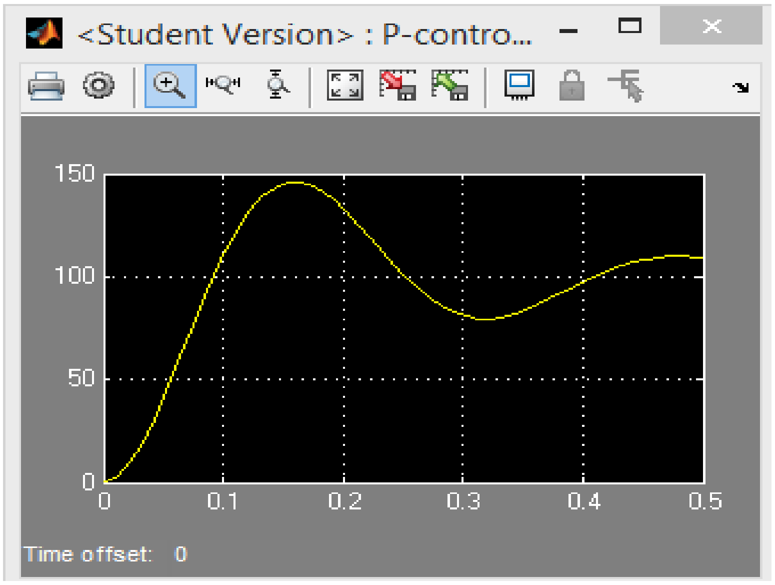


Figure 26: Simulated System Response with an Increased K_p for Improved Time Performance

Next I added the derivative control parameter and adjusted it to improve the system overshoot. A k_d value of 0.3 resulted in an overshoot of less than two percent, as shown in Figure 27.

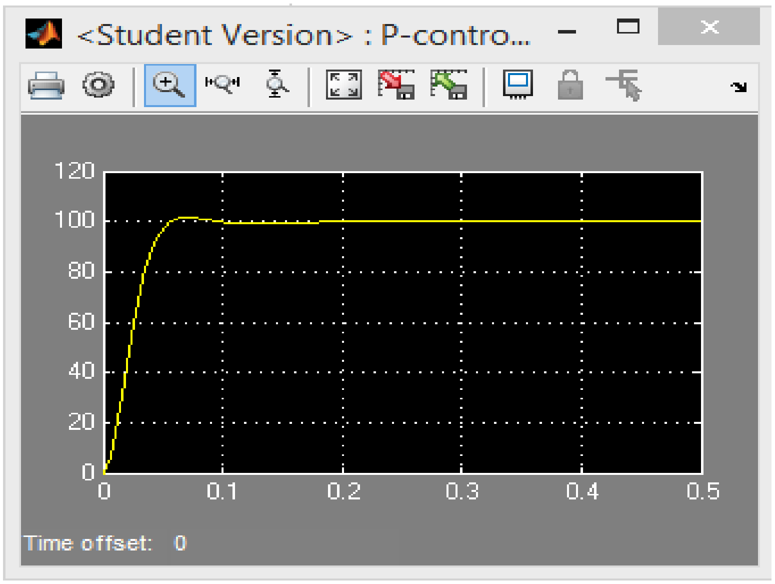


Figure 27: Simulated System Response with K_d Included for Decreased Overshoot

Finally, I added an integral control parameter to theoretically eliminate the steady-state error: k_i=1 appeared to accomplish this goal, but the shape of the simulated response appeared to be essentially unchanged otherwise.

4.7.2 PID Controlled Pivot Program Results

The results of the implementation of the PID controller with the parameters found through manual tuning on the Lego robot, as tested in the pivot program, showed a three percent steady-state error with no overshoot. This unexpected error could be attributed to the lack of precision in manual PID tuning, and acts as an advocate for the exploration of more formal techniques in the future: for example, the Ziegler-Nichols method.

A note here on the implementation of derivative control on the Lego EV3 motor: if the encoder sampling time was too slow, the rotation output of the motor was very jerky and did not match the simulated response; this was an issue when beginning to work with the robot. The Lego motor's encoder was originally set to a default sampling time of 0.1 seconds; when I lowered the sampling time to 0.01 seconds, the performance of the motor was much closer to the expected response.

4.8 Controller Selection and Testing within the Course Projects

As discussed under the design requirements for the controller, the most important performance characteristic for the Lego motor system in terms of a low error between the starting and ending position of the robot in the two course projects was the steady-state error. Therefore, although there is still room for improvement in its other performance characteristics, I decided to use the lead compensator for control of the individual Lego motor systems. To test its effectiveness, I compared the error between starting and ending positions when the courses were run with no control to the position errors when the lead compensator was applied to the motors.

4.8.1 Adjustment for Nonlinearity in the Course Projects

Since introductory controls engineering courses focus only on the control of linear systems, the design concepts explored in this chapter were only applicable up to a maximum

100 degree set point. To account for this, my Simulink models for the course projects – where the desired rotations are much higher than 100 degrees – were programmed as piecewise systems: maximum power was fed to the motor until the encoder's output was within 100 degrees of the desired rotation, and then control was applied to the individual motors for the final 100 degrees.. The following figures show selected screenshots from the Simulink model for course project 1.

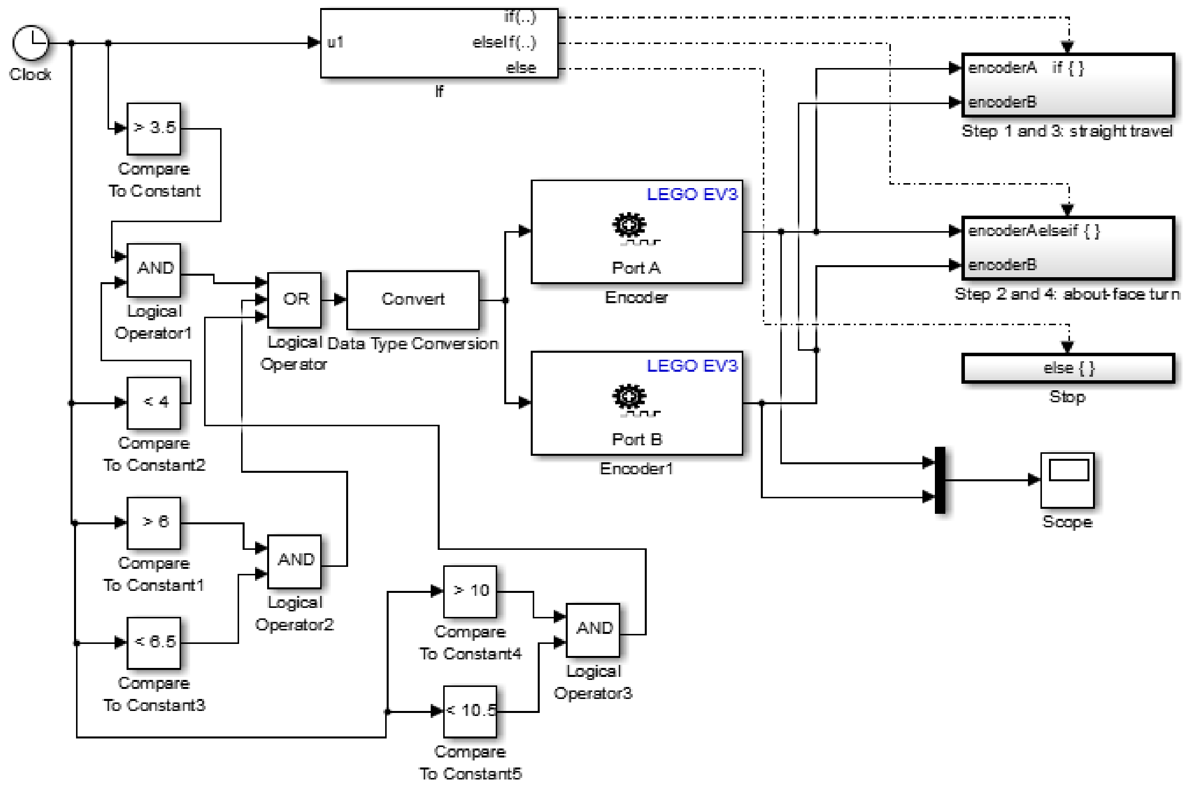


Figure 28: Simulink Model of the Step Transitions in Course Project 1

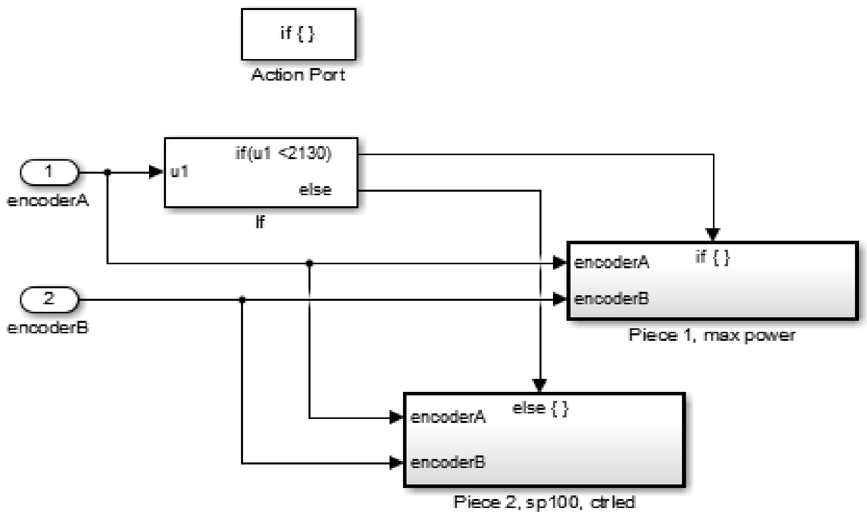


Figure 29: Simulink Subsystem for Piecewise Control on Straight Paths

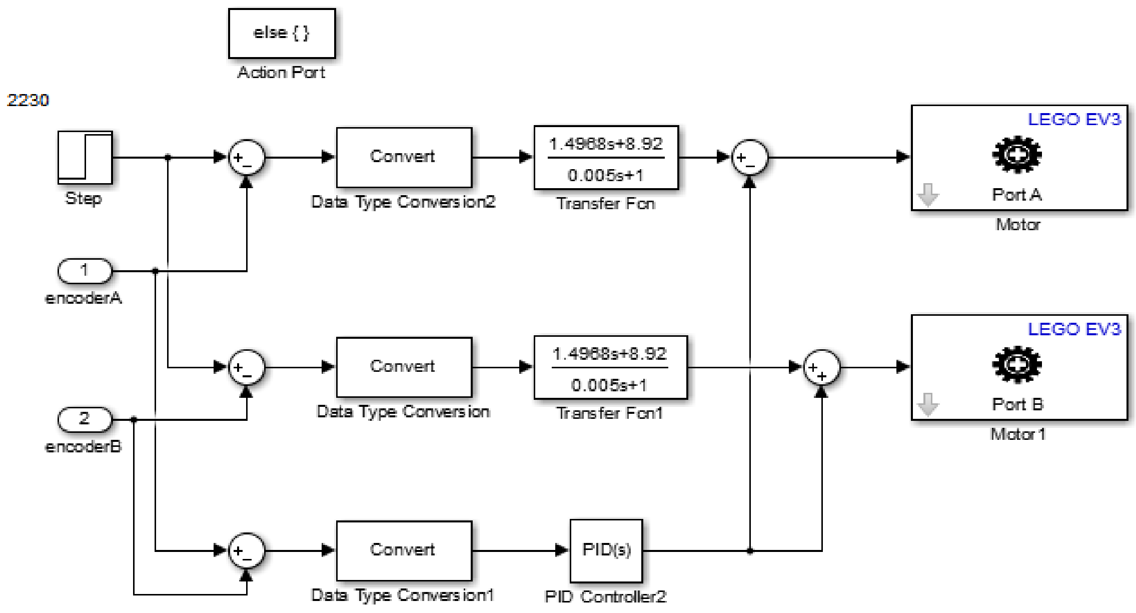


Figure 30: Simulink Model of Lead Compensation on the Final 100 Degrees of Rotation on the Straight Paths

4.8.2 Testing of the Controller Design within the Course Projects

After several trials with both of the course projects, controlled by the addition of my lead compensator, the worst errors between starting and ending position are shown in Figures 31 and 32. These position errors showed a definite improvement over the position errors of the course projects without the presence of control.

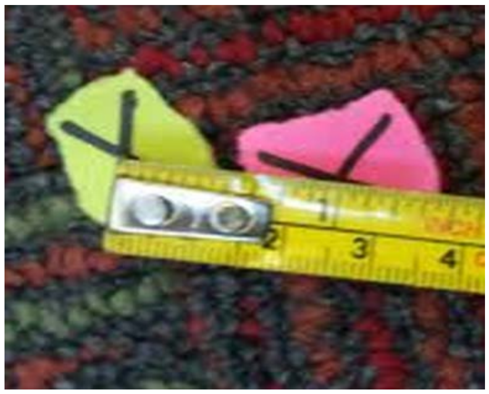


Figure 31: Position Error with Lead Compensation in Course Project 1

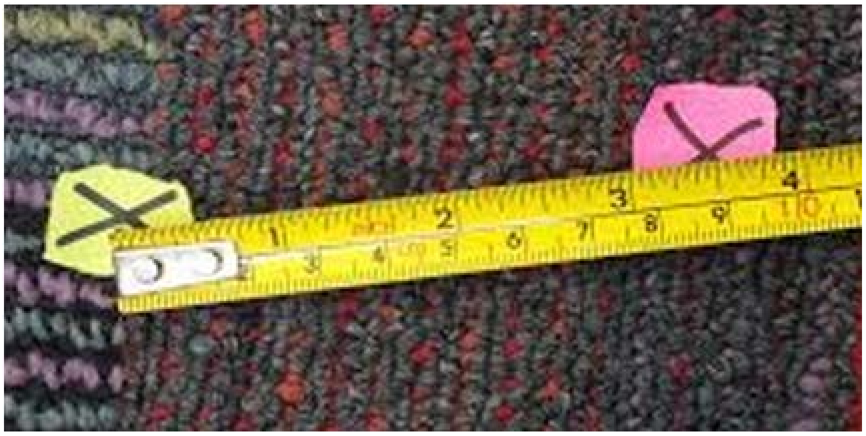


Figure 32: Position Error with Lead Compensation in Course Project 2

4.9 Controller Design Conclusion

The addition of control to the Lego EV3 motor system showed an improvement to its performance, as measured by a decrease in the starting and ending positions of the robot in the two Controls Systems course projects. As described earlier, lead compensator design is a recursive process. My focus in this initial design was on the steady-state error; as such I sacrificed some performance in the overshoot of the system, which lead to movement that was not very smooth. In fine-tuning the design of this controller in the future, the tradeoffs between error and overshoot would ideally be explored more thoroughly.

CHAPTER 5

CONCLUSION

The purpose of my research in this paper was to explore the practical applications of system modeling and controller design techniques on a real motor system using the Lego EV3 robot. I found a model for the motor system through experimentation and parameter estimation. Using this model, I designed a controller that improved the robot's performance, as tested using the two final course projects assigned to the University of North Texas Electrical Engineering Department's Controls Systems course. The techniques used, as well as the difficulties and discrepancies encountered during this process, served as an exploration of the practical applications of introductory theoretical engineering concepts.

The results of this research leave room for future work. Because the controller design studied in this paper was based off of system parameters estimated with the inclusion of friction effects as part of the internal dynamics -- versus as an external disturbance to the motor system -- the controlled performance of the Lego robot varies greatly with changes in travel surface. An important next step in controller design for this motor system, therefore, would be to design an adaptive, nonlinear controller, which was outside the scope of this paper. A more advanced controller could also help improve upon the variability of the motor system's performance. Hopefully, the results of this research will be a valuable starting point for the exploration of improved controller design techniques using the Lego EV3 robot.

References

[1] http://www.mathworks.com/help/supportpkg/legomindstormsev3/examples/getting-started-with-lego-mindstorms-ev3-hardware.html?prodcode=SL

[2] V. Sheth, M. Bokshi, Y. Wan, E. Pruett, C. Drotar, S. Fu, K. Namuduri, "A Test-bed for Multi-Domain Communication Networks using LEGO Mindstorms," *in proceedings of AIAA Infotech@aerospace Conference,* August 2013.

[3] G. Franklin, J. D. Powell, A. Emami-Naeini, *Feedback Control of Dynamic Systems,* Sixth Edition. Upper Saddle River, NJ: Pearson, 2010.

[4] Y. Kim, "Control Systems Lab Using a LEGO Mindstorms NXT Motor System," *Education, IEEE Transactions,* vol. 54, no.3, pp. 452-461, August 2011.

[5] http://www.dii.unisi.it/~control/ctm/PID/PID.html

[6] S. Wadoo, R. Jain, "A LEGO based Undergraduate Control Systems Laboratory," *in Systems, Applications and Technology Conference (LISAT), 2012 IEEE Long Island,* pp. 1-6, May 2012.

[7] http://higheredbcs.wiley.com/legacy/college/nise/0471794759/justask/SNT10060.html

[8] N. Maze, Y. Wan, K. Namuduri, M. Varanasi, "A Lego Mindstorms NXT-Based Test Bench for Cohesive Distributed Multi-agent Exploratory Systems: Mobility and Coordination," *in proceedings of AIAA Infotech@aerospace Conference,* 2012.