

Arduino control of ME 570 motor lab apparatus

by

Eric L Patterson

B.S., Kansas State University, 2000

A REPORT

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Alan Levin Department of Mechanical and Nuclear Engineering
Carl R. Ice College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2023

Approved by:

Major Professor
Jared Hobeck

Copyright

© Eric Patterson 2023.

Abstract

This paper first describes the existing “Motorlab” apparatus that is used for instruction in the Control of Mechanical Systems course, ME 570. It then documents the modification of the afore mentioned Motorlab to accommodate control by Arduino, allowing for full student programmability. A new lab exercise that utilizes Arduino control is then documented, along with the presentation of student reaction to the new exercise.

Table of Contents

List of Figures	vii
List of Tables	viii
Acknowledgements	ix
Chapter 1 - Introduction.....	1
Chapter 2 - Motorlab Hardware.....	2
2.1 Purpose.....	2
2.2 Brief History	3
2.3 Components	3
2.3.1 Motor.....	4
2.3.2 Amplifier.....	4
2.3.3 STM32 Board.....	5
2.3.4 Power Supply	6
Chapter 3 - Software	7
3.1 Microcontroller Code.....	7
3.2 MATLAB Interface	8
3.2.1 Controller Settings	8
3.2.2 Input Commands.....	9
3.2.3 Data Collection Settings	9
3.2.4 Motorlab GUI Actions	10
3.2.5 Data Matrix for the Motorlab GUI.....	10
3.2.6 Plotting Routines.....	10
Chapter 4 - ME 570 Laboratory.....	12
4.1 Common Laboratory Components.....	12
4.1.1 Assignment	12
4.1.2 MATLAB script.....	13
4.1.3 Narrative Quiz.....	14
4.2 Laboratory Exercises	15
4.2.1 Lab 2 – Estimating the viscous friction coefficient	16
4.2.2 Lab 3 – Feedback and Closed Loop Systems	17

4.2.3 Lab 5 – Second Order System	18
4.2.4 Lab 7 – Higher Frequency Dynamics	20
4.3 Student Feedback	21
Chapter 5 - Updating the Motorlab	22
5.1 Design Highlights	22
5.1.1 Current Monitor	23
5.1.2 Motor Encoder	24
5.1.3 Outputs	25
5.2 Construction	25
5.3 Arduino Software	26
5.3.1 Encoding	26
5.3.2 PWM	27
5.3.3 Control Loop	27
5.3.4 Interface	28
5.4 Arduino Testing and Behavior	28
Chapter 6 - Arduino Laboratory Exercise	32
6.1 Exercises	32
6.1.1 Coding	32
6.1.2 Integral Windup Control	33
6.1.3 Narrative Quiz	33
6.2 Student Feedback	34
Chapter 7 - Conclusion	36
References	37
Appendix A – Motorlab Handout (Revised 2013)	38
Appendix B – Arduino Code	57
Appendix C – Revised Schematics and Board Layouts	62
Motorlab STM32 Carrier Board Schematic	62
Motorlab STM32 Carrier Board Layout	63
Motorlab Arduino Shield Schematic	64
Motorlab Arduino Shield Board Layout	65
Appendix D – Student Survey Results	66

Appendix E – Selected Laboratory Exercises.....	68
Laboratory 2.....	68
Laboratory 3.....	72
Laboratory 5.....	76
Laboratory 7.....	81
Arduino Laboratory	87

List of Figures

Figure 2.1 Motorlab Apparatus.....	2
Figure 3.1 Motorlab GUI Primary Interface	8
Figure 3.2 Workspace Data Variable Structure	10
Figure 3.3 Mlolplots Function Display	11
Figure 4.1 Lab Exercise 1 Information Block	13
Figure 4.2 Example MATLAB from Lab 7	14
Figure 4.3 Narrative Quiz Questions Example	15
Figure 4.4 Lab Exercise 2 Information Block	16
Figure 4.5 Lab 2 Partial Results.....	17
Figure 4.6 Lab 3 Response Plots.....	18
Figure 4.7 Lab 4 Information Block	19
Figure 4.8 Lab 7 Information Block	20
Figure 4.9 Open and Closed Loop Pole Plot.....	21
Figure 5.1 Original Resistor Network.....	23
Figure 5.2 Arduino Board Resistor Network.....	24
Figure 5.3 Arduino User Interface	28
Figure 5.4 STM32 vs Arduino Results	29
Figure 5.5 Simulink Model	30
Figure 5.6 Physical and Simulated Results.....	30

List of Tables

Table 2.1 Manufacturer Motor Specifications	4
Table 4.1 Laboratory Exercise Summary	15

Acknowledgements

I would like to thank Dr. J.K. Shultis and Dr. Douglas McGregor for starting me down the grad school path. This degree may not be related to nuclear engineering, but without these two, I would not have started down the path. I would also like to thank Dr. Dale Schinstock, who rekindled my passion for manipulating the physical world with computers and related hardware. Thanks also to Dr. Jared Hobeck, who was willing to step in and see me to the end of my journey towards my master's degree. And most importantly, my heartfelt gratitude goes out to Dr. Warren White. He was as patient as the day is long and was always able to make complicated concepts seem simple. His passing left a void in the lives of everyone he touched.

Chapter 1 - Introduction

This report will provide an overview of the equipment and exercises involved in the laboratory section of ME 570, Control of Mechanical Systems I course. The primary experimental apparatus, or “Motorlab”, will be described. A selected set of exercises will be described, providing a taste of what the students experience in the ME 570 laboratory section. From there the direction will shift into student feedback, and the expressed desire for a way to break away from the feeling that the equipment hides too much. Modifications of the Motorlab system to allow for Arduino control were developed and will be explained, as well as a new lab exercise that utilizes the revised Motorlab.

Chapter 2 - Motorlab Hardware

2.1 Purpose

The Control of Mechanical Systems I course, referred heretofore as ME 570, offered by the Alan Levin Department of Mechanical and Nuclear Engineering at Kansas State University provides instruction in line with most introductory controls courses. The course description is as follows: “Introduction to modeling and control of dynamic systems encountered by Mechanical Engineers. Topics include basic linear systems modeling and analysis; feedback control; time response and stability of dynamic systems; introduction to root locus and frequency response design.” While all these topics are thoroughly covered in the lecture setting, retention and deeper understanding is better achieved by the hands-on experience provided by the companion laboratory section.

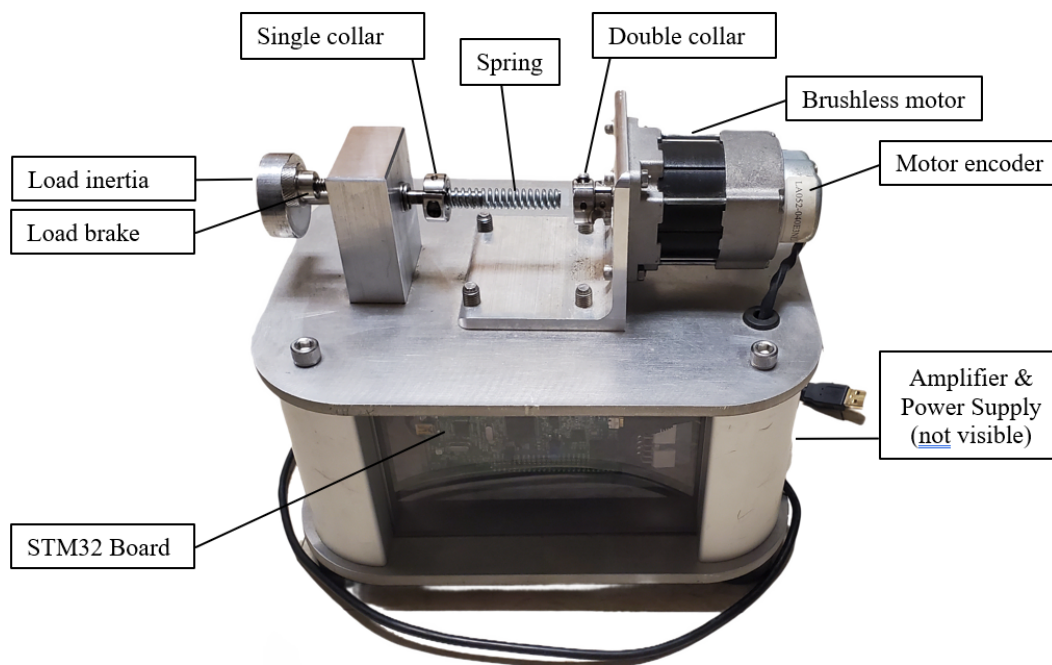


Figure 2.1 Motorlab Apparatus

2.2 Brief History

The first version of the Motorlab was created in the summer of 2002 by Dr. Dale Schinstock and Dr. Warren White at Kansas State University. The basic components were a LA052-040 brushless motor from Shinano Kenshi, a model 503 DC brushless servo amplifier from Copley Controls, a Renco RCML15 low profile encoder, a generic 24V DC power supply and a Precision MicroDynamics MC4000 DSP card [1].

The second version was created as part of a fall 2007 student honors project and placed into service in 2010. It replaced the MC4000 DSP card with a PC running National Instruments NI RT real-time operating as a cost saving measure. The user interface was also changed to use National Instruments LABView [1].

The third and fourth iterations replaced the real-time computer with a STM32F4Discovery board which contains a STM32F047VGT6 microprocessor. The third generation was created by Dr. Dale Schinstock in 2013, and employed a separate box to house the STM32 board [2]. The fourth generation, pictured in Figure 2.1, was created in the summer of 2016 by Dr. Schinstock's student, Shane Smith, and myself [3]. The major difference in the latter system is that it houses all components together. It also has no load encoder, which was underutilized. The STM32 firmware was not changed.

2.3 Components

The basic components that make up the Motorlab are the LA052-040E brushless motor and optical encoder, the Copley 503 brushless servo amplifier, or simply "amplifier", the STM32F4 Discovery board, referred to typically as the STM32 board or simply STM32, and the 24V DC power supply.

2.3.1 Motor

The motor chosen is a Shinano Kenshi LA052-040E 3-phase brushless DC motor with included optical encoder.

Table 2.1 Manufacturer Motor Specifications

Specification	Value	Unit
Model	LA052-040E	
Mfg	Shinano Kenshi Corporation	
Rated Power	40	W
Rated Voltage	24	V
Rated Speed	3000	rev/min
Rated Torque	0.127	N·m
Rated Current	2.5	A
Torque Constant	0.05	N·m/A
Back EMF Constant	0.05	V·s/rad
Phase Resistance	1.18	Ω
Phase Inductance	4.4	mH
Instantaneous Peak Torque	0.382	N·m
Max Speed	5000	rev/min
Rotor Inertia	1.1×10^{-5}	kg·m ²
Power Rate	1.48	kW/s
Mechanical Time Constant	5.2	ms
Electrical Time Constant	3.7	Ms
Encoder Resolution	1600	Counts/rev

2.3.2 Amplifier

The amplifier is a Copley Controls model 503 DC Brushless Servo Amplifier. It is designed to operate a 3-phase brushless DC motor with Hall effect sensor commutation in torque (current) mode. It is capable of delivering 5A continuous and 10A peak although care is taken to

limit the output current to a maximum of 3A as to not damage the motor. The amplifier is powered with 24VDC.

The motor current is commanded with a differential pair of voltage inputs that feed into an internal difference amplifier. The transfer function is 1 A/V by default. The output is a 25 kHz PWM modulated 3-phase current. The manufacturer stated closed-loop bandwidth of the amplifier is 2.5kHz with a 200uH load. Experiments were conducted with the LA052-040E and 24V supply that show the bandwidth of the amplifier as we use it is closer to 400 Hz. See Appendix A for the bandwidth analysis and amplifier technical documents. The amplifier also has a current monitoring port that provides a voltage signal of 0.5 V/A so that the motor current can be monitored.

2.3.3 STM32 Board

The real time calculations and computer communications are handled by a STM32F4DISCOVERY evaluation board. It was chosen because it is a 32-bit processor, it's fast, it has a 12-bit DAC, and it has a math co-processor. The CPU frequency is 168Mhz which allows for a relatively complicated controller that can be run in a 10kHz control loop. In addition, the math co-processor ensures any mathematics functions, like filters, minimize any burden on the CPU. The 32-bit processor shortens the CPU time required to operate on floats, doubles and long datatypes and allows for better precision. Furthermore, it has timers capable of A B encoder decoding and supports PWM frequencies beyond 100kHz [4].

The STM32 board is plugged into a custom circuit board that contains a 3.3V voltage regulator, resistor network to map the current monitoring input from the amplifier to voltages the STM32 board can read, and connectors for power, the motor encoder and connections from the amplifier.

2.3.4 Power Supply

The power supply is a CUI VF-D320-D524A-CFS 24V, 320W power supply. It provides 24Vdc at a maximum 8.33A. It also provides 5Vdc to power the STM32 board. This model is now obsolete, but newer models can be fitted.

Chapter 3 - Software

The Motorlab hardware is only part of the whole system. The interface and the code that runs on the microcontroller are nearly as important as the hardware. Poorly executed code can obstruct the learning process and lead to confusion and disillusionment.

3.1 Microcontroller Code

The microcontroller code serves two purposes. Control of the Motorlab hardware, and interfacing with the host PC. The PID function is easily recognizable – proportional, integral and derivative terms:

```
float pid(float Kp, float Ki, float Kd,
          float error, float *integral,
          float DT, float Umax)
{
    static float lastError;
    float U, USat;

    *integral = *integral + Ki*DT*error;
    USat = Umax;
    if (*integral > USat) *integral = USat;
    if (*integral < -USat) *integral = -USat;

    U = *integral + Kp*error + Kd*(error-lastError)/DT;
    USat = Umax;
    if (U > USat) U = USat;
    if (U < -USat) U = -USat;

    lastError = error;
    return(U);
}
```

In addition to the PID function, there is selection logic that chooses open-loop, position and velocity modes. There is also a velocity filter to smooth spikes that can happen when calculating the velocity. The vast majority of the firmware, however, is the implementation of a rudimentary operating system that handles timing and communication with the host PC via virtual COM port.

3.2 MATLAB Interface

The student interface is implemented in a collection of MATLAB m files that are kept in a folder that is in MATLAB's search path. The students launch it using the command `MotorlabGUI()`. The primary interface has several sections which are boxed on the GUI for convenience.

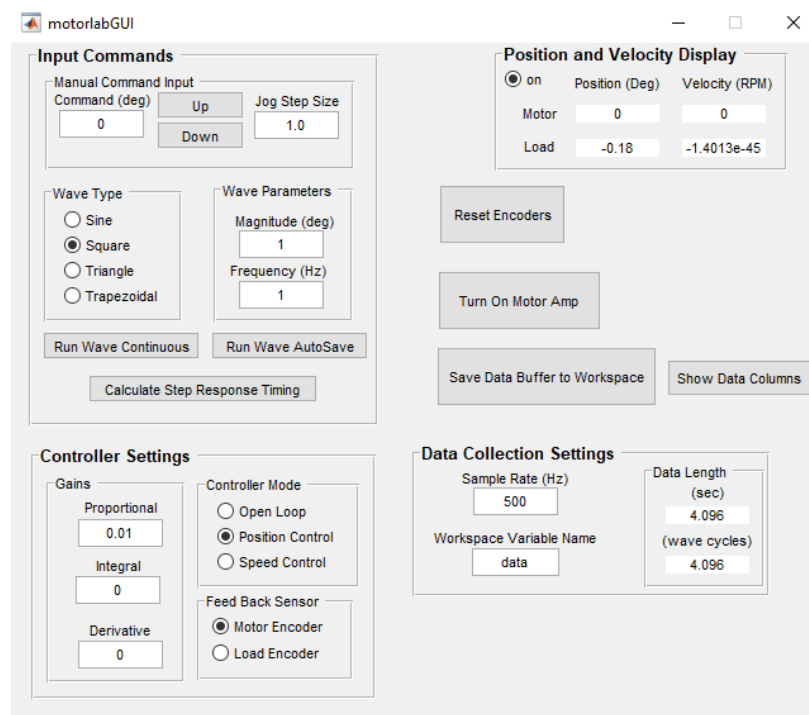


Figure 3.1 Motorlab GUI Primary Interface

3.2.1 Controller Settings

This is where the type of controller is defined. There are three blanks that correspond to the proportional control constant, K_p , the integral control constant, K_i , and the derivative control constant, K_d . The controller mode can be set to one of three modes. The first is Open Loop, which makes all commands currents (in amps). The next is Position Control which switches the commands to degrees. Finally, the Speed Control commands in revolutions per minute. The

feedback sensor is left on Motor Encoder since the load encoder is no longer present on the current revision of the Motorlab.

3.2.2 Input Commands

The input commands, both manual and wave, track the units selected in the controller settings – amps, degrees or rpm. The Wave Type section defines a periodic wave type that can be commanded to the motor continuously, or in the case of “Run Wave AutoSave” run in a one-shot configuration for the amount of time displayed in the Data Length blank in the Data Collection Settings section. The third button in this section is the Calculate Step Response Timing button. This is used in conjunction with the square wave type to implement a low, then high “step”. The data buffer used in the Motorlab is 2048 floats wide, and so if the Sample Rate were set to 500 Hz, it would take 4.096 seconds to fill the data buffer. If we only want one half of a complete square wave, essentially a step, we want the period of the square wave to be twice the time it takes to fill the buffer. So, this button calculates the required frequency of the square wave based on the data sample rate and the 2048 wide buffer: $F = \frac{SampleRate}{2 * 2048}$. We do this strictly as a convenience for the student.

3.2.3 Data Collection Settings

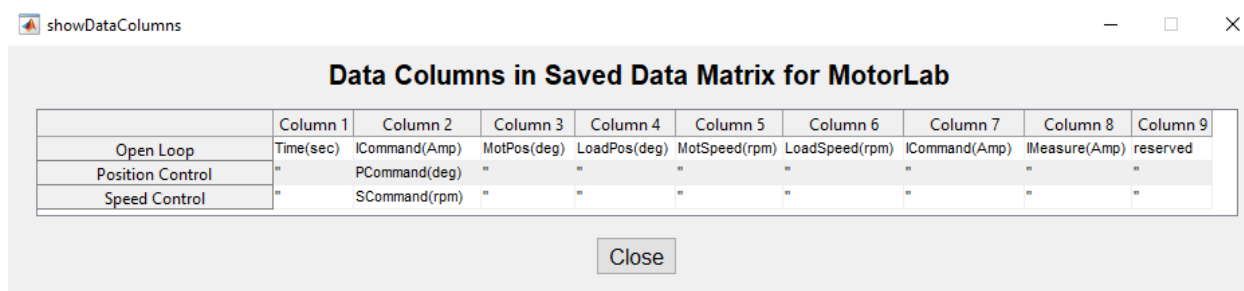
The rate at which the Motorlab parameters are sampled for input into MATLAB is set here, as well as the name of the variable that is set in the MATLAB workspace once the data is collected. The Data Length and wave cycles parameters are also calculated based on wave frequency and collection rate and displayed for informational purposes.

3.2.4 Motorlab GUI Actions

In addition to the previously described sections, there are four buttons on the Motorlab GUI. The first, Reset Encoders, clears the motor position encoder back to zero. The “Turn On Motor Amp” button enables and disables the amplifier. The Save Data Buffer to Workspace will save the current contents of the data buffer, which runs continuously, to the variable defined in the Workspace Variable Name control. The fourth button is strictly informational, and describes the workspace variable structure.

3.2.5 Data Matrix for the Motorlab GUI

The data variable that is set in the workspace by the Motorlab GUI is simply a 2048 row matrix with columns representing time, commanded value (current, position or speed), motor and load position, motor and load speed, commanded and measured current, and a reserved field for expansion.



The screenshot shows a window titled 'showDataColumns' with a close button. Inside, a table titled 'Data Columns in Saved Data Matrix for MotorLab' displays the structure of the data matrix. The table has 10 columns: Column 1 through Column 9, and a 'reserved' column. The rows represent different control modes: Open Loop, Position Control, and Speed Control. The 'Open Loop' row shows Time(sec), ICommand(Amp), MotPos(deg), LoadPos(deg), MotSpeed(rpm), LoadSpeed(rpm), ICommand(Amp), IMeasure(Amp), and reserved. The 'Position Control' row shows PCCommand(deg) and reserved. The 'Speed Control' row shows SCommand(rpm) and reserved.

	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8	Column 9
Open Loop	Time(sec)	ICommand(Amp)	MotPos(deg)	LoadPos(deg)	MotSpeed(rpm)	LoadSpeed(rpm)	ICommand(Amp)	IMeasure(Amp)	reserved
Position Control	"	PCCommand(deg)	"	"	"	"	"	"	"
Speed Control	"	SCommand(rpm)	"	"	"	"	"	"	"

Close

Figure 3.2 Workspace Data Variable Structure

3.2.6 Plotting Routines

There are three plotting routines that accompany the Motorlab GUI. One for each mode of operation. Open loop operation uses the `mlolplots(data)` function and includes plots of

commanded current and angular position vs time, commanded current and speed vs time, and both commanded and measured current vs time.

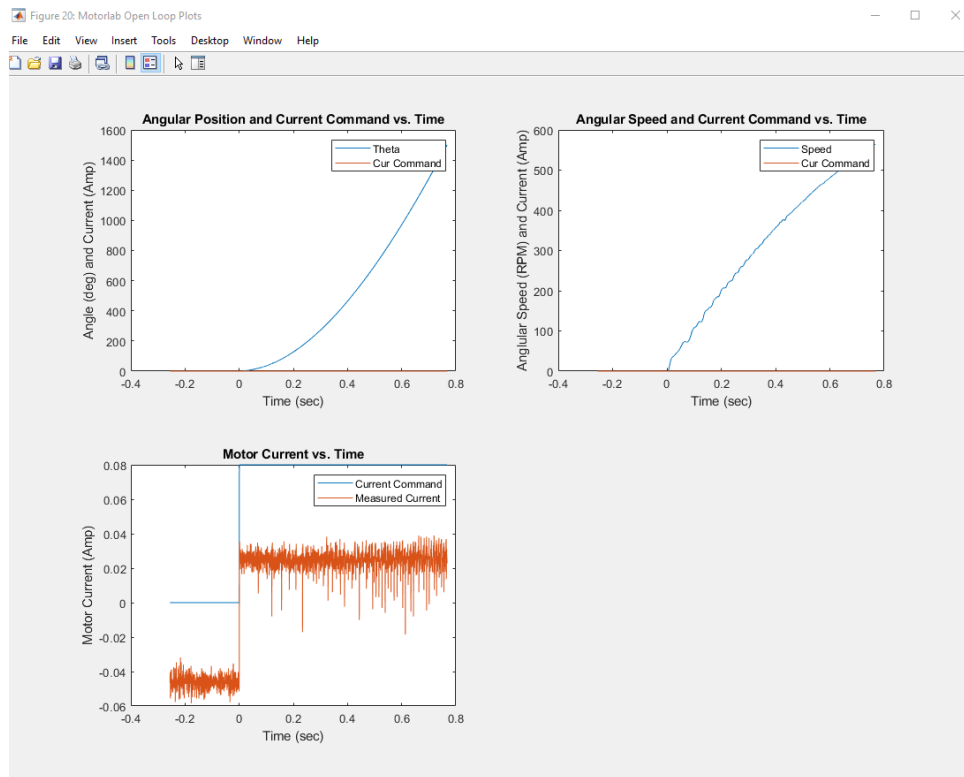


Figure 3.3 Mlplots Function Display

The other two plotting routines are `mlposplots()` and `mlspeedplots()`, which correspond to position and speed plots respectively.

Chapter 4 - ME 570 Laboratory

The introductory controls class for mechanical engineering students at Kansas State University is ME 570. The class syllabus describes the class as follows:

Introduction to modeling and control of dynamic systems encountered by mechanical engineers. Topics include basic linear systems modeling and analysis; feedback control; time response and stability of dynamic systems; introduction to root locus and frequency response design.

Many these concepts are difficult to grasp when disconnected from the physical systems that they describe. Therefore, like many other mechanical engineering programs, this class has a hands-on laboratory. It meets for two hours, once a week through the semester. Each week a different exercise is conducted and, with only two exceptions, the Motorlab apparatus is operated to reinforce the concepts covered in lecture.

4.1 Common Laboratory Components

Each lab period starts with the downloading of the day's assignment document and MATLAB script. Then the instructor gives an overview of the exercises to be completed and the students complete them. Finally, the students take an online quiz over the material just covered.

4.1.1 Assignment

Each week, the students are given an assignment document that fits generally the same format each week. One of the most important parts of the assignment is the information block. The information block, which is placed at the top-right of each exercise, contains a block diagram and definitions of the constants and formulas that are required to complete the exercise.

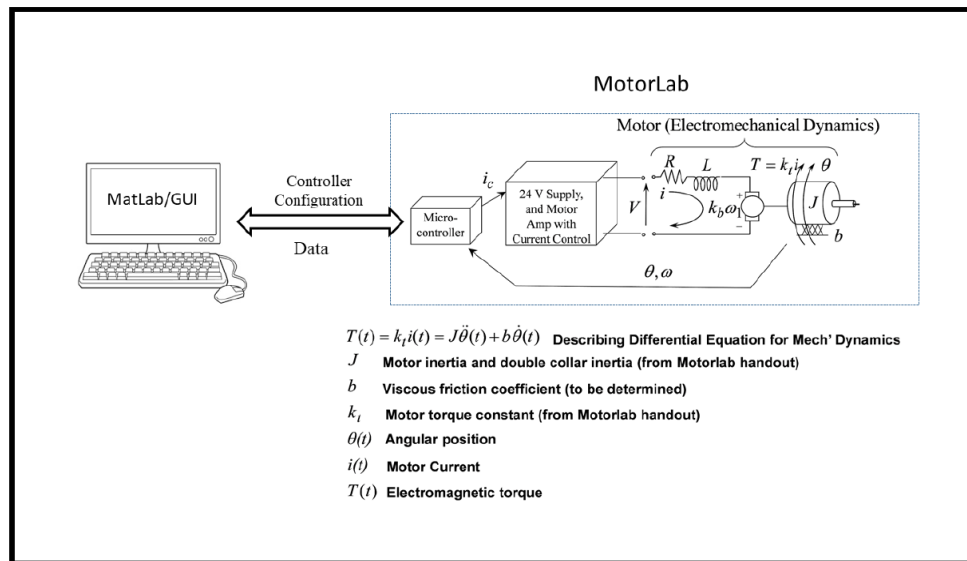


Figure 4.1 Lab Exercise 1 Information Block

Next is a description of what is to be done, along with instructions needed to complete the exercise. There may be tables to fill in, equations to solve, plots to generate, etc. There is also a “What to turn in” list. At the end of the assignment is typically a copy of the MATLAB code needed for the lab.

4.1.2 MATLAB script

Each assignment has a MATLAB live script that the students download and work with. In the first labs, the script is nearly complete, but as the semester progresses, the students are expected to do more and more of the coding. MATLAB live script is used to enable rich content that includes equations, Greek characters and interactive plots.

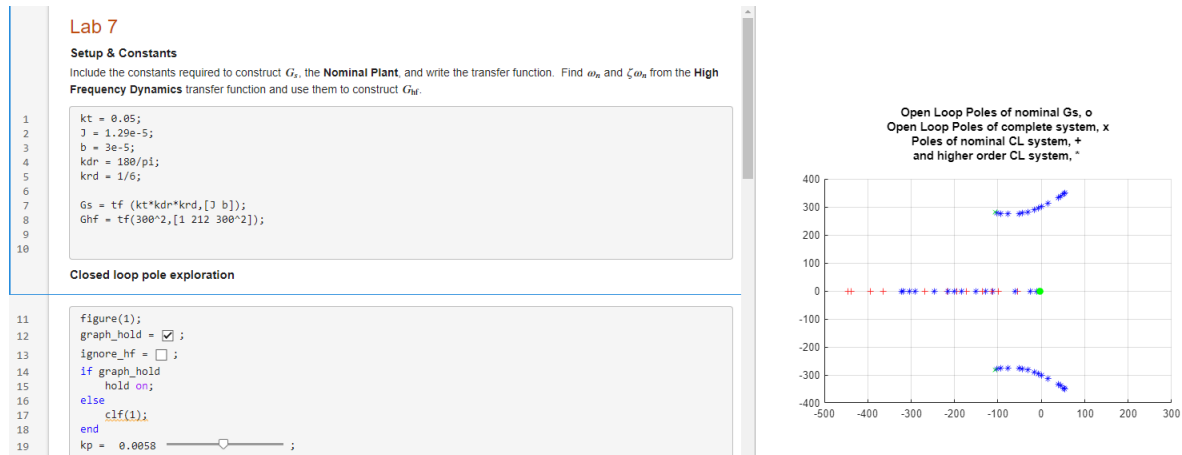


Figure 4.2 Example MATLAB from Lab 7

4.1.3 Narrative Quiz

The last portion of the laboratory is the narrative quiz. The purpose of the quiz is to make the students think about the exercise they just completed and hopefully gain better insight into the work they have just done. The quiz was originally in paragraph, or “narrative” form and was simply written on the handout with free-form blanks. We have since moved the quiz to an online format that the students seem to prefer.

Question 1
15 pts

In the first plot we can see the relationship between motor speed and the friction torque. With a constant motor current, the [Select] produced by the magnetic field is also constant. In this case the speed settles to a fairly constant value where the input/magnetic torque balances with the torque caused by [Select]. The typical model of friction for control design is [Select] torque is proportional to speed. This allows us to use it in linear diffe [Select] data points in the plot show that the actual friction has a [Select] component along with the linear component, resulting in zero velocity with small values of constant torque. Our estimate of the viscous (linear) coefficient of friction roughly capturing both the viscous and static friction effects is [Select].

Figure 4.3 Narrative Quiz Questions Example

4.2 Laboratory Exercises

The following table contains a summary of the exercises we conduct throughout the semester. This list was current through the fall 2021 semester, though on occasion the order of exercises is shuffled. In addition, there is a practical examination period in which the students are required to show proficiency in MATLAB and with the Motorlab, and there is a project where the students are given a system and must write a lab exercise around it.

Table 4.1 Laboatory Exercise Summary

Exercise	Motorlab Mode	Concepts Covered
1	-	MATLAB coding, familiarization with Motorlab
2	Open Loop	Friction (static and viscous), Torque
3	Position	Feedback, proportional and derivative control
4	-	Simulink, Friction (static and viscous)
5	Open Loop	2 nd order systems
6	Position	Position Control, 1 st and 2 nd order relationship
7	Velocity	Higher frequency dynamics – pole and zero visualization

8	Velocity	Integral control, system type
9	Position	Derivative control, SISOTOOL, root locus design
10	Velocity	Controller design, SISOTOOL, overshoot, settling time
11	Open Loop	Spring, resonance, model determination, bode plots
12	Velocity	Frequency response, bode plots
13	Velocity	PI tuning, Ziegler-Nichols

The following sections provide more detail for a selected set of labs that better illustrate just how the Motorlab is used in class. The complete lab handouts can be found in Appendix E.

4.2.1 Lab 2 – Estimating the viscous friction coefficient

This is the first lab where the Motorlab is used to collect data. The students are responsible for finding the viscous friction coefficient, b , and must generate an initial condition response from the Motorlab and compare it with theoretical results.

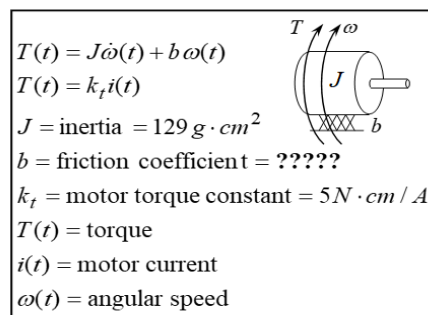


Figure 4.4 Lab Exercise 2 Information Block

First, the students are asked to solve the differential equation described in the information block to determine b for any static velocity:

$$T(t) = J\dot{\omega}(t) + b\omega(t) = k_t i(t)$$

$$b = k_t \frac{i(t)}{\omega(t)}$$

Then the students construct a vector of speeds that correspond to a given vector of commanded currents in open-loop mode. They then find b by adjusting the slope of a line that so that it fits the data they collected. Then the students capture data for the Motorlab using the Motorlab GUI as the motor decays from the speed produced by a 0.15 Amp current to zero when the current is removed. They then compare the collected data with the theoretical data.

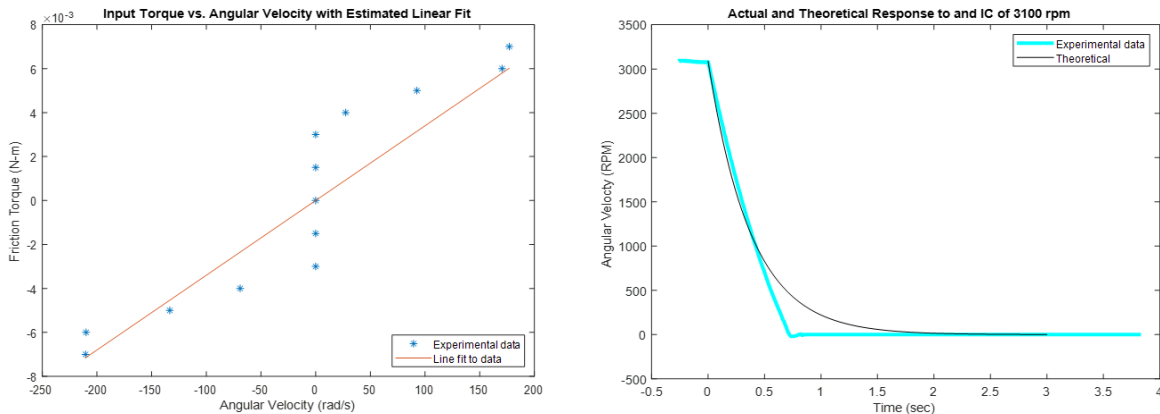


Figure 4.5 Lab 2 Partial Results

4.2.2 Lab 3 – Feedback and Closed Loop Systems

This lab introduces the concepts of feedback and controllers. The first exercise is using an iterative approach to find the closed loop system based on the block diagram. The students are given the first equation, $E(s) = \theta_c(s) - \theta(s)$. They are then asked to progress through the block diagram, substituting the previously solved value thusly:

$$\begin{aligned}
I_c(s) &= \text{_____} E(s) \\
I(s) &= \text{_____} I_c(s) \\
T(s) &= \text{_____} I(s) \\
\theta_r(s) &= \text{_____} T(s) \\
\theta(s) &= \text{_____} \theta_r(s)
\end{aligned}$$

If the students are observant, they will notice that the system is already solved for them in the information block so they have a solution to work toward.

The next task is to try two different controllers. One with proportional control only (Kp) and one with both proportional and derivative terms. They will collect a step response from the Motorlab and plot the two responses.

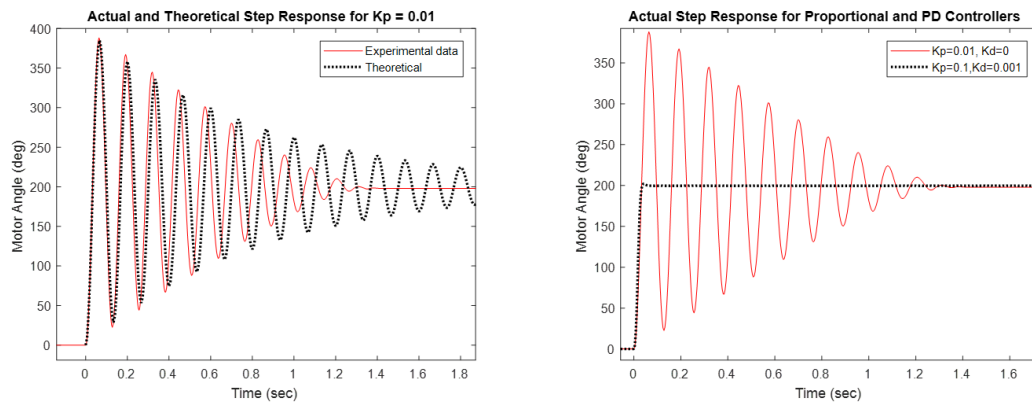


Figure 4.6 Lab 3 Response Plots

This time they have to complete more of the MATLAB code. This amount of code given to the students will continue to decline through the semester, helping them toward proficiency. Finally, they will also complete the narrative quiz.

4.2.3 Lab 5 – Second Order System

This lab introduces the spring into the system, creating a plant with a second order character.

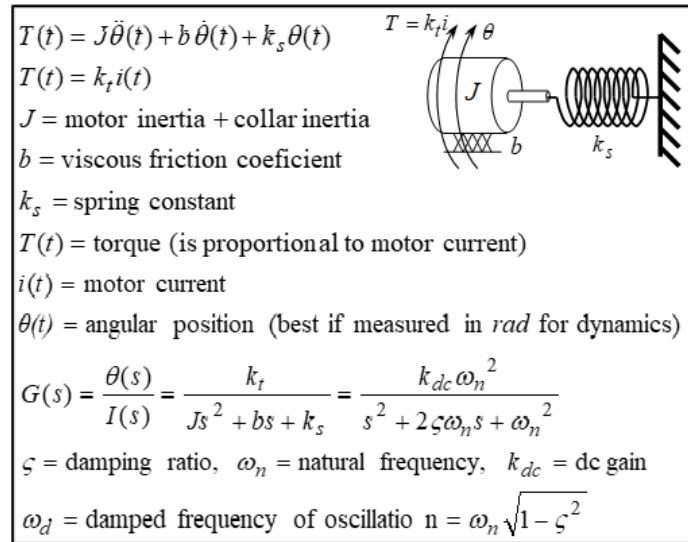


Figure 4.7 Lab 4 Information Block

The concepts of damping ratio, dc gain, and natural and damped frequencies are first explored here. The idea of a standard second order form is also introduced.

$$G(s) = \frac{k_{dc}\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

In this lab the students fix the spring in place with the brake thumbwheel. They then construct a position vector that corresponds to a variety of open-loop currents and use MATLAB to determine the spring constant, k_s . Then the students perform a 1 Amp step response, analyze the resulting position plot, and gather the information needed to find k_{dc} , ζ , and ω_n . The students then complete the MATLAB code needed to construct a plot of the measured step response along with a theoretical response using the standard second order form. They finish up, again, with the narrative quiz.

4.2.4 Lab 7 – Higher Frequency Dynamics

Most of what the microcontroller does is hidden from the students. However, in this lab we expose part of the system that is software derived on purpose to teach about higher frequency dynamics.

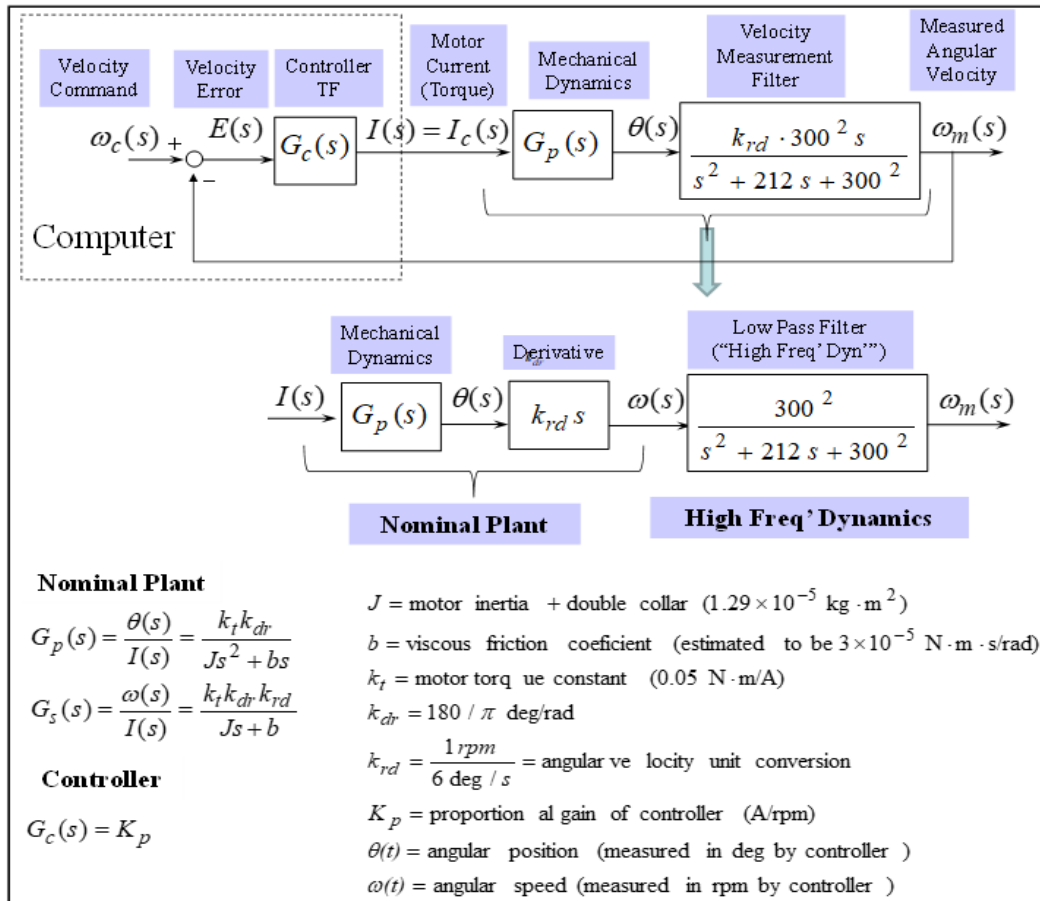


Figure 4.8 Lab 7 Information Block

As explained previously, there is a filter in the microcontroller firmware that serves to smooth spikes that can occur when taking the time derivative of the position encoder. In this lab, we give the students a block diagram that includes the filter so that they can visualize when open-loop high frequency poles can be safely ignored. This lab also serves to introduce the root-locus plot before they are formally introduced to it by creating a plot of open loop and closed loop poles with an adjustable K_p term in MATLAB live script.

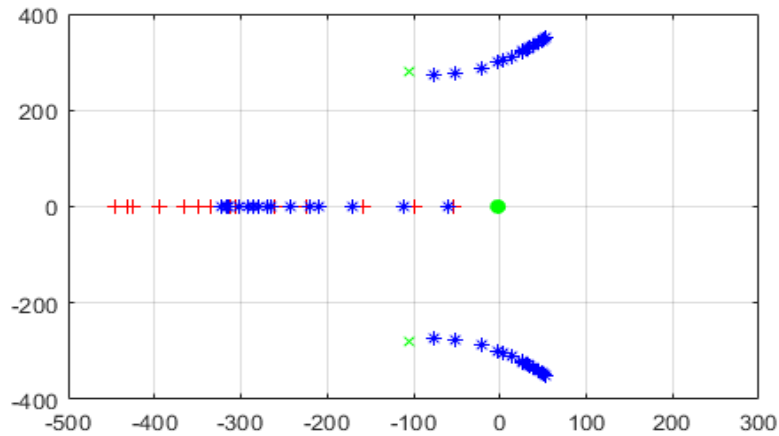


Figure 4.9 Open and Closed Loop Pole Plot

4.3 Student Feedback

After casually polling students, it became clear that while the students generally prefer the hands-on learning in lab over lecture, they still felt that the Motorlab experiments were a little to “canned”, especially later in the semester when they work with designing controllers and tuning. Of course, during COVID, they had to make due with videos and simulations of the Motorlab, but in the fall of 2021, the students were asked if they would appreciate being able to control the motor with the Arduino, which they had all used in ME 400, the introductory programming and numerical methods class. The response was universally positive.

Chapter 5 - Updating the Motorlab

Since the students had overwhelmingly expressed an interest in coupling the Arduino platform with the Motorlab in the Fall of 2021, work was started to retrofit all of the Motorlab systems by mid-semester of Spring 2022.

5.1 Design Highlights

The primary concern was to not change any of the existing characteristics of the Motorlab which could disrupt the existing lab exercises. At the same time, Arduino control had to be achieved without making extensive modifications to each station when required. The first step was to take inventory of all the hardware connections between the amplifier, the motor encoders, and the STM32 board. There are two inputs into the STM32 board and two outputs. The current monitor output of the amplifier is one input. The incremental encoder mounted on the rear of the motor provides the other input. The first output is a digital line that enables or disables the amplifier, and the other output is a pair PWM generated voltages connected to the amplifier that to command the motor current.

Nearly all of the students have used the Mega 2560 Rev 3 boards for ME 400, making them a natural choice. Although the Mega could be a little faster and the A/D converter could have better resolution.

Next, the physical layout and connection was considered. Since the Motorlab will not be routinely used with the Arduino, it was decided to tether the Arduino to the Motorlab so that it can be plugged and unplugged as needed. The Motorlab carrier board was modified to include necessary additional circuitry, as well as an offboard connector. An 8-Pin Mini-DIN circular connector and cable were employed since they are unlikely to be confused with any other typical

connectors such as USB or RS232 serial port cables. The Arduino board would basically be a shield with mating connector and remaining circuitry.

5.1.1 Current Monitor

The current monitor output from the amplifier provides a voltage level that corresponds to the current consumption of the motor. It has a 2A/volt output with a 10k Ω and 3.3nF RC low pass filter. Since the output can swing between $\pm 5V$, and the STM32 can only read voltages up to 3.3V, a resistor network is employed to map the bipolar voltage into one that the microcontroller can process.

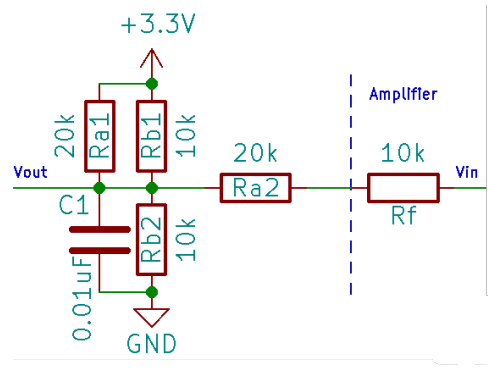


Figure 5.1 Original Resistor Network

The relationship between V_{in} and V_{out} is:

$$V_{out} = \frac{\frac{V_{in}}{R_f + R_a} + \frac{3.3}{R_a} + \frac{3.3}{R_b}}{\left(\frac{1}{R_f + R_a} + \frac{1}{R_a} + \frac{2}{R_b} \right)}$$

Therefore, the maximum current, $\pm 3A$, maps from 1.15706V to 1.9235V. The STM32 samples this voltage and applies the following equation to convert V_{in} to current:

$$I = -17.402V_{out} + 30.4211$$

The voltage, V_{out} , could be sent to the Arduino without modification, but I decided that it would be best to send the original 2A/V bipolar signal out of the Motorlab so other hardware could be easily used. I accomplished this using two components, a small 5V to $\pm 12V$ DC-DC

converter module and an OPA145IDBVT op-amp configured as a voltage follower to isolate the output of the amplifier and its $10\text{k}\Omega$ output impedance from the resistor networks on the STM32 board and the Arduino board.

On the Arduino board, the 2A/volt signal from the Motorlab is processed with a simpler resistor network. This network has a much simpler transfer function, though cannot map the full $\pm 5\text{V}$ range from the amplifier to voltages usable by the Arduino. However, since the amplifier is limited to 3A, the maximum voltage range should not exceed $\pm 1.5\text{V}$.

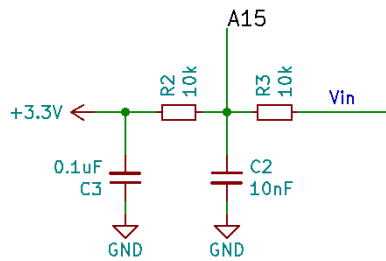


Figure 5.2 Arduino Board Resistor Network

The V_{out} (A15) to V_{in} relationship in this circuit is:

$$V_{out} = \frac{3.3 + V_{in}}{2}$$

In most cases it is not actually necessary for the Arduino to process the monitor current. The STM32 will still collect the current readings and report them to MATLAB for analysis in either case.

5.1.2 Motor Encoder

The incremental encoder on the rear of the motor produces A and B signals that determine position and direction. The STM32 has the ability to process quadrature signals with timers, but the Arduino has no built-in quadrature decoding. A significant amount of processing time would be spent in interrupt routines decoding the A and B signals in the Arduino.

Therefore, instead of using the Arduino for decoding, the LS7184N-S chip is used on the Arduino board which decodes A and B quadrature signals into CLOCK and UP/ $\overline{\text{DN}}$ signals. The CLOCK signal is wired into pin 47, a counter pin, and the other is wired into pin 51, which can process pin change interrupts. This scheme only interrupts the processor when there is a change in direction instead of every single change of A and B.

5.1.3 Outputs

The output current of the amplifier is controlled with a differential pair of inputs that are generated with a PWM signal in both microcontrollers. The transfer function is a simple 1:1 relationship between voltage and current. Directionality is achieved by holding one half of the differential pair low while the other is modulated. In order to select between the STM32 and Arduino outputs, a MC74LCX157DR2G 4 channel MUX with 5V safe inputs was chosen. Both the PWM signals and the amplifier enable pins are multiplexed through this chip. The A/B selection pin of the MUX is pulled low normally, but when the Arduino board is plugged in, a 5V signal is sent to the A/B selector to switch the PWM and enable pins to the Arduino.

5.2 Construction

The schematics and circuit board layouts were created using the open source KiCad package. The circuit boards were manufactured by PCBWay - www.pcbway.com. Each Motorlab was then taken apart and the STM32 carrier board was replaced. Rather than allow each student to use their own personal Arduino, 18 Arduinos were fitted with the new shield and had initial firmware placed on them. They are stored in the laboratory room and used exclusively for ME 570. Schematics and board layouts can be found in Appendix C.

5.3 Arduino Software

The Arduino software serves two purposes. The first, of course, is to control the Motorlab. The second is to illustrate to the more curious students just how typical hardware interface and user interface programming might work on a microcontroller. The entire code is contained inside a single file to make distribution to the students simple, though a more complicated file layout, including files for classes and includes, would be typical for an experienced programmer. The code is broken into relatively discrete blocks of code for readability, and will be described here along with the full code which can be found in Appendix B.

5.3.1 Encoding

Reading the motor encoder efficiently is a primary concern with the Arduino. Reading A and B channels of a quadrature encoder moving at higher speeds is typically done with pin change interrupts. Since the motor encoder is 1600 counts per revolution in quadrature, and the top speed of the motor is 5000RPM, the Arduino would have to process $\sim 133,333$ interrupts per second. Even if not used in quadrature, the Arduino would still need to process 33,333 interrupts per second. The solution is to first use hardware to change the quadrature signals into a single clock signal with a directional indicator. Then the Arduino uses a timer with the clock source set to the pin connected to the clock signal from the hardware. The timer chosen for this is TIMER5. It is initialized as a simple “up” counter with external clock. The direction is tracked using a simple pin change interrupt.

There are two places in the code where the timer is processed. First, in the control loop it is read and added to or subtracted from a global variable named EncTicks depending on direction. Finally, the timer is cleared. The other location is inside the interrupt service routine

for the pin change interrupt. Any time the motor changes direction, the interrupt routine fires and determines the direction. Then it updates the encoder count as in the control loop.

5.3.2 PWM

The amplifier requires a differential 1 amp/volt input between the REF+ and REF- inputs. This is handled by setting up another timer in the Arduino and configuring it for PWM output on two pins. One provides a variable duty cycle and the other is held at zero by setting the timer compare register to zero. When a change in current direction is required, the pins will swap. The current limit on the Motorlab is 3A, but the PWM signal on the output of the MUX circuit is 3.3V. Therefore, a maximum duty cycle is calculated to ensure that the current command does not exceed 3A.

5.3.3 Control Loop

The control loop is where most of the action is. It runs as fast as possible while still leaving some overhead for user interface processing. On the Arduino Mega 2560 R3 this is around 3.3kHz. The first action in the loop is to set a digital pin high, and the last action is to reset it low. This allows the measurement of the loop timing so that an optimum loop frequency can be found. The next action is to read and process the encoder and calculate speed and position.

The next bit of code is the “student” section. It is where the control law is processed, and includes error determination, proportional, integral and derivative calculation and integral windup control. This section is typically modified to include blanks or incomplete code segments so that students have the opportunity to code their controller.

5.3.4 Interface

The largest single part of the code is the user interface. The students are not expected to write the interface, although like most of the code, it is heavily commented so the more curious student would be able to modify it to suit their whims. All of the communications are processed via the Arduino's USB port and the virtual serial driver on the computer. The entirety of the interface is contained in a single function, `doGui()`.

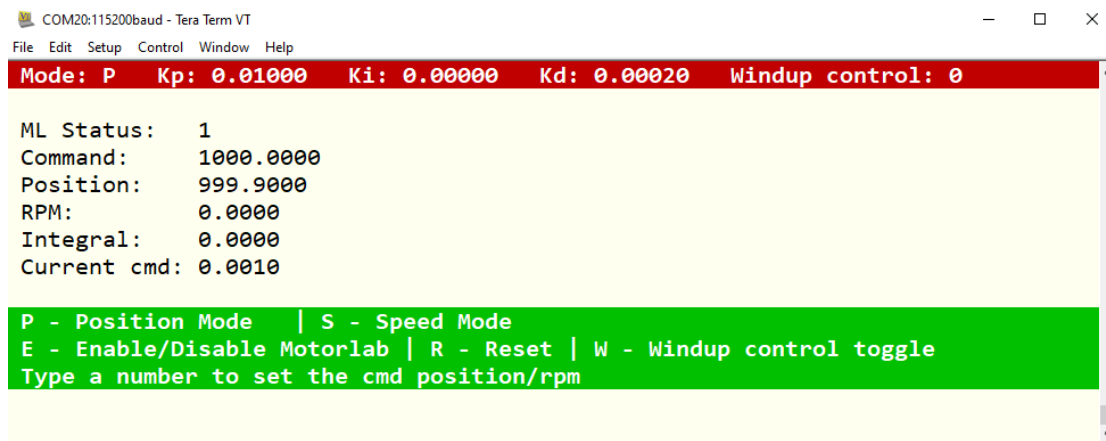


Figure 5.3 Arduino User Interface

It consists of an input portion where it checks to see if a key had been pressed or command (number) was entered, and an output portion to display information for the user. The writing section is sprinkled with escape characters and commands that are processed by any VT100 compatible terminal emulator.

5.4 Arduino Testing and Behavior

The Arduino system was then tested against results from the original system. The systems were run in position control mode with a proportional controller gain of 0.01. The motor was given a 200 degree step and the data was captured with the Motorlab GUI. The

resulting plots yield an interesting but predictable stretching of the decay time caused by the Arduino's slower control timing loop and lower resolution PWM generator.

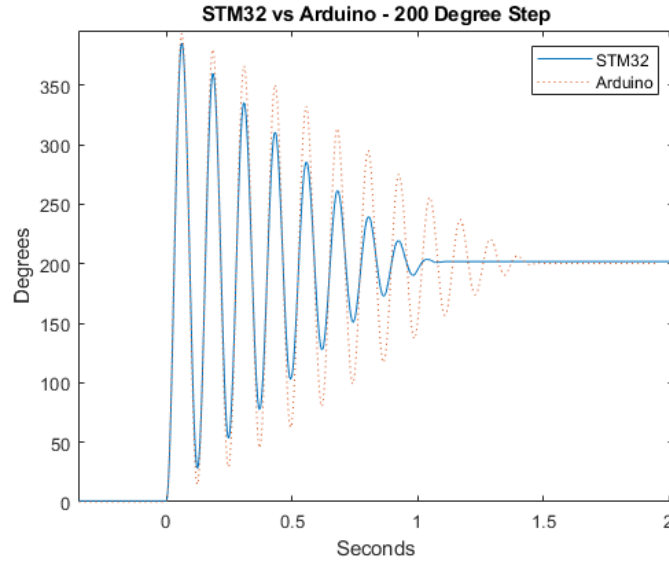


Figure 5.4 STM32 vs Arduino Results

A model was built in Simulink to attempt to match the Arduino results. The initial model consisted of a simulation that included both static and viscous friction which was used as an exercise to illustrate how Simulink works. The Arduino model built upon this model, adding a feedback loop time delay, a zero-order hold in the controller, and a quantizer to model the PWM resolution. Both the delay and zero-order hold parameters are set to the Arduino control loop time, 3ms. The quantization is determined with the following sets of equations:

$$\text{Max PWM} = 255 \frac{3V}{3.3V} = 232 \text{ states}$$

$$\text{Quantization Interval} = \frac{3V}{\text{Max PWM}} = 0.01294 \text{ volts/state}$$

The resulting model exports simulated data into the MATLAB workspace where it was processed.

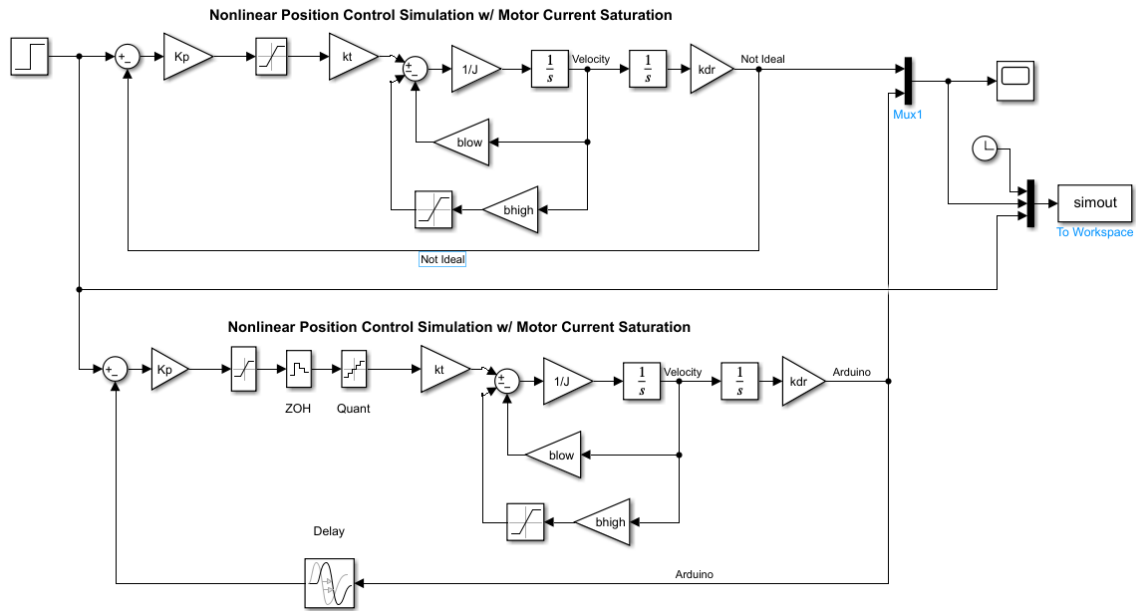


Figure 5.5 Simulink Model

The model results for both systems were then compared with the experimental data. The models and physical systems were in close agreement.

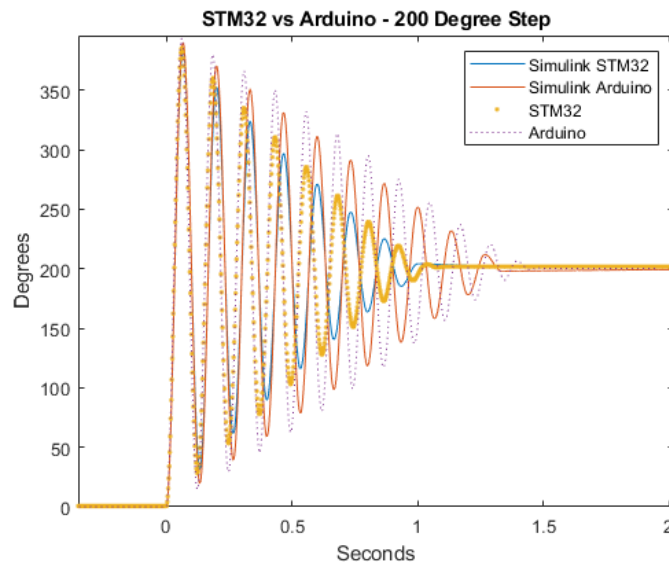


Figure 5.6 Physical and Simulated Results

These results hint at a potential for additional learning in more advanced controls classes, but the students in ME 570 are not adequately prepared to explain the difference. Therefore, any

lab exercise written for the Arduino will need to be crafted to avoid calling these differences to attention.

Chapter 6 - Arduino Laboratory Exercise

The entire purpose of this hardware and software retrofit is to help the students understand controllers better. Therefore, a new lab exercise was written to take advantage of the new system. This exercise was first introduced in the Spring 2022 semester.

6.1 Exercises

This lab exercise covers two major concepts that were not adequately covered in previous exercises. The first is programming the hardware. Prior to this exercise, everything was hidden behind the Motorlab GUI. The second is integral windup control. Both of these concepts did appear in exercises before as code snippets and hand-drawn inserts, but there was no way to experiment with them.

6.1.1 Coding

The first requirement is that the student download the starter code, make changes to it that result in a valid controller, and flash it to the Arduino. The starter code contains question marks as placeholders for the items the students must determine.

```
/*
 * Begin student code
 *
 * Okay - now we have a command term (cmd) and a physical term (actual)
 * Calculate the error first - command (cmd) minus (actual)
 * We've also previously defined Kp, Ki, Kd and the variables "derivative",
 * "last_err", and "integral"
 * And for the integral and derivative, we have the variable "dt" (time between
 * loop iterations)
 * Create a controller, and set the variable "out" with the above variables and constants
 * Error needs 1 line, derivative needs 2 lines, integral needs one (or two),
 * defining "out" needs one.
 * You code this block using the the following already defined variables
 * (don't set Kp, Ki, or Kd to a number - they are set elsewhere)
 *
 * Kp, Ki, Kd, actual, proportional, integral, derivative, dt, err and last_err
 */

err = cmd - ?????; // Error term - standard stuff you should know
proportional = ?? * err; // Proportional term
derivative = ?? * (err - ????????) / ??; // Derivative term
last_err = err;
integral = ??????? + ?? * err * dt; // Integral term
// Windup prevention - leave this line alone
if (do_windup) { integral = constrain(integral,-maxIntegral,maxIntegral); }
out = ????????? + ??????? + ?????????;
```

```
/*  
* End of student code  
*/
```

Once the student has programmed their Arduino, they are asked to connect to the Arduino with a terminal emulator that can process vt100 escape codes. They are then asked to enable the Motorlab, type in 1500 into the interface to set the position to 1500 degrees, then take a screen capture and record their observations.

6.1.2 Integral Windup Control

The default state of the Arduino interface is for windup control to be disabled. The students can see what windup control does in the code they modified before, but may not really grasp what it is. This exercise instructs the students to reset the Motorlab, set it to speed mode, and set the target speed to 9000RPM, which exceeds the maximum speed of the motor. They are then instructed to let the Motorlab to run for 5-10 seconds, causing the integral term to balloon. Then they set the speed to 900RPM and watch the speed of the motor stay constant until the integral term “drains”. They are asked to explain what happens to the integral term when the command is 9000RPM and what happens when the command was changed to 900RPM. Finally, they repeat the experiment with windup control on, and answer the same questions.

6.1.3 Narrative Quiz

In the Narrative Quiz, they are presented with the following paragraphs. Their answers, which were drop-down multiple choice or fill in the blank, are underlined.

When the GUI is in Position Control mode attempting to reach 1500 deg the motor reaches steady state quickly with 0 degrees steady-state error. As predicted for our motor in position control mode (think of the transfer function for our motor in position mode) which is type one.

With the GUI in Speed control mode and integral wind up control off, when you put in a desired speed of 9000RPM the integral term kept increasing. The motor was not able to reach 9000RPM. The motor saturated at approximately 3800 RPM.

Once we enter a speed command of 900RPM, the integral term has to decrease before the speed can reduce. After several long seconds (possibly minutes), the motor reaches a speed of approximately 900 RPM. Unlike position control, in speed control our motor (again think back to the transfer function for the motor in speed mode) is a type zero system. Due to this we observe some steady-state error when in speed control mode.

When we turn on windup control and put in 9000RPM, the integral term increases until it reaches its maximum value of 3. The motor saturates at a speed of approximately 3800 RPM.

When we change the desired speed to 900RPM, the motor reaches the desired speed faster than with windup control off.

6.2 Student Feedback

The students were given a survey following the Arduino lab exercise in the Fall 2022 Semester, asking if they liked the lab. The results were very positive, with the majority simply responding Yes, but many giving more feedback. Many enjoyed getting their “hands dirty” and felt that the concepts would be useful going forward. A few hit snags using the VSCode interface used in the department for Arduino programming. And only a couple had nothing

positive to say. Looking forward, the instructor for ME 570 does plan to survey the students again in the spring 2023 semester. A full listing of comments can be found in Appendix D.

Chapter 7 - Conclusion

The objective of this report was to provide insight into how the Motorlab is used in our introductory controls class, convey the desire of the students to have more insight into the hardware, and address student concerns with revisions to the current Motorlab. Highlights from several lab exercises illustrated the three basic modes of operation, open-loop, position and speed control used in class. A student survey using the TEVAL system was conducted in the spring of 2021 and the students clearly felt that being able to write, or at least read and understand, the controller code would help understand just what the Motorlab is doing. The hardware design process started almost immediately, prototyping solutions over winter break. A number of challenges arose with using the Arduino, a lesser microcontroller like discrete time delay terms and analog quantization. A simulation model parallel to that of the original system was created and tested and was proved accurate. This allows lab exercises can be written to avoid highlighting the deficiencies in the Arduino system. The Motorlabs were retrofitted during spring break, and a new lab exercise was written to take advantage of the system. The students received the new exercise well enough that the current instructor for ME 470 is having the students do a more involved project with the Arduino.

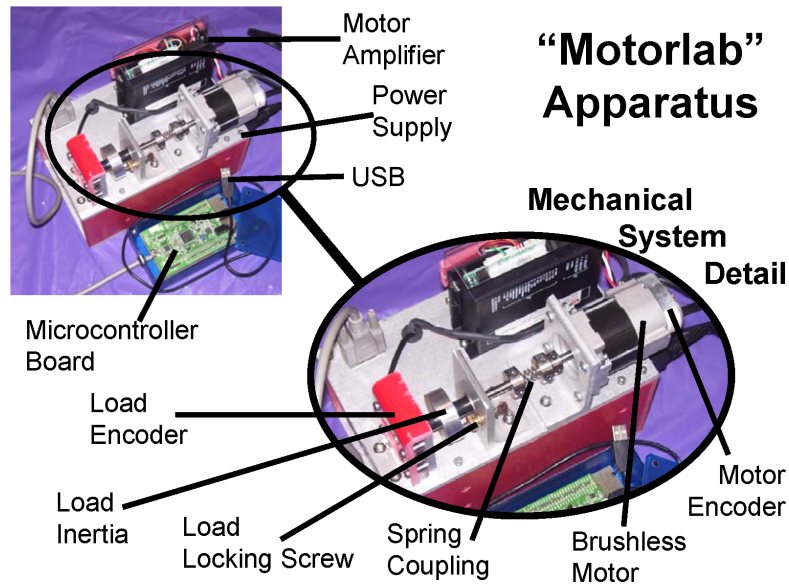
Currently, the students are familiar with the Arduino since they use it in ME 400. The Arduino will be replaced with Espressif ESP32-S3-DevkitC boards in the Fall of 2023. The first students who use the new microcontrollers should start appearing in ME 570 the following semester. Therefore, new shields will need to be created to replace the Arduino shields. The software will only require changes to hardware definitions. The lab exercise should not need modification.

References

- [1] J. Wieneke, D. Schinstock, W. N. White and G. Hu, "Redesign of an undergraduate controls laboratory with an eye toward accommodating future upgrades," Proceedings of the 2010 American Control Conference, Baltimore, MD, USA, 2010, pp. 384-389, doi: 10.1109/ACC.2010.5530834.
- [2] D. Schinstock, S. Schinstock and W. N. White, "Micro-controller based update of inexpensive undergraduate control systems laboratory hardware," 2015 American Control Conference (ACC), Chicago, IL, USA, 2015, pp. 2807-2812, doi: 10.1109/ACC.2015.7171160.
- [3] S. Smith, M. Professor, and D. Schinstock, "Demonstrating introductory control systems concepts on inexpensive hardware," 2017. Accessed: May 03, 2023. [Online]. Available: <https://krex.k-state.edu/bitstream/handle/2097/34627/ShaneSmith2017.pdf>
- [4] "STM32F405xx STM32F407xx." Accessed: Apr. 27, 2023. [Online]. Available: <https://www.st.com/resource/en/datasheet/stm32f407vg.pdf>

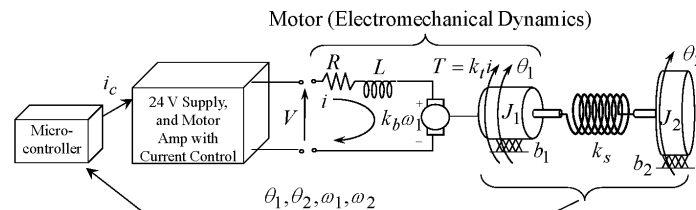
Appendix A – Motorlab Handout (Revised 2013)

“Motorlab” Dynamics and Controls System

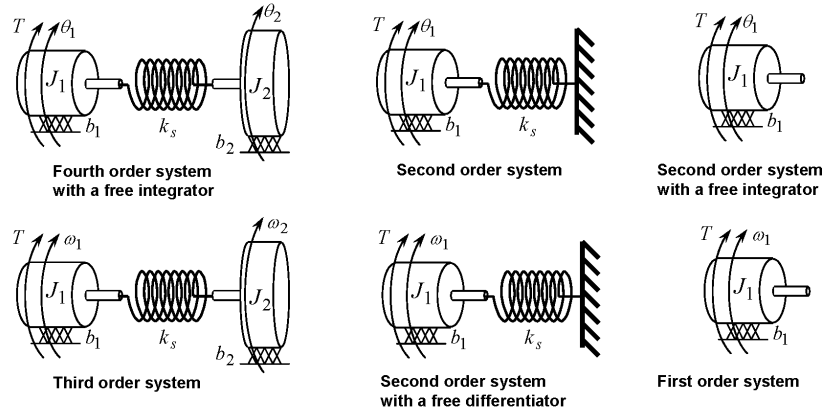


System Description

Below is a schematic representation of the Motorlab system in a closed-loop position or speed control configuration. There are two position sensors on the apparatus, a motor encoder and a load encoder. The speeds of the two inertias are measured by numerically differentiating the position signals in the computer controlling the system (microcontroller). The motor amplifier has a control loop that measures and controls the electric current in the motor windings. This results in what is commonly known as a “torque controlled” motor, since the magnetic torque is proportional to the current in the windings. The microcontroller is interfaced to the motor amplifier through a $\pm 10\text{V}$ analog signal. By varying the magnitude of this voltage the microcontroller can change the current in the motor. This voltage, which is proportional to the controlled current, serves as a current command (desired current) for the current control loop in the amplifier. An additional sensor, not shown below, is the current sensor in the amplifier used to implement the current control. The signal from this sensor is also read by the microcontroller, using an analog to digital converter. Although this signal is not used in the control loops on the microcontroller, it is recorded for data analysis.



Several different configurations of the system can be utilized in experiments. Either sensor, the motor or load encoder, can be used for the feedback of the control loop. The selection is made in the software interface. The motor encoder is known as a “collocated” sensor since it is co-located with the input to the mechanical system, the motor torque. The load sensor is separated from the input to the system by a spring and is therefore known as a “non-collocated” sensor. In addition to varying which sensor is used, the mechanical system can be changed with the lock down screw and the spring coupling. Also, a choice can be made between velocity control or position control by selecting the appropriate control program. Any of the following mechanical models may be realized using the Motorlab hardware and software.

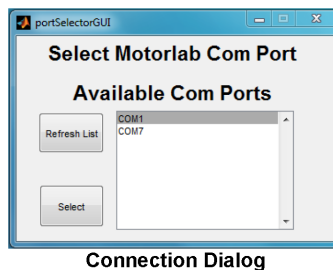


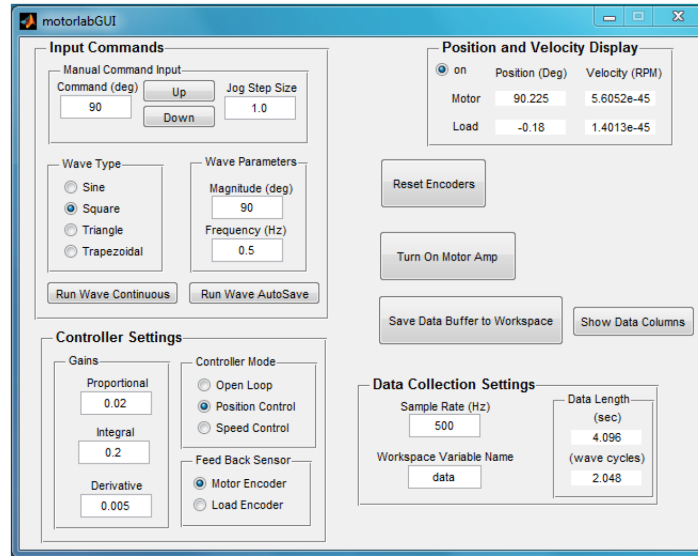
Software

The software for the system can be found in the “c:\Motorlab” directory on the laboratory machines. All the needed Matlab functions can be found there. The software that is on the microcontroller is included in this directory in the motorlabRepo.zip file. This program is burned into the flash memory of the microcontroller and runs on power up. The software that runs on the PC is a GUI written in Matlab (“motorlabGUI.m”). There are additional m-files in the “Motorlab” directory that can be used to plot data from the system.

User Interface

To run the Motorlab GUI you must open Matlab and add the “c:\Motorlab” directory to the Matlab path or set this directory as the current directory. Normally you will add it to the path and set the current directory to the location where you are storing your files. The microcontroller should be plugged into USB. In the Matlab command window type “motorlabGUI.” The opening dialog (below) asks you to select the communication port for the microcontroller. If more than one port is listed you should be able to detect which is the Motorlab by unplugging the USB or powering it down and then clicking the “Refresh List” button. The GUI should open after selecting the com port.





Motorlab GUI

Data Acquisition

The microcontroller stores data in a circular buffer that is 2048 data samples in length with 9 variables in each sample. After 2048 sample periods the buffer begins to be overwritten with the more recent data. At any time the buffer contains the most recent 2048 samples. Pressing the "Save Data Buffer to Workspace" button will write this data to a 2048x9 matrix in the Matlab workspace. Pressing the "Run Wave AutoSave" button starts the wave type selected and then writes the data to the Matlab workspace once the buffer has filled with new data. The time length of the data depends on the sample rate. If for example the sample rate is set to 500 Hz, then the last 4.096 seconds (2048/500) of data will be saved in the buffer.

The data matrix saved in the Matlab workspace contains 9 variables (columns). The ninth column is reserved. The other eight are listed below. Note that the variable in the second column changes. It depends on the "Controller Mode" chosen at the time of the data storage.

Column	1	2	3	4	5	6	7	8
Description	Time	Command	Motor Encoder	Load Encoder	Motor Speed	Load Speed	Current Command	Motor Current
Variable	t (sec)	θ_c (deg), ω_c (rpm), i_c (Amp)	θ_1 (deg)	θ_2 (deg)	ω_1 (rpm)	ω_2 (rpm)	i_c (Amp)	i (Amp)

M-files for plotting

There are m-files provided in the "c:\Motorlab" directory that can be used to plot the data from the Motorlab. Although you will frequently want the access the data with your own m-files, these files are useful for quickly viewing the data after acquiring it. There is one file for each of the "Controller Mode" settings.

File: mloplots.m function: mloplots(data,Iscale); Uses data generated by the Motorlab in open loop control. If an "Iscale" argument is supplied then the commanded current values are scaled by the Iscale value in the plots.

example: mloplots(data); Does not scale the current command.

example: mloplots(data,Iscale); Multiplies commanded current values by Iscale.

File: mlposplots.m function: mlposplots(data); Uses data generated by the Motorlab position control mode.
example: mlposplots(data);

File: mlspeedplots.m function: mlspeedplots(data); Uses data generated by the Motorlab velocity control mode.
example: mlvelplots(data);

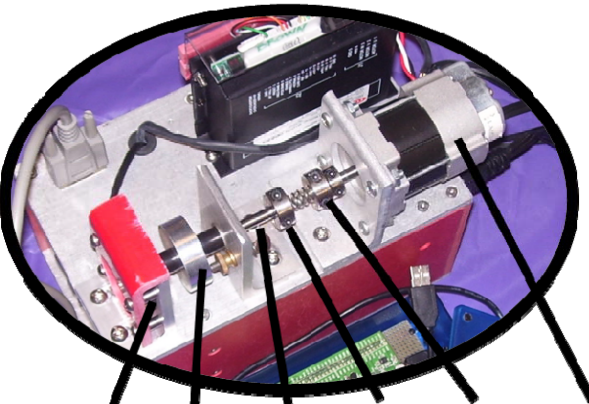
File: trapprof.m function: [x,v,t]=trapprof(DX,Vmax,Amax,DT) Trapezoidal-velocity motion profile generation
Outputs: x=position vector, v=trapezoidal velocity vector, t=time vector
Inputs: DX=distance to move, Vmax=maximum velocity, Amax=maximum acceleration, DT=time step for outputs
example: [x,v,t]=trapprof(DX,Vmax,Amax,DT)

Hardware Specifications

Important Scaling Considerations

- Motor Amplifier Scaling = 1 Amp/Volt. Therefore, one Volt output from the microcontroller corresponds to a one Amp command to the current control loop in the motor amplifier. The plotting routines provided take this scaling into consideration.
- Position is measured in degrees and velocity is measured in RPM. The output of the control algorithm in the microcontroller is measured in Volts. Therefore, for example, the units of the proportional and derivate gains in the position controller would be Volts/deg and Volts*sec/deg, respectively. When multiplied by the amplifier scaling (1 Amp/Volt) these gains become Amp/deg and Amp*sec/deg. The units of the proportional gain in the velocity controller would be Volts/RPM (or Amp/RPM if amplifier scaling is included).

Inertias



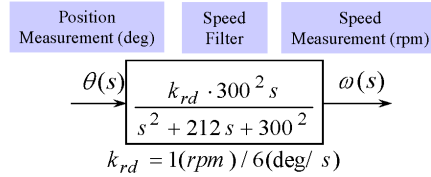
Object	Load Encoder	Load Wheel	Load Shaft	Single Shaft Collar	Double Shaft Collar	Motor Rotor and Motor Encoder
Inertia(g-cm ²)	0.8	32	1.4	15	19	110

A Few Other Details

- Max Data Acquisition Sample Rate = 10 kHz (the control update rate of the microcontroller software)
- Motor Encoder Resolution = 360 deg/1600 counts = 0.225 deg/count
- Load Encoder Resolution = 360 deg/2000 counts = 0.18 deg/count
- Max motor velocity with the 24 Volt power supply is about 4000 rpm

Speed Measurement

The two speeds measured by the Motorlab system are found using a discrete time approximation (i.e. computer code) of a derivative with a low pass filter. The continuous time transfer function for this filter is given below. It uses the encoder position measurement for input. Note the free s in the numerator performs the differentiation and the filter with a cutoff frequency of 300 rad/s helps to filter spikes in the speed measurement caused by differentiating the discrete steps inherent in an encoder position measurement.



Specs from Motor Manufacturer's Data Sheet

LA052-040E Motor Dynamic Specs From Shinano Kenshi		
	UNITS	Value
RATED POWER	W	40
RATED VOLTAGE	VDC	24
RATED SPEED	rpm	3,000
RATED TORQUE	N-cm	12.7
	kgf-cm	1.3
RATED CURRENT	A	2.5
TORQUE CONSTANT	N-cm/A	5.0
	kgf-cm/A	0.51
BACK EMF CONSTANT	V/krpm	5.2
PHASE RESISTANCE	Ohm	1.18
PHASE INDUCTANCE	mH	4.4
INSTANTANEOUS PEAK TORQUE	N-cm	38.2
MAX SPEED	rpm	5,000
ROTOR INERTIA	g-cm ²	110
POWER RATE	kW/s	1.48
MECHANICAL TIME CONSTANT	ms	5.2
ELECTRICAL TIME CONSTANT	ms	3.7
MASS	kg	0.6

Current Control Loop Model

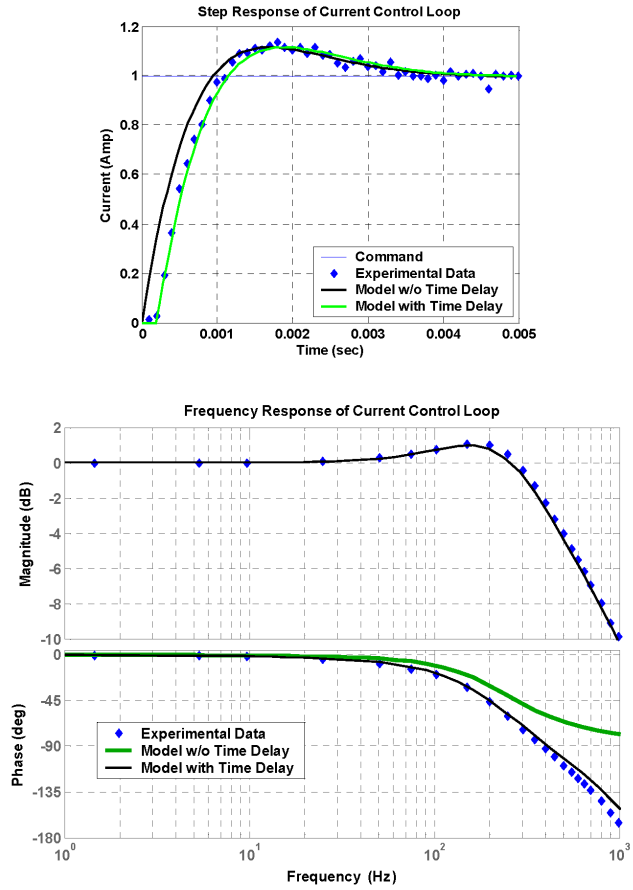
The motor amplifier has a current control loop. As configured in the Motorlab apparatus this loop has a bandwidth of approximately 400 Hz. Using data acquired from step and sinusoidal responses the following two closed loop transfer functions have been identified as approximate models for the closed-loop current control dynamics.

$$T_i = \frac{\omega_n^2(s+z)}{z(s^2 + 2\zeta\omega_n s + \omega_n^2)} \text{ or } \begin{cases} T_{i\text{delay}} = \frac{\omega_n^2(s+z)}{z(s^2 + 2\zeta\omega_n s + \omega_n^2)} e^{-t_d s} \\ T_{i\text{pade}} = \frac{\omega_n^2(s+z)}{z(s^2 + 2\zeta\omega_n s + \omega_n^2)} \cdot \frac{s^2 - 6s/t_d + 12/t_d^2}{s^2 + 6s/t_d + 12/t_d^2} \end{cases}$$

where :
 $z = 170 \cdot 2\pi \text{ (rad/sec)}$
 $\omega_n = 230 \cdot 2\pi \text{ (rad/sec)}$
 $\zeta = 0.8$
 $t_d = 0.0002 \text{ (sec)}$

Two of the models above contain a time delay while the other does not. One model with the time delay uses the exponential (exact) representation with the delay, while the other uses a second order Pade' approximation of the

delay. In the following two figures the responses of these two models are compared with actual data acquired from one of the Motorlab systems. Both the step response and the frequency response models are shown.



Motor Amplifier Manual



FEATURES

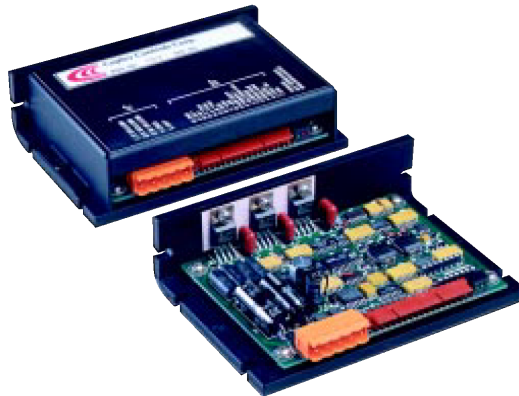
- *CE Compliance to 89/336/EEC*
- *Recognized Component to UL 508C*
- Complete torque (current) mode functional block
- Drives motor with 60° or 120° Halls
- Single supply voltage 18-55VDC
- 5A continuous, 10A peak more than double the power output of servo chip sets
- Fault protected
Short-circuits from output to output, output to ground
Over/under voltage
Over temperature
Self-reset or latch-off
- 2.5kHz bandwidth
- Wide load inductance range 0.2 to 40 mH.
- +5, +15V Hall power
- Separate continuous, peak, and peak-time current limits
- Surface mount technology

APPLICATIONS

- X-Y stages
- Robotics
- Automated assembly machinery
- Component insertion machines

THE OEM ADVANTAGE

- **NO POTS:** Internal component header configures amplifier for applications
- Conservative design for high MTBF
- Low cost solution for small brushless motors to 1/3 HP



PRODUCT DESCRIPTION

Model 503 is a complete pwm servoamplifier for applications using DC brushless motors in torque (current) mode. It provides six-step commutation of three-phase DC brushless motors using 60° or 120° Hall sensors on the motor, and provides a full complement of features for motor control. These include remote inhibit/enable, directional enable inputs for connection to limit switches, and protection for both motor and amplifier.

The /Enable input has selectable active level (+5V or gnd) to interface with most control cards.

/Pos and /Neg enable inputs use fail-safe (ground to enable) logic.

Power delivery is four-quadrant for bi-directional acceleration and deceleration of motors.

Model 503 features 500W peak power output in a compact package using surface mount technology.

An internal header socket holds components which configure the various gain and current limit settings to customize the 503 for different loads and applications.

Separate peak and continuous current limits allow high acceleration without sacrificing protection against continuous overloads. Peak current time limit is settable to match amplifier to motor thermal limits.

Header components permit compensation over a wide range of load inductances to maximize bandwidth with different motors.

Package design places all connectors along one edge for easy connection and adjustment while minimizing footprint inside enclosures.

High quality components and conservative ratings insure long service life and high reliability in industrial installations.

A differential amplifier buffers the reference voltage input to reject common-mode noise resulting from potential differences between controller and amplifier grounds.

Output short circuits and heatplate overtemperature cause the amplifier to latch into shutdown. Grounding the reset input will enable an auto-reset from such conditions when this feature is desired.

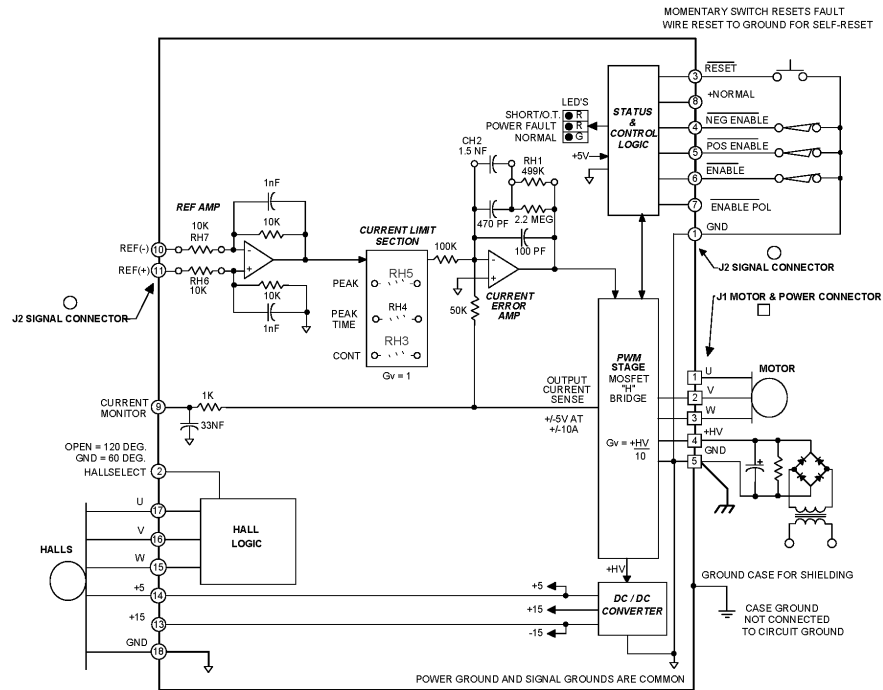


Corporate Offices: 410 University Avenue
Westwood, MA 02090

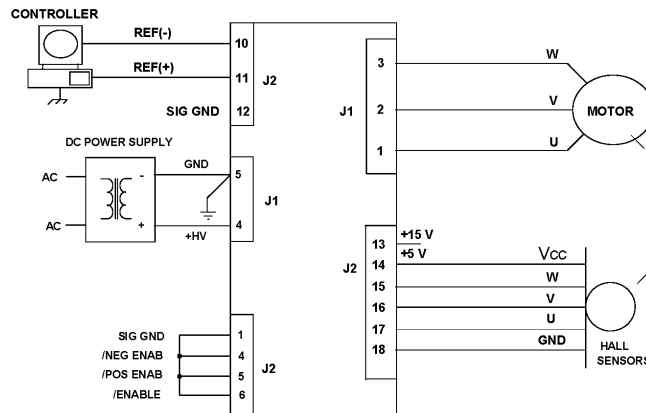
Telephone: (781) 329-8200
Fax: (781) 329-4055
E-mail: sales@copleycontrols.com
<http://www.copleycontrols.com>

Model 503
DC Brushless Servo Amplifier

FUNCTIONAL DIAGRAM



TYPICAL CONNECTIONS



APPLICATION INFORMATION

To use the model 503 set up the internal header with the components that configure the transconductance, current limits, and load inductance. Current-limits and load inductance set up the amplifier for your particular motor, and the transconductance defines the amplifiers overall response in amps/volt that is required by your system.

COMPONENT HEADER SETTINGS

Use the tables provided to select values for your load and system. We recommend that you use these values as starting points, adjusting them later based on tests of the amplifier in your application.

LOAD INDUCTANCE (RH1,CH2)

Maximizes the bandwidth with your motor and supply voltage. First replace CH2 with a jumper (short). Adjust the value of RH1 using a step of 1A or less so as not to experience large signal slew-rate limiting. Select RH1 for the best transient response (lowest risetime with minimal overshoot). Once RH1 has been set. choose the smallest value of CH2 that does not cause additional overshoot or degradation of the step response.

TRANSCONDUCTANCE (RH6,7)

The transconductance of the 503 is the ratio of output current to input voltage. It is equal to $10k\Omega/RH6$ (Amps/Volt). RH6, and RH7 should be the same value and should be 1% tolerance metal film type for good common-mode noise rejection.

CURRENT LIMITS (RH3, 4, & 5)

The amplifier operates at the 5A continuous, 10A peak limits as delivered. To reduce the limit settings, choose values from the tables as starting points, and test with your motor to determine final values. Limit action can be seen on current monitor when output current no longer changes in response to input signals. Separate control over peak, continuous, and peak time limits provides protection for motors, while permitting higher currents for acceleration.

SETUP BASICS

1. Set RH1 and CH2 for motor load inductance (see following section).
2. Set RH3, 4, & 5 if current limits below standard values is required.
3. Ground the /Enable (/Enable Pol open), /Pos Enable, and /Neg Enable inputs to signal ground.
4. Connect the motor Hall sensors to J2 based on the manufacturers suggested signal names. Note that different manufacturers may use A-B-C, R-S-T, or U-V-W to name their Halls. Use the required Hall supply voltage (+5 or +15V). *Note that there is a 30 mA limit at +5V. Encoders that put-out Hall signals typically consume 200-300 mA, so if these are used, then they must be powered from an external power supply.*
5. Connect J1-4,5 to a transformer-isolated source of DC power, +18-55V. Ground the amplifier and power supply with an additional wire from J1-5 to a central ground point.

6. With the motor windings disconnected, apply power and slowly rotate the motor shaft. Observe the Normal (green) led. If the lamp blinks while turning then the 60/120° setting is incorrect. If J2-2 is open, then ground it and repeat the test. In order to insure proper operation, the correct Hall phasing of 60° or 120° must be made.

6. Turn off the amplifier and connect the motor leads to J1-1,2,3 in U-V-W order. Power up the unit. Apply a sinusoidal reference signal of about 1 Hz. and 1Vrms between Ref(+) and Ref(-), J2-10,11.

7. Observe the operation of the motor as the current monitor signal passes through zero. When phasing is correct the speed will be smooth at zero crossing and at low speeds. If it is not, then power-down and re-connect the motor.

There are six possible ways to connect the motor windings, and only one of these will result in proper motor operation. The six combinations are listed in the table below. Incorrect phasing will result in erratic operation, and the motor may not rotate. When the correct combination is found, record your settings.

	J1-1	J1-2	J1-3
#1	U	V	W
#2	V	W	U
#3	W	U	V
#4	U	W	V
#5	W	V	U
#6	V	U	W

GROUNDING & POWER SUPPLIES

Power ground and signal ground are common (internally connected) in this amplifier. These grounds are isolated from the amplifier case which can then be grounded for best shielding while not affecting the power circuits.

Currents flowing in the power supply connections will create noise that can appear on the amplifier grounds.

This noise will be rejected by the differential amplifier at the reference input, but will appear at the digital inputs. While these are filtered, the lowest noise system will result when the power-supply capacitor is left floating, and each amplifier is grounded at its power ground terminal (J1-5). In multiple amplifier configurations, always use separate cables to each amplifier, twisting these together for lowest noise emission. Twisting motor leads will also reduce radiated noise from pwm outputs. If amplifiers are more than 1m. from power supply capacitor, use a small (500-1000µF) capacitor across power inputs for local bypassing.



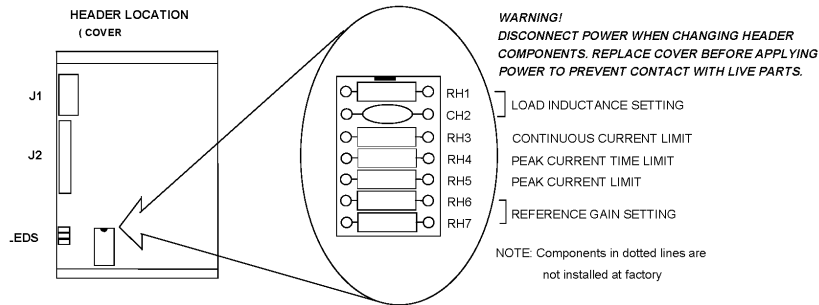
Corporate Offices: 410 University Avenue
Westwood, MA 02090

Telephone: (781) 329-8200
Fax: (781) 329-4055
E-mail: sales@copleycontrols.com
<http://www.copleycontrols.com>

Model 503
DC Brushless Servo Amplifier

APPLICATION INFORMATION (CONT'D)

COMPONENT HEADER



CONTINUOUS CURRENT LIMIT (RH3)

I _{cont} (A)	RH3 (Ω)
5	open *
4	20k
3	8.2k
2	3.9k
1	1.5k

INPUT TO OUTPUT GAIN SETTING (RH6, RH7)

Note 1

Example: Standard value of RH6 is 10kΩ, thus G = 1 A/V

PEAK CURRENT LIMIT (RH5) Note 3

I _{peak} (A)	RH5 (Ω)
10	open *
8	12k
6	4.7k
4	2k
2	750

LOAD INDUCTANCE SETTING (RH1 & CH2) Note 2

Load (mH)	RH1	CH2
0.2	49.9 k	1.5 nF
1	150 k	1.5 nF
3	499 k	1.5 nF *
10	499 k	3.3 nF
33	499 k	6.8 nF
40	499 k	10 nF

PEAK CURRENT TIME-LIMIT (RH4) Note 4

T _{peak} (s)	RH4 (Ω)
0.5	open *
0.4	10 M
0.2	3.3 M
0.1	1 M

Times shown are for 10A step from 0A

Notes: * **Standard values installed at factory are shown in italics.**

1. RH6 & RH7 should be 1% resistors of same value.
2. Bandwidth and values of RH1, CH2 are affected by supply voltage and load inductance. Final selection should be based on customer tests using actual motor at nominal supply voltage.
3. Peak current setting should always be greater than continuous current setting.
4. Peak times will double when current changes polarity. Peak times decrease as continuous current increases.

Model 503 DC Brushless Servo Amplifier

TECHNICAL SPECIFICATIONS

Typical specifications @ 25°C ambient, +HV = +55VDC. Load = 200μH. in series with 1 ohm unless otherwise specified.

OUTPUT POWER	
Peak power	
Unidirectional	±10A @ 50V for 0.5 second, 500W
After direction change	±10A @ 50V for 1 second, 500W
Continuous power	±5A @ 50V, 250W
OUTPUT VOLTAGE	
Vout = 0.97HV -(0.4)(Iout)	
MAXIMUM CONTINUOUS OUTPUT CURRENT	
Convection cooled, no conductive cooling	±2A @ 35°C ambient
Mounted on narrow edge, on steel plate, fan-cooled 400 ft/min	±5A @ 55°C
LOAD INDUCTANCE	
Selectable with components on header socket	200 μH to 40mH (Nominal, for higher inductances consult factory)
BANDWIDTH	
Small signal	-3dB @ 2.5kHz with 200μH load
Note: actual bandwidth will depend on supply voltage, load inductance, and header component selection	
PWM SWITCHING FREQUENCY	
25kHz	
ANALOG INPUT CHARACTERISTICS	
Reference	Differential, 20K between inputs with standard header values
GAINS	
Input differential amplifier	X1 as delivered. Adjustable via header components RH6, RH7
PWM transconductance stage	1 A/V (output vs. input to current limit stage)
OFFSET	
Output offset current (0 V at inputs)	20 mA max. (0.2% of full-scale)
Input offset voltage	20 mV max (for 0 output current, RH6,7 = 10kΩ)
LOGIC INPUTS	
Logic threshold voltage	HI: ≥ 2.5V , LO: ≤ 1.0V, +5V Max on all logic inputs
/Enable	LO enables amplifier (/Enable Pol open) , HI inhibits: 50 ms turn-on delay
/POS enable, /NEG enable	LO enables positive and negative output currents, HI inhibits
/Reset	LO resets latching fault condition, ground for self-reset every 50 ms.
/Enable Pol (Enable Polarity)	LO reverses logic of /Enable input only (HI enables unit, LO inhibits)
LOGIC OUTPUTS	
+Normal	HI when unit operating normally, LO if overtemp, output short, disabled, or power supply (+HV) out of tolerance
	HI output voltage = 2.4V min at -3.2 mA max., LO output voltage = 0.5V max at 2 mA max.
	Note: Do not connect +Normal output to devices that operate > +5V
INDICATORS (LED's)	
Normal (green)	ON = Amplifier enabled, no shorts or overtemp, power within limits
Power fault (red)	ON = Power fault: +HV <18V OR +HV > 55V
Short/Overtemp (red)	ON = Output short-circuit or over-temperature condition
CURRENT MONITOR OUTPUT	
±5V @ ±10A (2A/volt), 10kΩ, 3.3nF R-C filter	
DC POWER OUTPUTS	
+5VDC	30mA (includes power for Hall sensors)
+15VDC	10mA
Total power from all outputs not to exceed 200mW.	
PROTECTION	
Output short circuit (output to output, output to ground)	Latches unit OFF (self-reset if /RESET input grounded)
Overtemperature	Shutdown at 70°C on heatplate (Latches unit OFF)
Power supply voltage too low (Undervoltage)	Shutdown at +HV < 18VDC (operation resumes when power >18VDC)
Power supply voltage too high (Overvoltage)	Shutdown at +HV > 55VDC (operation resumes when power <55VDC)
POWER REQUIREMENTS	
DC power (+HV)	+18-55 VDC @ 10A peak.
Minimum power consumption	2.5 W
Power dissipation at 5A output, 55VDC supply	10W
Power dissipation at 10A output, 55VDC supply	40W
THERMAL REQUIREMENTS	
Storage temperature range	-30 to +85°C
Operating temperature range	0 to 70°C baseplate temperature
MECHANICAL	
Size	3.27 x 4.75 x 1.28 in. (83 x 121 x 33mm)
Weight	0.52 lb (0.24 kg.)
CONNECTORS	
Power & motor	Weidmuller: BL-125946; Phoenix: MSTB 2.5/5-ST-5.08
Signal & Halls	Molex: 22-01-3167 housing with 08-50-0114 pins



Corporate Offices: 410 University Avenue
Westwood, MA 02090

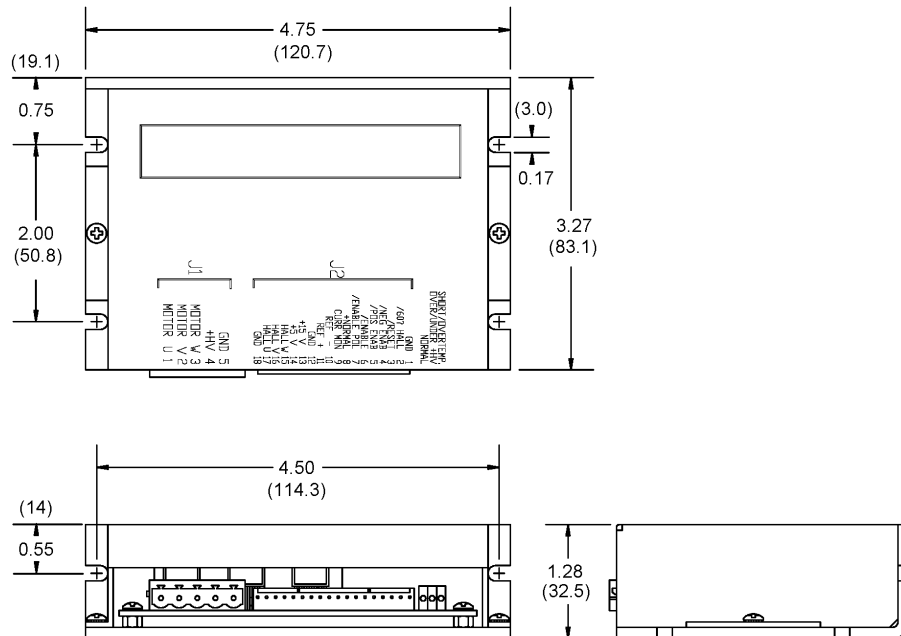
Telephone: (781) 329-8200
Fax: (781) 329-4055
E-mail: sales@copleycontrols.com
http://www.copleycontrols.com

225

Model 503 DC Brushless Servo Amplifier

OUTLINE DIMENSIONS

Dimensions in inches (mm.)



ORDERING GUIDE

Model 503	5A Continuous, 10A Peak, +18-55VDC Brushless Servoamplifier
-----------	---

OTHER BRUSHLESS AMPLIFIERS

- Model 505** Same power output as 503. Adds Hall / Encoder tachometer feature for velocity loop operation.
- 5001 Series** Six models covering +24-225VDC operation, 5-15A continuous, 10-30A peak. With optional Hall / Encoder tachometer, and brushless tachometer features.
- Model 513R** Resolver interface for trapezoidal-drivemotors. Outputs A/B quadrature encoder signals and analog tachometer signal for velocity loop operation. +24-180VDC operation, 13A continuous, 26A peak.

***Excerpt of Data Sheet for the
STM32F4 Microcontroller***



UM1472 User Manual

STM32F4DISCOVERY STM32F4 high-performance discovery board

Introduction

The STM32F4DISCOVERY helps you to discover the STM32F4 high-performance features and to develop your applications. It is based on an STM32F407VGT6 and includes an ST-LINK/V2 embedded debug tool interface, ST MEMS digital accelerometer, ST MEMS digital microphone, audio DAC with integrated class D speaker driver, LEDs, pushbuttons and an USB OTG micro-AB connector.

Figure 1. STM32F4DISCOVERY

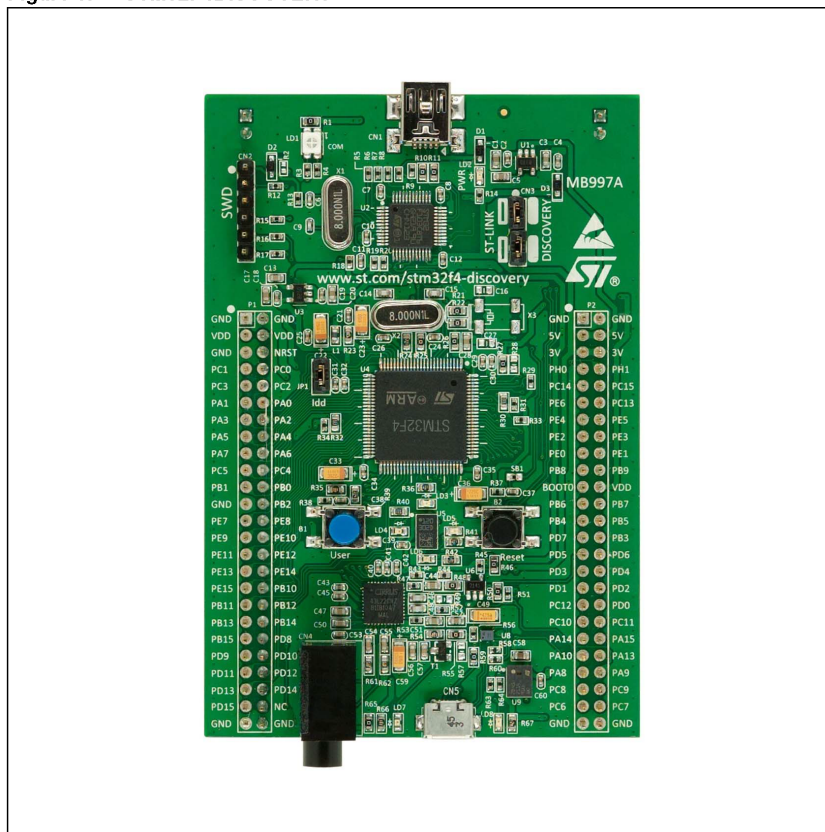
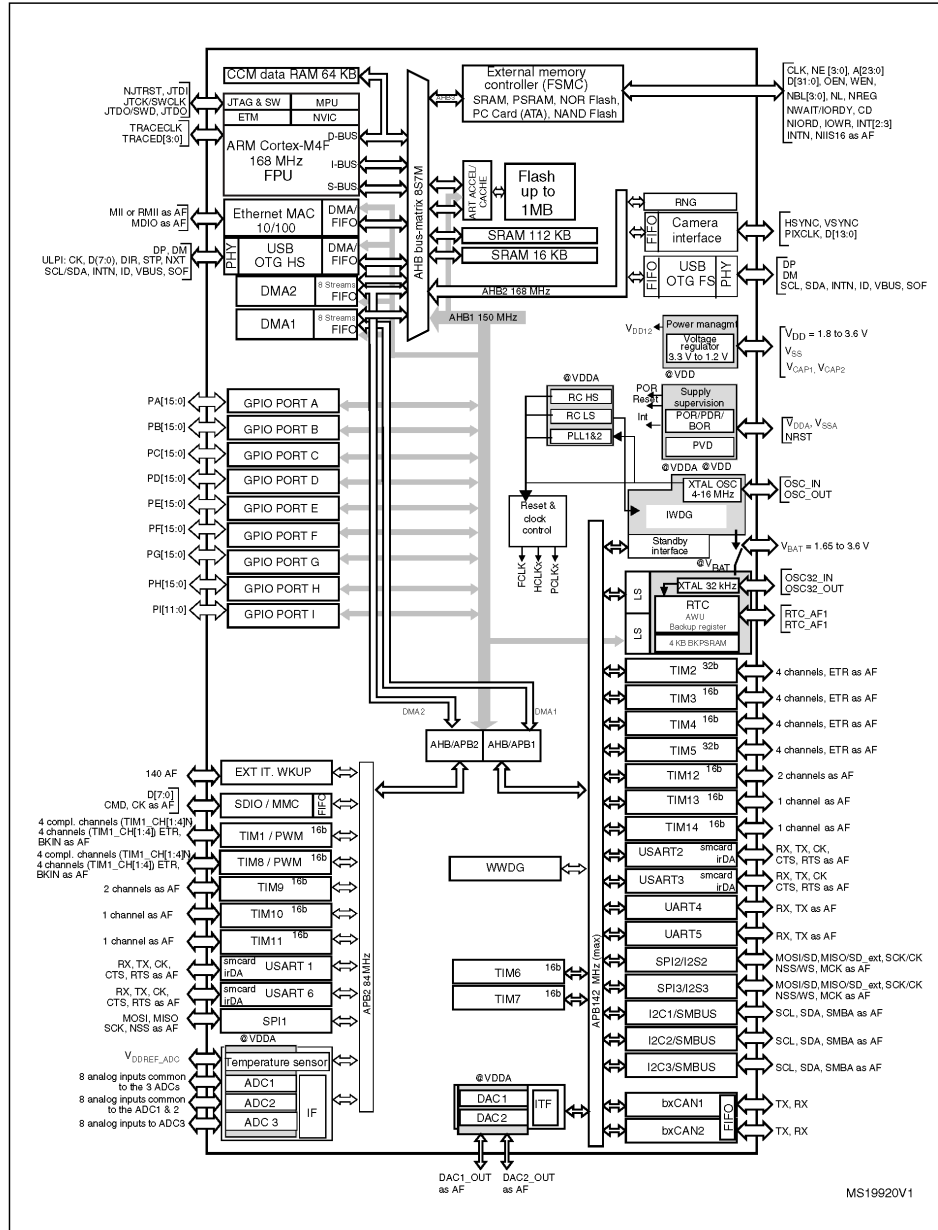


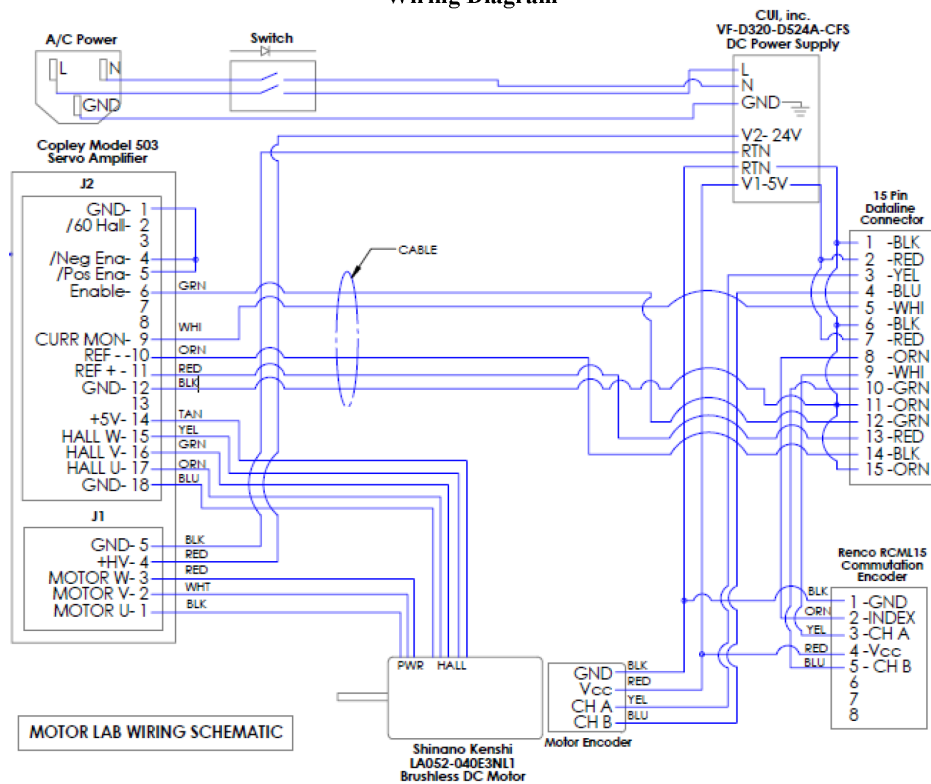
Figure 6. STM32F407VGT6 block diagram



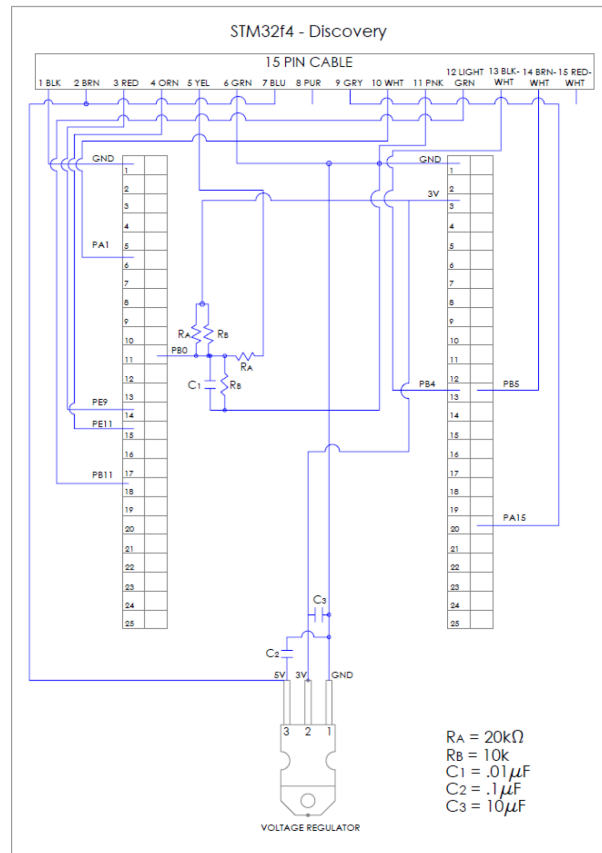
Wiring for the “Motorlab” Apparatus

15 Pin Microcontroller Connections					
Amp J2	Hardware Wire Color	15pin Connector	Function	15pin Cable	STM32F4 Discovery
	Black	1 Black	Motor Encoder Ground	1 Black	GND
	Red	2 Red	+5V Power	2 Brown	Voltage Regulator
	Yellow	3 Yellow	Motor Encoder Channel A	3 Red	PE9 (TIM1-Ch1)
	Blue	4 Blue	Motor Encoder Channel B	4 Orange	PE11 (TIM1-Ch2)
Curr Mon	White	5 White	Current Monitor	5 Yellow	PB0 (ADC1-Ch8) thru resistor network
	Black	6 Black	Inertia Encoder Ground	6 Green	GND
	Red	7 Red	+5V Power	7 Blue	Voltage Regulator
	Orange	8 Orange	Inertia Encoder Index	8 Purple	--
	Yellow	9 White	Inertia Encoder Channel A	9 Gray	PA15 (TIM2-Ch1)
	Blue	10 Green	Inertia Encoder Channel B	10 White	PA1 (TIM2-Ch2)
GND	12 Black	11 Orange	Amp Signal Ground	11 Pink	GND
Enable	6 Green	12 Green	Amp Signal (Amp Enable)	12 Light Green	PB11 (Digital Out)
Ref+	11 Red	13 Red	Current Command +	13 Black-White	PB4 (TIM3-Ch1)
Ref-	10 Orange	14 Black	Current Command -	14 Brown-White	PB5 (TIM3-Ch2)
GND	12 Black	15 Orange	GND	15 Red-White	--

Wiring Diagram



STM32F4Discovery Host Board



Appendix B – Arduino Code

```

/*
 * Description :   KSU ME570 Arduino / Motorlab code
 * Author      :   Eric Patterson
 * Date       :   03/23/2022
 * License    :   MIT
 * This code is published as open source software. Feel free to share/modify.
 *
 * Students - see code starting line 176
 */
#include <Arduino.h>

/*
 * Defines
 */
#define Kp_p 0.1           // Position Kp, Ki and Kd
#define Ki_p 0.0
#define Kd_p 0.001

#define Kp_s 0.002         // Speed Kp, Ki and Kd
#define Ki_s 0.02
#define Kd_s 0.0

#define Ref_nPin 12        // Current control positive -> amplifier
#define Ref_pPin 11        // Current control negative -> amplifier
#define Ref_nOCR OCR1B     // Output Compare Register Timer 1 B (pin 12)
#define Ref_pOCR OCR1A     // Output Compare Register Timer 1 A (pin 11)
#define enc_ckPin 47        // Count combined encoder pulses on pin 47 / timer5
#define enc_DirPin 51      // Counter direction on pin 51 - pin change ISR
#define m1EnablePin 43     // "Turn On" the motorlab
#define sinPin 22          // command sine wave if enabled
#define sinPinPos 23       // Set to 5v all the time
#define ledPin 32          // Keep track of how long code takes (debugging)
#define ticks_to_deg 0.225 // Convert encoder ticks to degrees - 1600cts/rev
#define ctrPWM 255         // PWM counter counts to this number, then reset
#define ctrLP 74           // Sets timed loop frequency - 16E6/(64*(ctrLP+1)) =
                           // 3.333kHz
#define maxIntegral 30     // Somewhat arbitrary maximum value for integral term
/*
 * Global Variables
 */
bool redraw = true;        // Serial output - redraw first time
float integral = 0;        // Add err*dt each loop
bool UpDown = 0;          // Which way is the counter going?
float last_err = 0;        // What was the error last time (derivative calc)
float last_pos = 0;        // What was the position last time (velocity calc)
float pos = 0;             // Position in degrees - global for debug
float rpm = 0;             // RPM - global for debug
float err = 0;             // Error term
float out = 0;             // Output of the controller - global for debug
boolean toggle = 0;        // used to bookend the timed loop with pulses
float Kp, Ki, Kd;          // Which Kp, Ki and Kd are we using
char mode = 'P';           // Position = P, Speed = S
int maxPWM = ctrPWM * 3.0 / 3.3; // Keep pwm limited to 3v (amplifier amps) 512*(3/3.3)
float dt = (64 * (ctrLP+1)) / 16E6; // Time between loops - /64 prescalar, count to ctrLP+1,
// all over 16MHz, then repeat.
boolean do_windup = 0;     // Enable/Disable integral windup
bool reset = true;        // Clear on first run and when asked to

/*
 * Volatile Global Variables - modifiable in multiple loops/functions
 */
volatile long EncTicks = 0;
volatile int ctr = 0;

```

```

volatile float cmd = 0;                // Command position/rpm - global for debug

/*
 * Do initial bootup stuff
 */
void setup() {
    cli();                            // Disable interrupts
    digitalWrite(mlEnablePin,1);       // Turn off motorlab amp (active low)

    Serial.begin(115200);              // Use serial port for human communication
    pinMode(sinPin,INPUT);             // Activate sine mode when pulled high by jumpering neighboring
pin sinPinPos
    digitalWrite(sinPin,1);           // Set internal pullup on so it'll have to go low to change
state
    pinMode(sinPinPos,OUTPUT);         // Always low so we can have something to pull sinPin down
    digitalWrite(sinPinPos,0);
    pinMode(enc_DirPin,INPUT);         // Direction pin (from LS7084)
    pinMode(Ref_pPin,OUTPUT);          // positive PWM
    pinMode(Ref_nPin,OUTPUT);          // negative PWM
    pinMode(ledPin,OUTPUT);            // led pin - debugging
    pinMode(mlEnablePin,OUTPUT);       // motorlab enable (active low)

    /*
     * Port B is where the enc_DirPin lives.
     * We use it specifically because on the Mega, it is one of the few that does pin change
interrupts
     * Setup ISR for it (see datasheet)
     */
    DDRB = 0;
    PORTB |= B00000100; // Input on portb2 (pin 52)
    PCMSK0 |= B00000100; // Interrupt on pin change (PCINT2) - this includes pin 52
    PCICR |= B00000001; // Interrupts for pins PCINT23:16

    /*
     * Use Timer 1 to do PWM on Ref_pPin and Ref_nPin
     * PWM frequency needs to be as fast as possible
     * See datasheet for parameter meanings
     */
    DDRB |= (1 << DDB5) | (1 << DDB6); // Pin direction for 11,12
    PORTB |= (1 << DDB5) | (1 << DDB6);
    TCCR1A = 0;
    TCCR1A = (1 << COM1A1) | (1 << COM1B1) | (1 << WGM11); // Fast PWM w/ ICR top
    TCCR1B = (1 << CS10) | (1 << WGM12) | (1 << WGM13); // No Prescaler
    ICR1 = ctrPWM; // Determines frequency

    /*
     * The Timer 2 ISR is the timed loop. It's where the control law lives
     * Set it up for as fast as we can without getting behind ~3kHz
     */
    TCCR2A = 0; // set entire TCCR2A register to 0
    TCCR2B = 0; // same for TCCR2B
    TCNT2 = 0; // initialize counter value to 0
    OCR2A = ctrLP; // Output compare register - F = (16E6) / (ctrLP*64) - 1
    TCCR2A |= (1 << WGM21);
    TCCR2B |= (1 << CS22); // set prescaler to 64
    TIMSK2 |= (1 << OCIE2A); // reset to zero when OCR2A is reached

    /*
     * Setup Timer 5 for pulse train - read processed encoder pulses
     * Here we set the counter to external clock (pin 47) so the cpu isn't wasting a bunch of time
with interrupts
     */
    TCCR5A = 0; // Set normal count mode
    TCCR5B = 0 | (1 << ICNC5)
                | (1 << CS52)
                | (1 << CS51); // Falling edge external clock mode, noise cancelling (see
datasheet)
    TCNT5 = 0; // Reset clock

    /*
     * Assume Position mode to begin with

```

```

*/
Kp = Kp_p;
Ki = Ki_p;
Kd = Kd_p;

/*
* Tidy up and get ready to run
*/
UpDown = (PINB >> 2) & 1;      // Get initial direction

sei();                          // Enable interrupts
}

/*
* "Main" loop - where we interact with the human
*/
void loop() {
    /* If the sinPin is pulled low, go into a demo mode where the motor position moves in a
    sinusoid */
    if (!digitalRead(sinPin)) {
        digitalWrite(m1EnablePin,0);
        cmd = 180.0 * sin(millis()/200.0); }
    else {
        doGui();                // Gui is in a function at the end of the code
        delay(100);             // Do display updates every 1/10th second
    }
}

/*
* Timed loop - timer2, 2kHz
* This is where all the action is!
*/
ISR(TIMER2_COMPA_vect) {
    float derivative;
    float actual;

    digitalWrite(ledPin,1);      // Set ledPin high so we can see start of loop on scope
    if (reset) { doReset(); }    // Set everything to default state - may have to reset more
    than once if motor is coasting.

    ctr = TCNT5-1;               // Read current number of encoder clock pulses
    TCNT5 = 1;                   // Set counter back to 1 - we risk loosing counts if we do
    this on an edge, but oh well ;)
    EncTicks += UpDown? -ctr : +ctr; // If up, add last batch of ticks, subtract if not
    pos = EncTicks*ticks_to_deg;  // Convert ticks to current position
    rpm = ((pos-last_pos) /dt)/6; // Calculate current RPM - if we used a better uP we'd
    implement a filter
    last_pos = pos;              // Keep track of last position so we can do dPosition/dTime
    to get RPM

    if (mode == 'P') { actual = pos; } // If in position mode, set actual to position
    if (mode == 'S') { actual = rpm; } // if in speed mode, set actual to speed

    /*
    * Begin student code
    *
    * Okay - now we have a command term (cmd) and a physical term (actual)
    * Calculate the error first - command (cmd) minus (actual)
    * We've also defined Kp, Ki, Kd and the variables "derivative", "last_err", and "integral"
    * And for the integral and derivative, we have the variable "dt" (time between loop iterations)
    * Create a controller, and set the variable "out" with the above variables and constants
    * Error needs 1 line, derivative needs 2 lines, integral needs one (or two), defining "out"
    needs one.
    * You code this block
    */

    err = cmd - actual;          // Error term - standard stuff you should know
    derivative = Kd * (err - last_err) / dt;
    last_err = err;
    integral = integral + Ki * err * dt;

```

```

    if (do_windup) { integral = constrain(integral,-maxIntegral,maxIntegral); }           // Windup
prevention
    out = err*Kp + integral + derivative;

    /*
    * End of student code
    */
    setPWM(out);           // Change the output PWM to the commanded current
    digitalWrite(ledPin,0); // We're done. Lower the pin so we can see how long our code took
(on the scope)
}

/*
* Interrupt handler so the hardware can tell us when the encoder changes direction
*/
ISR(PCINT0_vect){
    cli();
    ctr = TCNT5-1;
    UpDown = (PINB >> 2) & 1;           // Change the direction variable (we read this with direct
port manipulation for speed)
    EncTicks += UpDown? +ctr : -ctr; // Add the ticks in ctr to the EncTicks variable in the old
direction
    TCNT5 = 1;           // The pulse "right now" was in the new direction, so we'll
leave it on the counter
    sei();
}

/*
* Function to set the PWM duty cycle. It's compact to hopefully preserve cpu cycles
*/
void setPWM(float current) {
    unsigned int dc;
    dc = maxPWM * constrain(abs(current),0.0,3.0)/3.0; // Don't exceed 3 amps
    if (current >= 0) { Ref_pOCR = dc; Ref_nOCR = 0; } // Forward
    if (current < 0) { Ref_nOCR = dc; Ref_pOCR = 0; } // Reverse
}

/*
* Set nearly everything to zero. May have to do it more than once if the motor is coasting
*/
void doReset() {
    ctr = 0;
    TCNT5 = 1;
    integral = 0;
    last_err = 0;
    err = 0;
    cmd = 0;
    pos = 0;
    last_pos = 0;
    EncTicks = 0;
    rpm = 0;
    reset = 0;
}

/*
* Stick GUI down here in a function to keep main loop clean
*/
void doGui() {
    String str;
    char s;
    bool enable = !digitalRead(m1EnablePin); // The amplifier
is eanabled on a low state

    if (Serial.available()) { // You must have
typed something into the terminal!
        str = Serial.readString(); // read stuff
from com port
        s = str.charAt(0); // first
character in serial buffer
        if (isDigit(s)) { cmd = str.toFloat(); } // It's a number
- use as command

```

```

        else if (s == '-') { str.setCharAt(0,'0'); cmd = str.toFloat() * -1.0; } // It's a
negative number - use as command after messing with it
        else if (s == 'e' || s == 'E') { digitalWrite(mlEnablePin,enable); } // e =
enable/disable motorlab
        else if (s == 'p' || s == 'P') { mode = 'P'; } // p = position
mode
        else if (s == 's' || s == 'S') { mode = 'S'; } // s = speed mode
        else if (s == 'w' || s == 'W') { do_windup = !do_windup; } // w = windup
toggle
        else if (s == 'r' || s == 'R') { reset = true; } // r = reset
        redraw = true; // Clear screen
when we make touch a key
    }

/*
 * Swap in position or speed Kp, Ki and Kd terms depending on mode. Keeping this here to speed
up timed loop.
 */
if (mode == 'P') { Kp = Kp_p; Ki = Ki_p; Kd = Kd_p; }
if (mode == 'S') { Kp = Kp_s; Ki = Ki_s; Kd = Kd_s; }

/* Make a pretty GUI
 * VT100 codes - this will look like gibberish on non-vt100 terminals
 * \x1b (escape character)
 * [H - Home (top left)
 * [K - clear to end of line
 * [2K - clear whole line
 * [J - clear to end of screen
 * [row;columnf - set position on screen (from home, top left)
 * [attr1;attr2;attr3m - set font attributes, foreground/background
 * Google VT100 codes for a lot more
 */
if (redraw) {
    Serial.print("\x1b[?25l");
    Serial.print("\x1b[H"); // Go home
    Serial.print("\x1b[1;37;41m\x1b[2K"); // Reverse video for top line & clear to end of line
    Serial.print(" Mode: "); Serial.print(mode);
    Serial.print(" Kp: "); Serial.print(Kp, 5);
    Serial.print(" Ki: "); Serial.print(Ki, 5);
    Serial.print(" Kd: "); Serial.print(Kd, 5);
    Serial.print(" Windup control: "); Serial.print(do_windup);
    Serial.println("\x1b[m");
    Serial.println("\x1b[2K");
    Serial.println("\x1b[2K ML Status: ");
    Serial.println("\x1b[2K Command: ");
    Serial.println("\x1b[2K Position: ");
    Serial.println("\x1b[2K RPM: ");
    Serial.println("\x1b[2K Integral: ");
    Serial.println("\x1b[2K Current cmd: ");
    Serial.println("\x1b[2K");
}

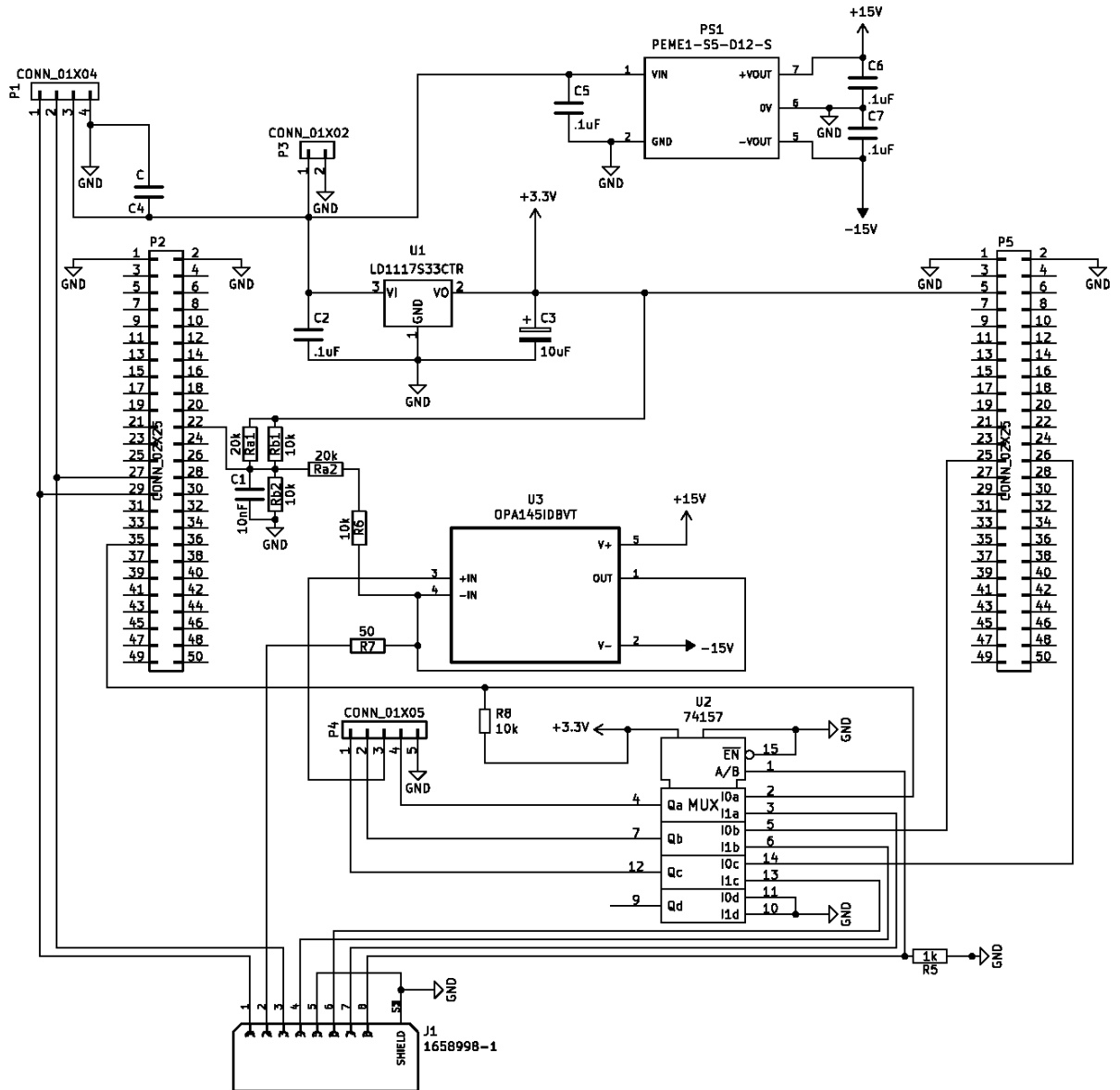
Serial.print("\x1b[3;15f"); Serial.print(enable); Serial.print("\x1b[K");
Serial.print("\x1b[4;15f"); Serial.print(cmd,4); Serial.print("\x1b[K");
Serial.print("\x1b[5;15f"); Serial.print(pos,4); Serial.print("\x1b[K");
Serial.print("\x1b[6;15f"); Serial.print(rpm,4); Serial.print("\x1b[K");
Serial.print("\x1b[7;15f"); Serial.print(integral,4); Serial.print("\x1b[K");
Serial.print("\x1b[8;15f"); Serial.print(out,4); Serial.print("\x1b[K");
Serial.println("");
Serial.println("");

if (redraw) {
    Serial.print("\x1b[1;37;42m\x1b[2K"); // Reverse video for this one too
    Serial.println("\x1b[2K P - Position Mode | S - Speed Mode");
    Serial.println("\x1b[2K E - Enable/Disable Motorlab | R - Reset | W - Windup control
toggle");
    Serial.println("\x1b[2K Type a number to set the cmd position/rpm \x1b[m");
    Serial.println("\x1b[J");
    redraw = 0;
}
}

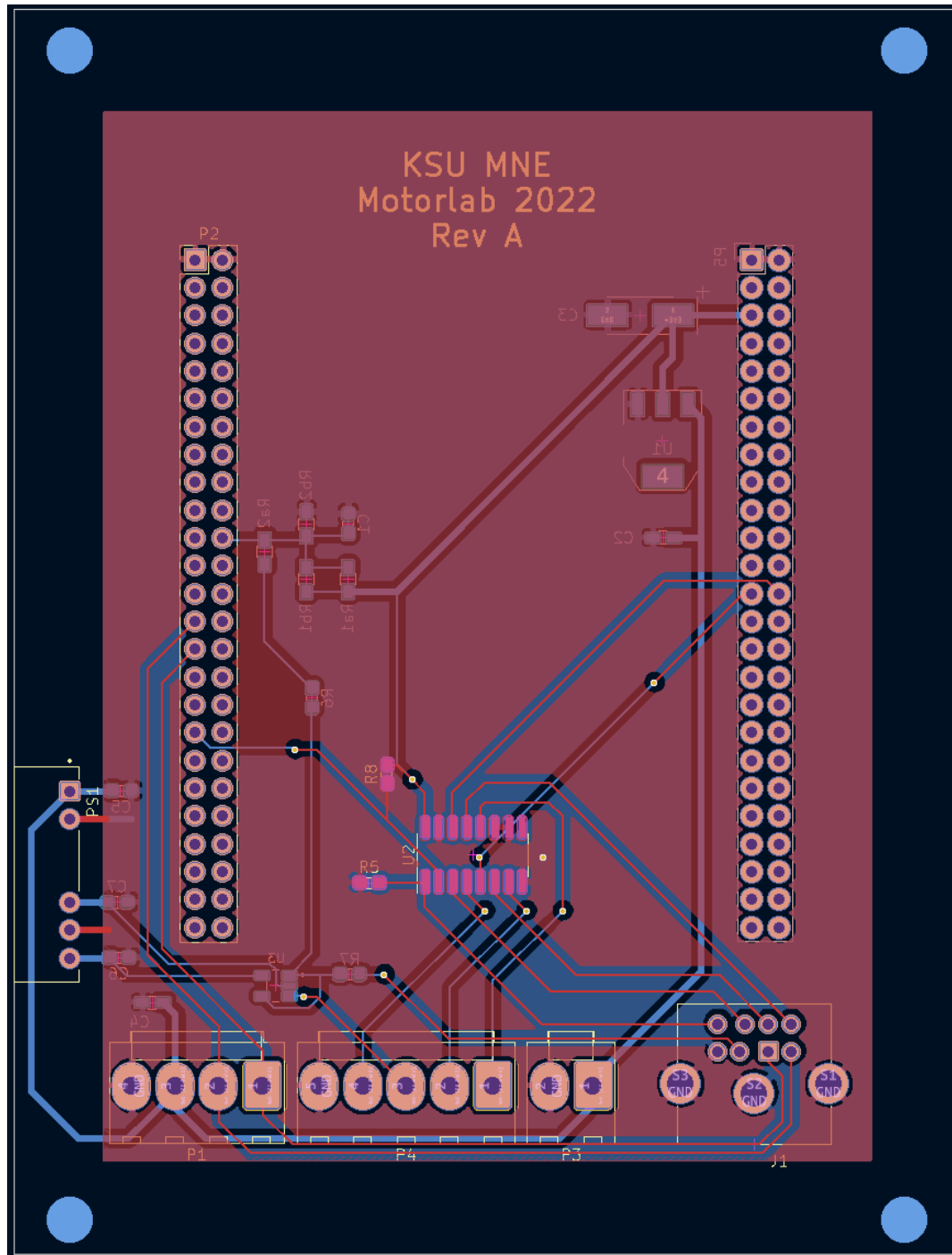
```

Appendix C – Revised Schematics and Board Layouts

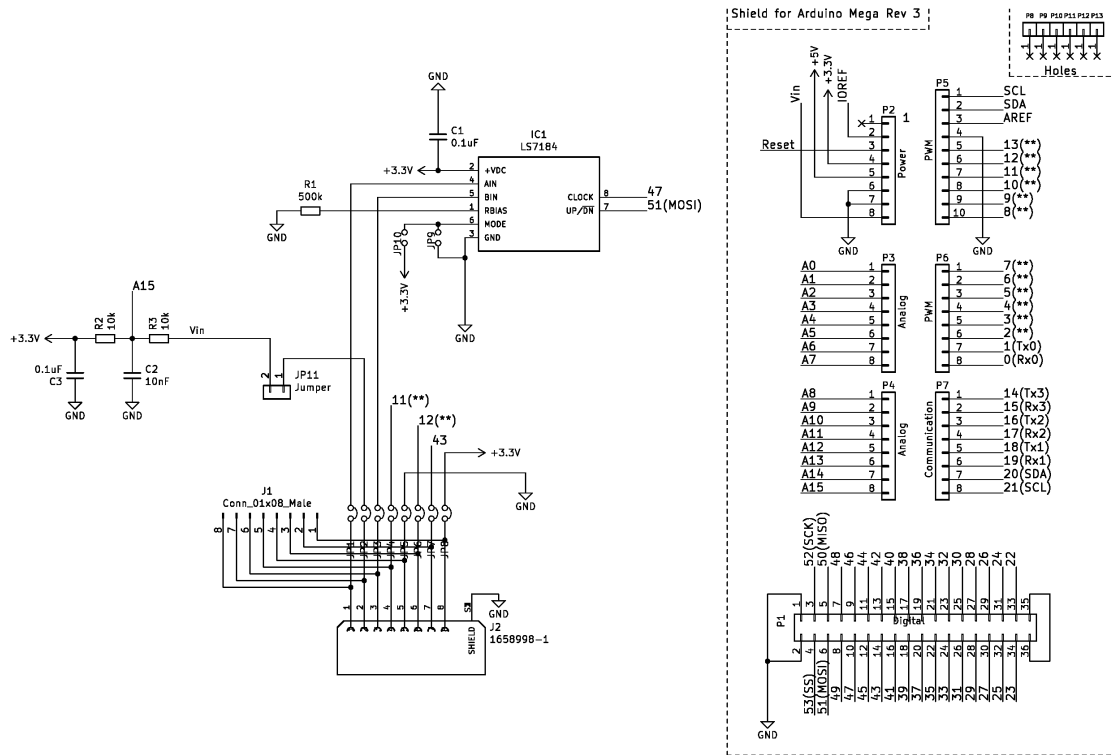
Motorlab STM32 Carrier Board Schematic



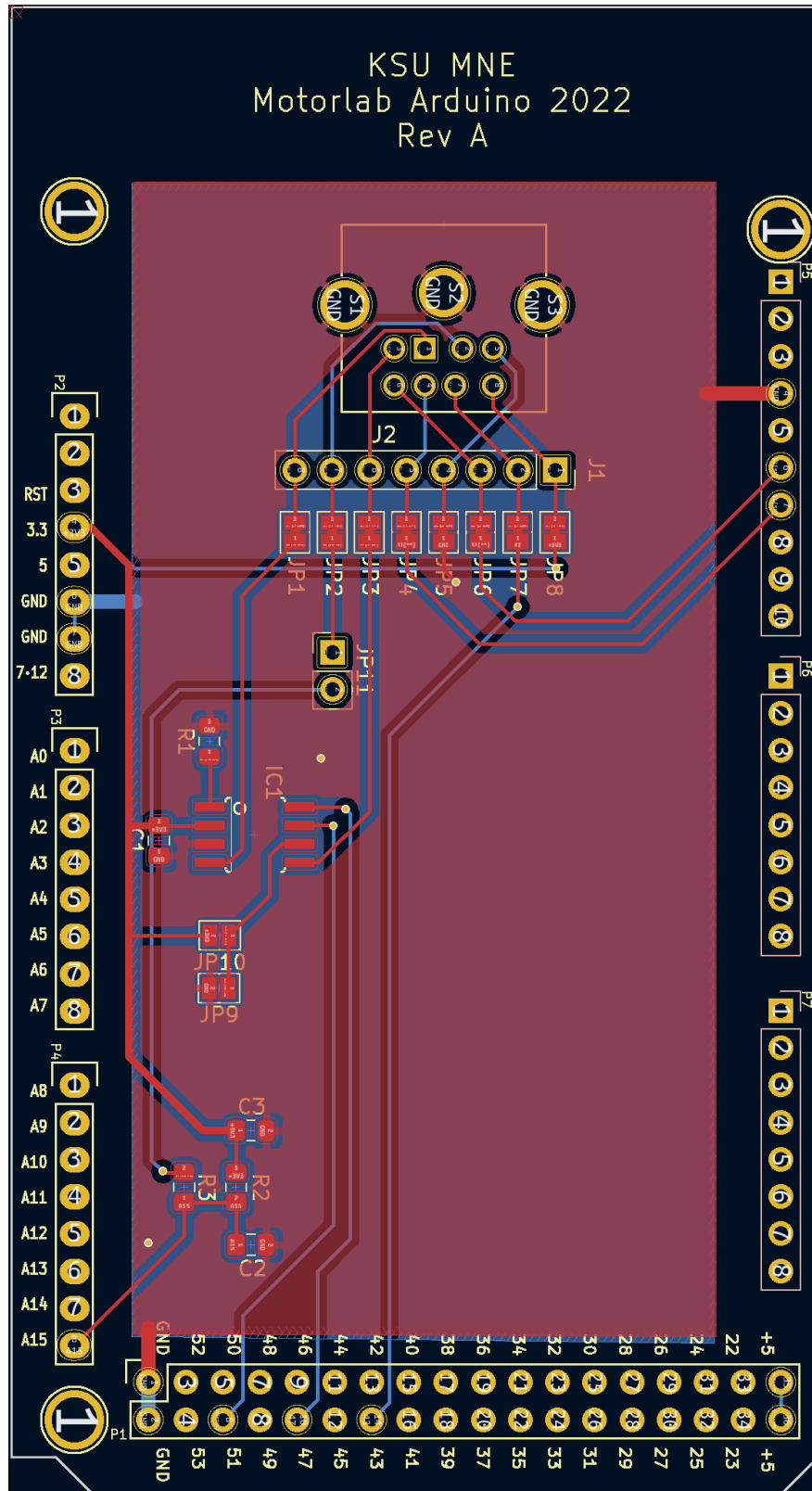
Motorlab STM32 Carrier Board Layout



Motorlab Arduino Shield Schematic



Motorlab Arduino Shield Board Layout



Appendix D – Student Survey Results

Did you like this lab?

YES! 👍

Yes, it was very quick and fun

Yes, it was fun.

Once we figured out how to run all of it. Its too early in the morning for this.

Yes

Yes. It was easier but a good way to change the normal Matlab labs.

Neutral, the instructions weren't clear about where to locate the proper file. VSCode is also not my favorite compiler.

Yes, it was something different! I think the instructions could be a little more clear, especially with the setup when you have to press all.

Yes

Yes

Wasnt terrible

We enjoyed this lab, we would not enjoy being quizzed on it though.

it was very different. definitely a change of pace. 6/10 from me.

It was a bit confusing, but not too difficult.

This lab was a little stressful, but only because it was unfamiliar. Directions maybe just need to be slightly more clear. Especially when it comes how to open workspaces.

Questionable

it's aight.

Yes

Yes, it took a short time :)

Yes

Yes. it was cool to use VS Code for the control

Yes.

Was a cool lab, but made frustrating with problems dealing with the Arduino code being able to be uploaded.

Yes it was fun and short. We learned a lot.

The lab was pretty straight forward, the platform that was used was a little bit different but understandable. Overall the lab was enjoyable.

I really enjoyed this lab, I think this is a really applicable way to implement and understand PID control. I do not know how PID is traditionally implemented in industry and having a microcontroller take care of it is cool. I had to learn a lot about implementing PID in arduino based on Brett Beauregard (i think thats his name) - he made a PID library (that I havent had much success with), but he also has a great blog implementing basic PID algorithm. Having

more activities would be cool but I also get that it takes extra long to make the GUI and all of the background stuff just to show students how this works.

Yeah, it was enjoyable, once I realized I didn't open the arduino file into an actual workspace.

I thought it was refreshing compared to the other labs. We were a bit rusty on our coding but the code was very simple. Maybe some more explanation on how the new GUI works.

Yes. This lab was different but was interesting and informative.

Yes, this is a good lab. I'm still kinda hazy on the controls, and what exactly the K_p , K_d , so on is doing. I followed instructions and copied the previous lab. This lab put it in the context of Arduino programming which I am more familiar with and the implementation of windup, speed, and position controls is clearly important in getting the arduino to tell a robot or motor to do what I want it to do.

This lab was kind of interesting for a change. I tend to understand code better and learn more when I write more of the code myself or at least type it in by myself. I totally understand that it is not feasible for us to write this code from scratch so I am not suggesting that.

yes, it was interesting.

I did not really enjoy this lab

Yes I liked this lab! I haven't used VS code in a while so it was a struggle relearning that.

I did. It was a neat change of pace, and it was very quick.

It was alright

yes it is fun :)

Yes, I like coding in C++ and it was a break from matlab. ME400 vibes. Much fun.

This lab was good to work in different GUI and use the Arduino as we had in other classes.

Although the lab allowed students to observe motor control using a different controller, a lot of the process was ignored. By this I mean most of the code was not discussed, we just inputted code and did not receive a briefing over the rest of it so it was more or less the same as MATLAB.

Appendix E – Selected Laboratory Exercises

Laboratory 2

Laboratory #2

Overview of laboratory procedure

1. The lab instructor will complete an introduction to MATLAB.
2. Experimentally determine a viscous friction coefficient, b , for the motor of the Motorlab.
3. Obtain data for an initial condition (IC) response for the motor speed, $\omega(t)$.
4. Compare the data for the IC response to the theoretical initial condition response found using an initial condition and the differential equation describing the dynamics,
 $T(t) = J\dot{\omega}(t) + b\omega(t)$.
5. Learn from the results by completing the Lab2 Narrative quiz in canvas.

$$T(t) = J\dot{\omega}(t) + b\omega(t)$$

$$T(t) = k_t i(t)$$

$$J = \text{inertia} = 129 \text{ g} \cdot \text{cm}^2$$

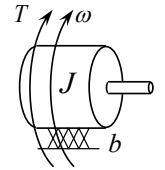
$$b = \text{friction coefficient} = \text{????}$$

$$k_t = \text{motor torque constant} = 5 \text{ N} \cdot \text{cm} / \text{A}$$

$$T(t) = \text{torque}$$

$$i(t) = \text{motor current}$$

$$\omega(t) = \text{angular speed}$$



ONE REPORT is due from each group, but you all are responsible for understanding what is in the report and how it was generated.

The starting MATLAB code for this lab, Lab2.mlx, can be downloaded from canvas. Then you will have to modify/complete it to complete the lab. You should use the help in MATLAB and use your instructor(s) to continue to learn the MATLAB language.

Estimating the Viscous Friction Coefficient

In steady state, where $\dot{\omega}(t) = 0$, $T(t) = b\omega(t)$. A constant torque/current should result in a constant speed in steady state. We should be able to estimate the viscous friction coefficient by obtaining steady state speed and current data. Fill in the table below with data obtained from the Motorlab. You will probably discover that friction is often a hard thing to model and that the linear, viscous friction, model is not completely accurate. We are attempting to find an "engineering estimate" that might be used in closed loop control where completely accurate models are not necessary.

Current Command (A)	-0.14	-0.12	-0.10	-0.08	-0.06	-0.03	0	0.03	0.06	0.08	0.10	0.12	0.14
Avg. Speed (rpm)													

Using the data from the table above obtain a plot of torque vs. angular velocity. On this same plot, draw the "best fit" line through the data by playing with the slope of the line described by $T = b_{\text{estimate}}\omega$. Generate this plot by completing the top of the m-file provided and running it with successive guesses at b_{estimate} .

Comparing Theoretical and Experimental IC Responses

- SOLVE THE DESCRIBING DIFFERENTIAL EQUATION for $\omega(t)$, given that $T(t) = 0$ and given that the initial condition (IC) is $\omega(t=0) = \omega_0$. Do this by hand, paying close attention to units. Use the equation for $\omega(t)$ to generate data for the theoretical IC response.
- Obtain experimental data for the IC response from the actual system
 1. Use a sample frequency of 500 Hz.

2. Jog the current to 0.15 Amps and allow the speed to settle. Wait four seconds for the data buffer to fill.
3. Hit the “Turn Off Motor Amp” button to turn off the current – count to 3 seconds – then immediately hit the “Save Data Buffer to Workspace” button. This should give data with about one second of initial speed 3 seconds of IC response.
4. In MATLAB and view the data with “mlolplots()”. In figure (2) of MATLAB you should see the IC response. Zoom in and use the cursor to find your actual initial velocity and a time at which the torque was set to zero.
5. Complete the bottom part of the m-file provided so that it generates the experimental response and the theoretical response on the same graph.

Things to Turn In

1. Plot 1): torque vs. angular velocity
2. Plot 2): initial condition response, this should include data cursors show values at on $t = \text{one time constant}$.
3. Completed m-file code with comments.
4. Hand development of the solution to the differential equation with the I.C.
5. **Complete the Lab 2 Narrative on canvas - *individually***

Items 1-4 need to be turned in as a single PDF document. You can use “save as” on the Matlab livescript to save it as a word document, then photograph your hand derivation in part 4 and insert it into the end of the word document. Add a cover page with your names, lab section, and the lab number. Finally, save the word document as a PDF.

To submit your PDF, follow these steps:

1. Login to canvas
2. Open “Assignments” on the left
3. Choose “Lab2”
4. Click on “Submit Assignment” and follow the directions to upload your PDF

Lab 2

Setup & Constants

```
format short;           % so all the numbers aren't 14 digits long
kt = 0.05;              % motor torque constant (N-m/A)
r2w = 2*pi/60;          % constant to convert from rpm to rad/s
J = 1.1e-5 + 0.19e-5;   % mass moment of inertia (N-m-s^2)
                        % motor shaft/rotor and double collar
```

Values obtained by successive runs of the motorlab using different open loop current commands

Fill in the rpm vector before running the code

```
% The vector below is the motor current command data for torque-speed plot
i=[ -0.14 -0.12 -0.10 -0.08 -0.06 -0.03 0 0.03 0.06 0.08 0.10 0.12 0.14];

% The vector below is the speed data for torque-speed plot
rpm = [???]; % fill in your RPM values that correspond to the currents in i

T=kt*i;      % convert current data to torque data
w=r2w*rpm;   % convert speed data from rpm to rad/s
```

Compare collected data and best fit theoretical line

use the slider to adjust b_estimate to make the fit line closely match the two endpoints

```
% Generate two data points for the line to estimate friction
% using the values at the beginning and end of the rpm vector

w_estimate = r2w*[rpm(1) rpm(end)]
b_estimate = 0.00036; % play with this to get good fit
T_estimate = b_estimate*w_estimate; % Construct a calculated linear fit for torque
                                     % (with intercept = 0 and slope = b)

% Generate torque vs. speed plot
figure(1);
plot(w,T,'*',w_estimate,T_estimate);
ylabel('Friction Torque (N-m)');
xlabel('Angular Velocity (rad/s)');
title('Input Torque vs. Angular Velocity with Estimated Linear Fit');
legend('Experimental data','Line fit to data','location','SouthEast');
```

Initial Condition response comparison - uses motorlab data described in IC response section.

Uncomment the following code after running motorlab and obtaining the "data" variable in your workspace

Column	1	2	3	4	5	6	7	8
Description	Time	Command	Motor Encoder	Load Encoder	Motor Speed	Load Speed	Current Command	Motor Current
Variable	t (sec)	θ_c (deg), ω_c (rpm), i_c (Amp)	θ_1 (deg)	θ_2 (deg)	ω_1 (rpm)	ω_2 (rpm)	i_c (Amp)	i (Amp)

Format of the motorlab data output matrix for reference

```

%% IC response part of this file requires "data" to be in the workspace
%% Uncomment the lines below to complete the IC response plot
%
% data_time = data(:,1);           % extract the first column of the data matrix
% data_rpm = data(:,5);           % extract the speed column of the data matrix
%
% wo = data_rpm(1)                % initial speed (rpm)
% tau = J/b_estimate;             % time constant (sec)
%
% derivative = diff(data_rpm);    % find the rough derivative of the RPM vector
% tv = data_time(derivative < -5); % time vector of when data was falling
% to = tv(1)                      % time when torque was shut off in original data
% %to = -.532;                   % visually assign if the derivative doesn't work
%
% data_time = data_time-to;       % shift the time vector of the data to zero at IC
%
%% Generate time vector for plot of theoretical response
% theory_time = 0:0.01:3;
%% using the time vector use the solution of the differential eq' with an
%% initial condition to generate the theoretical IC response
% theory_rpm = wo*exp(-theory_time/tau);
%
% figure(2);
% plot2 = plot(data_time,data_rpm,theory_time,theory_rpm);
% ylabel('Angular Velocity (RPM)');
% xlabel('Time (sec)');
% title('Actual and Theoretical Response to and IC of ??? rpm');
% legend('Experimental data','Theoretical');
% plot2(1).LineWidth = 3;
% plot2(1).Color = 'c';
% plot2(2).Color = 'black';

```

Laboratory 3

Laboratory #3

In this lab you will compare a proportional controller for the Motorlab position control to a proportional-derivative (PD) controller.

Overview of laboratory procedure

1. In preparation for the lab find the closed-loop TF (CLTF) given below. Also, go through the m-file code provided, reading the comments and understanding how it relates to the procedure.
2. Obtain data for the step responses.
3. Learn from the results by completing the fill-in-the-blanks.

Closed loop position control system model (a TF).

$$\frac{\theta(s)}{\theta_c(s)} = \frac{(K_d s + K_p) k_t k_{dr}}{J s^2 + (b + K_d k_t k_{dr}) s + K_p k_t k_{dr}}$$

Finding the closed-loop transfer function (CLTF)

Using the block diagram complete the equations below. Each blank represents one block. Then by back substituting to eliminate variables, starting from the bottom, obtain an equation with $\theta(s)$ and $\theta_c(s)$. Use this to find the CLTF.

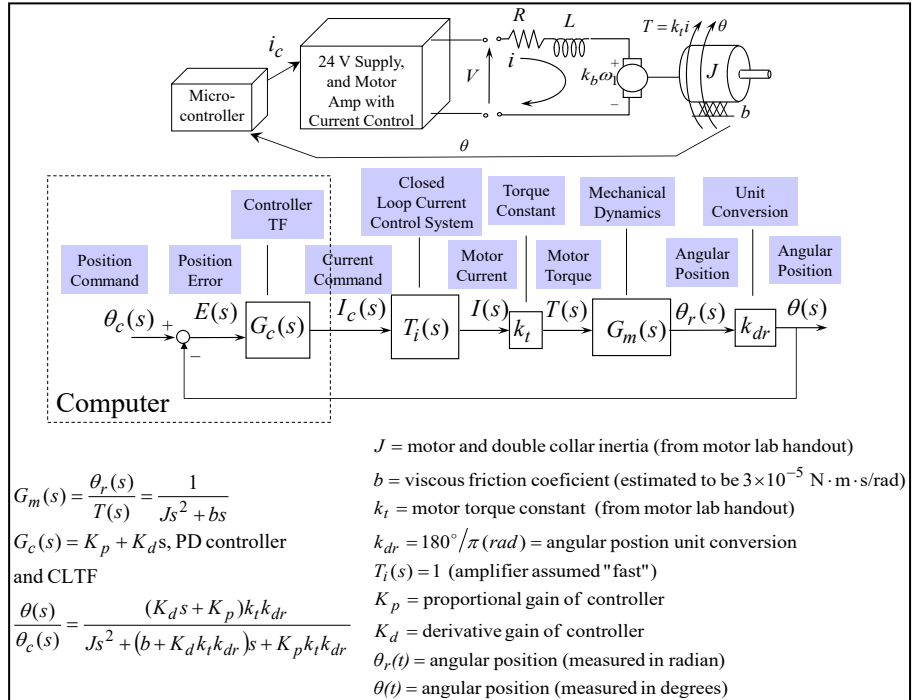
$$\begin{aligned} E(s) &= \theta_c(s) - \theta(s) \\ I_c(s) &= ______ E(s) \\ I(s) &= ______ I_c(s) \\ T(s) &= ______ I(s) \\ \theta_r(s) &= ______ T(s) \\ \theta(s) &= ______ \theta_r(s) \end{aligned}$$

Generating data for the closed-loop step response of a proportional controller and a PD controller

- Use a proportional controller with $K_p = 0.01$ (Amp/deg) and a PD controller with $K_p = 0.1$ (Amp/deg) and $K_d = 0.001$ (Amp·s/deg).
- Use a square wave with a magnitude of 200 degrees to generate the experimental step response data. Store the data in "stepdata", for the proportional controller and "stepdataPD" for the PD controller.
- Although the default sample rate should work, make sure that you get a nice/complete step response by using "mlposplots()" on the data right after you take it.

Things to turn in.

- Plot 1 – compare the theoretical and experimental step responses for the proportional controller.
- Plot 2 – compare the experimental step responses (not the models/theoretical) for the proportional and PD controllers.



- Your development of the CLTF
- Completed m-file code with comments.
- Complete the Fill-in-the-Blanks.

Fill-in-the-Blanks

In the development of the closed loop transfer function (CLTF) we used many relationships. As example we represented the relationship between motor current and the electromagnetic ????? on the rotor with a constant of proportionality, k , provided by the motor manufacturer. We also used a ????? friction approximation for the friction in the motor, although in the laboratory #2 we observed that is not completely accurate. However, some of our assumptions about the dynamics of the system are probably fairly accurate. For example, our motor ????? parameter probably captures the inertial effects of the dynamics fairly well. Because the controller (implemented in C-code on the micro-controller) measures the angle in ????? rather than radians, we included a unit conversion in the CLTF. This is an example of the Engineering details needed to solve real problems rather than textbook problems. The first figure demonstrates that it ????? possible to accurately predict the behavior of such a system by paying attention to these details.

The first figure demonstrates that the CLTF developed can be an accurate model of the system. There are obviously some differences between actual response of the system and that predicted by the theory. For example, because the decay of the oscillations is theoretically an ?????, the theoretical response's oscillations never go to zero, whereas the actual system does stop oscillating. This is probably explained by the nonlinear friction of the motor. However, after all the work in the development of the CLTF, we see that it does predict the oscillations and that it actually predicts the size and ????? of the oscillations very well. This figure also demonstrates that, in this case, a controller with only a proportional gain will result in a system with lightly ????? oscillations.

The second figure demonstrates that we can improve our controller by including appropriate dynamics in the controller, such as a ????? term. With the PD controller we can get a very fast response that does not ?????. It goes to the commanded value quickly and stays there. We ????? excited about continuing to learn about this stuff so that we can figure out how to make that happen for many different kinds of systems :)

Word bank for blanks. Note values should be what you found for your system!

Current		Degrees	Second-Order
	Oscillate		
Voltage		Furmans	Damping
	Damp		
Torque		Grades	Frequency
	Are Very (☺)		
Viscous		Is	Oiled
	Are Not		
Static		Is Not	Damped
Torque Constant		Exponential	Derivative
Inertia		Logarithmic	Integral

Lab 3

Setup & Constants

Run the motorlab first to populate the "stepdata" and "stepdataPD" variables

% You need add in kt and J from the last lab

```
b= 3e-5;          % N-m-s/rad - nominal viscous friction
kdr=180/pi;       % deg/rad
```

Motor model transfer function response

New commands: tf() and step()

% First step has no kd (derivative) - just kp (porportional)

```
kp = 0.01;        % proportional controller gain
```

```
kd = 0.0;         % derivative controller gain
```

% Construct the transfer function

% Closed loop TF model using the tf([num poly],[den poly]) format

```
Gcl=tf(kt*kdr*[kd kp],[J b+kd*kt*kdr kp*kt*kdr])
```

% Alternative method of constructing transfer functions

```
s = tf('s');
```

```
Gcl = kt*kdr*(??*s + ??) / (J*s^2 + (b+kd*kt*kdr)*s + kp*kt*kdr)
```

```
[theta_model,t_model]=step(Gcl); % Generate the *unit* step response of model
```

% Since the transfer functions we use in ME570 are linear, we can simply

% multiply the unit step response by a constant to obtain different step

% heights

```
theta_model = theta_model*???; % scale the response to the real step size
```

Plot theoretical (model) response vs real (stepdata) response

```
t_real=stepdata(:,1); % extract time column of the data matrix
```

```
theta_real=stepdata(:,3); % extract angle column of the data matrix
```

% Plot experimental (real) and Theoretical step responses for Kp=0.01

```
figure(1)
```

```
plot1 = plot(t_real,theta_real, t_model,theta_model);
```

```
ylabel('Motor Angle (deg)');
```

```
xlabel('Time (sec)');  
title('Actual and Theoretical Step Response for Kp = 0.01');  
legend('Experimental data','Theoretical');  
plot1(1).Color = 'red';  
plot1(2).Color = 'black';  
plot1(2).LineWidth = 2;  
plot1(2).LineStyle = ':';
```

Compare your two experimental results (from stepdata and stepdataPD)

- Extract time and theta from stepdataPD into variables t_realPD and theta_realPD
- Plot both real responses on the same graph - plot(x1,y1, x2,y2, x3,y3, ..., xn,yn)
- There is no theoretical data shown in this plot

Laboratory 5

Laboratory #5

In this lab you are to experimentally determine the coefficients for a model of the motor-and-spring system in the Motorlab. You should be able to determine the inertia and motor torque constant from the Motorlab handout. Then you should be able to estimate the other parameters (b, k_s) using experimental data. You are also to compare the step response of your model to experimental data. With the spring coupling in place and the load inertia locked down, the Motorlab apparatus will have the dynamic system described by the equations and schematic model to the right. This is the dynamic system we are studying in this laboratory.

Estimating the Spring Rate and Natural Frequency

Looking at the differential equation in the model it can be seen that in steady state with a constant torque $T = k_s \theta$. Therefore, we should be able to estimate the spring's coefficient by obtaining steady state position and current data. It is important to note that θ is measured from the equilibrium position with zero torque. Fill in the table below and use the data to estimate the spring rate.

Current (A)	-2.0	-1.5	-1.0	-0.5	0.0	0.5	1.0	1.5	2.0
Position (deg)									

Using the data from the table above obtain a plot of torque vs. angular position. On this same plot, draw the "best fit" line through the data by playing with the slope of the line described by $T = k_{estimate} \theta$. Generate this plot by completing the top part of the m-file and running it with successive guesses of $k_{estimate}$. Then, knowing the spring rate and the mass moment of inertia you should be able to estimate the natural frequency, ω_n .

Estimating the Damping Ratio and Friction Coefficient

Using a square wave input with an amplitude of 1 Amp, obtain a step response that can be used to estimate the damping ratio. Do this by fitting an envelope to the decay of the oscillations. The poles of a second-order, under-damped system are $-\zeta\omega_n \pm j\omega_d$, and the envelope of the oscillations is formed by using the exponential $e^{-\zeta\omega_n t}$. So the envelope of the step response of a second order system may be plotted using the step response of a first order system with a pole of $-\zeta\omega_n$, the real part of the second order poles. Knowing ω_n from above, you can vary ζ to fit an envelope to the second order step response.

Comparing the Actual Step Response to the Theoretical

Using your coefficients, you should now be able to generate a TF in MATLAB that approximates the system. You could either use the (k_t, J, b, k_s) set of coefficients or the $(k_{dc}, \zeta\omega_n, \omega_n^2)$ set of coefficients. Both should give you the same TF. Try it. Complete the bottom section of the m-file which plots data obtained from the Motorlab square wave along with the theoretical data.

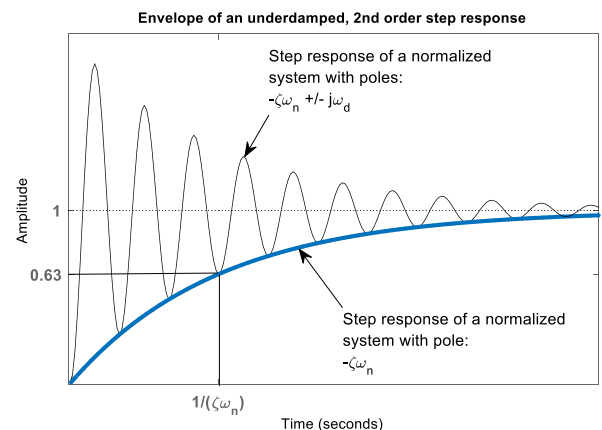
$$T(t) = J\ddot{\theta}(t) + b\dot{\theta}(t) + k_s\theta(t)$$

$$T(t) = k_t i(t)$$

J = motor inertia + collar inertia
 b = viscous friction coefficient
 k_s = spring constant
 $T(t)$ = torque (is proportional to motor current)
 $i(t)$ = motor current
 $\theta(t)$ = angular position (best if measured in rad for dynamics)

$$G(s) = \frac{\theta(s)}{I(s)} = \frac{k_t}{Js^2 + bs + k_s} = \frac{k_{dc}\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

ζ = damping ratio, ω_n = natural frequency, k_{dc} = dc gain
 ω_d = damped frequency of oscillation = $\omega_n \sqrt{1 - \zeta^2}$



Things In Report

- Plot 1: Straight line fit to Torque vs. Displacement data.
- Plot 2: Experimental step response and estimate of the envelope. This should have four data cursors: 1) steady state value, 2) value of the envelope at one time constant, and 3) & 4) two cursors at peaks to calculate the period of the oscillations.
- Plot 3: Experimental and theoretical step responses together.
- Completed m-file with comments.
- Table of coefficients.
- Completed fill-in-the-blanks.

Our coefficients (with units) for the two forms of the transfer function (TF) model found in this lab are given in the table below.

Coefficients of the TF with physical coefficients		Coefficients of TF in standard second order form	
$k_t =$		$k_{dc} =$	
$J =$		$\zeta =$	
$b =$		$\omega_n =$	
$k_s =$			

Fill-in-the-Blanks (remember to use units where applicable).

The envelope for the step response of the second order transfer function $G_{2nd} = \frac{k_{dc}\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$, is the step response of the first order where transfer function $G_{1st} = \frac{??*\zeta\omega_n}{s + \zeta\omega_n}$ where ?? is the ????.

In the second plot, which shows the step response of the actual system, we can see several things related to the TF modelling the system. It can be seen that with the step input that has an amplitude of [value], the steady state value of the response is about [value], which validates the dc gain of [value] given in the table above. Also, we can see two things related to the poles of the TF, which are $-\zeta\omega_n \pm j\omega_d =$ [value].

The frequency of oscillation is found from imaginary parts of the poles to be [value]. This predicts a period of oscillation of $2\pi/\omega_d =$ [value]. The actual period of oscillation for the real system is found from the difference between two peaks on the response to be [value]. The time constant for the envelope of the oscillations is found from real part of the poles to be $1/\zeta\omega_n =$ [value]. At this time the envelope of the oscillations has an angular position of [value]. This is approximately $0.63 \times \text{steady-state-value} =$ [value].

With the damping ratio found in this lab the frequency of oscillation, $\omega_d = \omega_n\sqrt{1 - \zeta^2}$, is [value] percent of the natural frequency, ω_n . With lightly damped systems, such as in this lab, the ???? and the natural frequency are essentially the same thing. Using the physical parameters of the Motorlab, b , J , and k_s , we can write equations for the more generic parameters of an under-damped second order system:

$$\omega_n = \sqrt{k_s/J}, \quad \zeta = b/2J\omega_n = b/\sqrt{4Jk_s} \text{ and } \omega_d = \omega_n\sqrt{1 - \zeta^2} = \sqrt{k_s/J} \sqrt{1 - b^2/4Jk_s} = \sqrt{k_s/J - b^2/4J^2}$$

Fill in the table below with increases or decreases.

	ω_n	ω_d	ζ	pole magnitude
As k_s increases	<u>?????</u>	<u>?????</u>	<u>?????</u>	<u>?????</u>
As J increases	<u>?????</u>	<u>?????</u>	<u>?????</u>	<u>?????</u>

Word bank for blanks that don't say value ☺ Note values should be what you found for your system!

Gain

Time Constant

Frequency of Oscillation

Damping Ratio

Setup & Constants

Even if you may not need them each time, constants like k_t , J , b , etc should generally be included

Note: the usual value for b assumes a freely rotating shaft and collar. We obviously don't have that this time so we will find it experimentally in the last section.

```
s = tf('s'); % allow us to write transfer functions in a nice way

kdr = ???; % degrees to radians

% vector of current commands to try in motorlab
i=[-2 -1.5 -1 -0.5 0 0.5 1 1.5 2];
% vector of position data - determined from motorlab
angle= [ ??? ];

% Unit conversions
theta=kdr*angle; % convert deg to rad
T=kt*i; % convert current vector to torque vector
```

Torque vs Angular Position Experimental and Estimated Plot

- Collect a squarewave response from the motorlab as described in the handout
- Be sure to name the variable "stepdata" so that the code matches your data
- Remember to set the **Sample Rate** to 500 Hz, then press the **Calculate Step Response Timing** before collecting stepdata

```
% generate two points for straight line fit
% the spring isn't likely to move more than 30 degrees
theta_estimate=(pi/180)*[-30 30]; % angle estimate

% What is the slope of our motorlab theta vs torque?
fit = polyfit(theta,T,1) % first order polynomial fit
k_estimate = fit(1) % slope is first element
% k_estimate = ???; % manual entry to fine tune the DC gain

% Torque estimate using slope found above
T_estimate = k_estimate*theta_estimate;

% plot straight line with data
figure(1);
plot(theta,T,'*',theta_estimate,T_estimate);
ylabel('Friction Torque (N-m)');
xlabel('Angular Position (rad)');
title('Input Torque vs. Angular Position with Estimated Straight Line Fit');
legend('data from Motorlab','straight line estimate');
```

Find Damping Ratio using Envelope Fit

Uncomment this section after completing the above section

- Estimate the damping ratio by fitting an envelope to the step response
- Requires step response data has been imported into stepdata
- Uncomment and complete the lines to find zeta

```
% J=1.1e-5 + 0.19e-5; % motor inertia (kg-m^2 or N-m-s^2/rad)
% ks = k_estimate; % spring constant (N-m/rad)
%
% Kdc = kt/ks; % DC gain of transfer function
% wn = sqrt(???); % natural frequency (rad/s)
% zeta_estimate = ???; % estimated damping ratio
%
% % Generate a step response from a first order system with a pole
% % equal to the real part of the 2nd order poles
% real_part = zeta_estimate*wn; % real part of poles
% envelope_TF = (Kdc*real_part)/(s + real_part)
% [envelope_y,envelope_time] = step(envelope_TF);
%
% % Extract data from the experimentally generated step response
% dataTime=stepdata(???); % extract time column of the data matrix
% dataAngle=stepdata(???); % extract third angle column of the data matrix
% dataTh = dataAngle*kdr; % convert to rad
%
% % plot estimated envelope with step response data
% figure(2);
% plot(dataTime,dataTh,envelope_time,envelope_y,'--');
% ylabel('Angular Position (rad)');
% xlabel('Time (rad)');
% title('Step Response Data with Estimated Envelope');
% legend('step response from Motorlab','estimated envelope');
%
```

Find coefficients and compare model to experimental data

Uncomment this section after completing the above section

- Find coefficients and compare model to experimental data
- Requires step response has been imported into stepdata
- Uncomment and complete the lines below to generate the step response
- comparison and to print coefficients

```
% % Print coefficients for different forms of the TF
% zeta = zeta_estimate % damping ratio
% wn=sqrt(????) % natural frequency (rad/s)
% b=2*zeta*wn*???? % friction coefficient (N-m-s/rad)
% wd=wn*sqrt(1-zeta^2) % damped oscillation frequency (rad/s)
% Tosc=2*pi/wd % period of oscillations (sec)
% J = J % motor inertia (N-m-s^2/rad)
% ks = ks % spring constant (N-m/rad)
%
% % Print both forms of the TF
% G = (kt/J)/(s^2 + 2*zeta*wn*s + wn^2)
% Gcheck = kt/(J*s^2 + b*s + ks)
% zpk(Gcheck)
%
% I = 1; % size of step input
% [theoryTh,theoryTime]=step(I*G); % get the theoretical step response

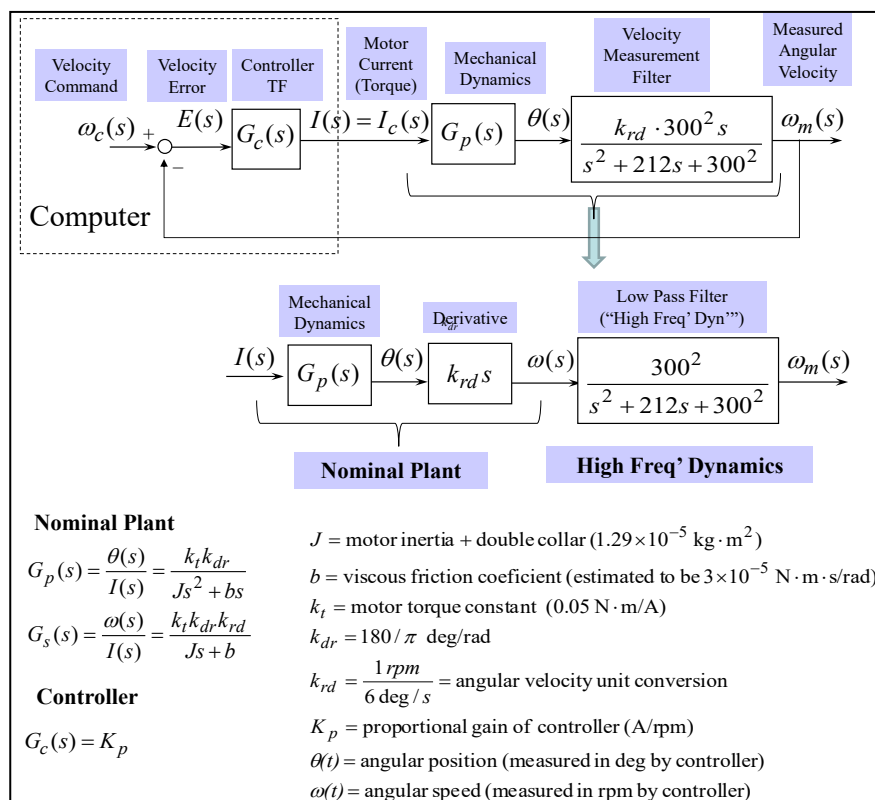
% DO THE PLOT COMPARING THE ACTUAL STEP RESPONSE TO THE MODEL
```

Laboratory 7

Introduction

In this lab we will use the velocity control system in the Motorlab to look at the concept of "higher frequency dynamics." This lab should illustrate there are always some higher frequency dynamics that will affect you if you "turn up the gains" too much. We can ignore them to a point, but they are there. Often, we do not have a good model for them, or even know for sure what is causing them, but they are there.

We have a rule of thumb: *We can ignore open loop poles and zeros when they are more than 10 times larger (in terms of magnitude, which is the distance from the origin of the s plane) than the closed loop poles that result from ignoring them.* You should note it refers to the effect of open loop poles on the closed loop system. This is typical in control system design. We are usually trying to make predictions or calculations for the closed loop system using open loop models.



Collecting Data

You should collect data for at least four gains, listed in the table below. Use a sample rate of 500 Hz for all the data. For the first three gains you should collect a step response, utilizing the Calculate Step Response Timing button to get half a wave cycle for the first three data points. For the last gain we want to capture the unstable growth of the response and will utilize the below Special Procedure.

Gain, K_p (units?)	Square Wave (rpm)	Name of MATLAB workspace matrix for data
0.0008	1000	data1
0.0016	1000	data2
0.0032	1000	data3
0.008	Use Special Procedure below	data4

Table 1: Information for Acquiring Data

Special Procedure:

1. Turn off the amplifier.
2. Set the gain to 0.008.
3. Change the rpm to 50 rpm in the manual command window.
4. Turn on the amplifier. Then wait one second and save the data to the workspace.
5. Use `mlspeedplots(data4)` to plot the data and zoom in on the exponential growth in both speed and current plots.

Things to Turn In

- Include the three plots for the measured and calculated step response when $K_p = 0.0008$, $K_p = 0.0016$, and $K_p = 0.0032$. These should include three responses, one measured and the two models of the closed loop system.
- Use datatips on the plot for $K_p = 0.0032$ to show that the frequency of oscillation and time constant of your experimental data roughly matches the 2nd order pole frequency and time constant returned by the `damp()` command.
- Include a documented (including units) copy of your MATLAB code.
- Completed Fill-in-the-Blanks (with units).

Fill-in-the-Blanks

The “damp” command in MATLAB prints out the poles of a TF in both Cartesian form and polar form. The Cartesian form gives the real and ???? parts, while the polar form gives the damping ratio and the magnitude. The angle in polar form is directly related to the ????.

The nominal system model is the one without the higher frequency dynamics (i.e. without the low pass filter on speed). The nominal ???? TF has one pole and the nominal closed loop TF has ???? pole(s). With the higher frequency dynamics, the open loop TF has ???? poles and the closed loop TF has ???? poles. Closing the loop ???? change the number of poles. It changes the ???? of the poles (i.e. they move in the s-plane).

One of the open loop poles in the system with the higher frequency dynamics has the same value as the nominal open loop pole. This pole is much ???? in magnitude than the other two poles. The magnitude of the other open loop poles in the model with the high frequency dynamics is [value], which is their natural frequency. Therefore, according to our rule of thumb, when the ???? loop poles of the nominal system approach a magnitude of [value] we might expect the accuracy of the nominal model to be questionable.

At a gain of $K_p=0.0008$ [unit] the magnitude of the closed loop pole of the nominal model is [value] rad/s and is close to the pole at [value] rad/s in the closed loop model with the higher frequency dynamics. At this gain we are very close to our rule of thumb, and although there are slight differences, the step responses from the two models and from the actual system all look very similar. The other two closed loop poles in the model with the filter have complex values of [value], which have a magnitude of [value] rad/s. This magnitude is much larger than the magnitude of the real pole and therefore these poles [q6] affect the response very much.

At a gain of 0.0016 [unit] the magnitude of the closed loop pole of the nominal model is [value] rad/s, so it is ???? than the magnitude predicted by our rule of thumb where we will start to see significant differences between the two models. At this gain the real closed loop pole of from the higher order model has a value of [value], so we might expect similar behavior from the two models. However, the other two closed loop poles in the model with the filter have complex values of [value] rad/s, which have a magnitude of [value] rad/s. At this gain the magnitude of the real pole and the complex poles are closer together than for the lower gain and we begin to see the effects of the complex poles with some ???? in the responses of the higher order model and in the actual system.

At a gain of 0.008 [unit] the closed loop pole of the nominal model is at [value] rad/s, which predicts a fast, first order, stable response. However, the higher order model and the actual system are ????, as predicted by the positive real parts of the two closed loop poles at [value] rad/s. Although the oscillations in the actual response from the Motorlab do not grow to infinity, they do begin grow and then reach a ???? cycle after a few oscillations. It can be seen in plot of the motor current that it saturates at [value] Amps.

Word bank for blanks. Note values should be what you found for your system!

Imaginary	Real	Damping Ratio		Natural Frequency	Open-
Loop	Closed-Loop				
One		Two	Three	One	
	Two		Three		
One		Two	Three	Does	
	Does Not		Location		
Flavor		Smaller	Larger	Open	
	Closed		Do		
Do Not		Smaller	Larger	Damping	
	Oscillations				
Rock Solid	Unstable	Steady-State		Limit	
Infinite					

Lab 7

Setup & Constants

Include the constants required to construct G_s , the **Nominal Plant**, and write the transfer function. Find ω_n and $\zeta\omega_n$ from the **High Frequency Dynamics** transfer function and use them to construct G_{hf} .

Closed loop pole exploration

```
figure(1);
graph_hold = false;
if graph_hold
    hold on;
else
    clf(1);
end
kp = 0.0002;

Tnominal_slide = feedback(kp*Gs,1); % CL TF for nominal model
Thf_slide = feedback(kp*Ghf*Gs,1); % CL TF for higher order model
[p_ol,z] = pzmap(Ghf*Gs);
[pnominal_slide,z] = pzmap(Tnominal_slide);
[phf_slide,z] = pzmap(Thf_slide);

p1 = plot(real(p_ol), imag(p_ol), 'og', ...
          real(pnominal_slide),imag(pnominal_slide), '+r', ...
          real(phf_slide),imag(phf_slide),'*b'); % Nominal model
set(p1,'markerfacecolor',['g' ; ; ]);

s=sprintf('Open Loop Poles, o \n');
s=[s sprintf('Poles of nominal CL system, + \n')];
s=[s sprintf('and higher order CL system, * \n')];
title(s);
xlim([-500 300]);
ylim([-400 400]);
grid on;
disp(real(pnominal_slide));
disp('');
disp(real(phf_slide));
```

Determine poles and zeros for each k_p

```
kp=[ 0.0008 0.0016 0.0032 0.008]; % array of kp gains used
```

```

for i=1:length(kp)                                % cycle through the gains
    Tnominal(i) = feedback(kp(i)*Gs,1);           % CL TF for nominal model
    Thf(i) = feedback(kp(i)*Ghf*Gs,1);           % CL TF for higher order model
    l = sprintf('----- %d -----',kp(i));     % string generated from current kp
value
    disp(l);                                       % show me the string!
    damp(Tnominal(i))                             % show the poles in polar form
    damp(Thf(i))                                  %      ""
end

```

Plots

Step response for $k_p = 0.0008$

```

tfinal = .3;
[speedN,timeN]=step(1000*Tnominal(1), tfinal); % step response of nominal model
[speedHF,timeHF]=step(1000*Thf(1), tfinal);    %      ""    of higher order model
speed= data1(:,5);    %extract the first speed column of the data matrix
time=data1(:,1);      %extract the time column of the data matrix

figure(2);
plot(timeN,speedN,timeHF,speedHF,time,speed);
title('Plot of CL step response for Kp=0.0008');
legend('model with nominal dynamics','model with hi freq
dynamics','actual','location','southeast');
xlabel('time (sec)'); ylabel('speed (rpm)');

```

Step response for $k_p = 0.0016$

```

tfinal = .3;
[speedN,timeN]=step(1000*Tnominal(2), tfinal); % step response of nominal model
[speedHF,timeHF]=step(1000*Thf(2), tfinal);    %      ""    of higher order model
speed= data2(:,5);    %extract the first speed column of the data matrix
time=data2(:,1);      %extract the time column of the data matrix

figure(3);
plot(timeN,speedN,timeHF,speedHF,time,speed);
title('Plot of CL step response for Kp=0.0016');
legend('model with nominal dynamics','model with hi freq
dynamics','actual','location','southeast');
xlabel('time (sec)'); ylabel('speed (rpm)');

```

Step response for $k_p = 0.0032$

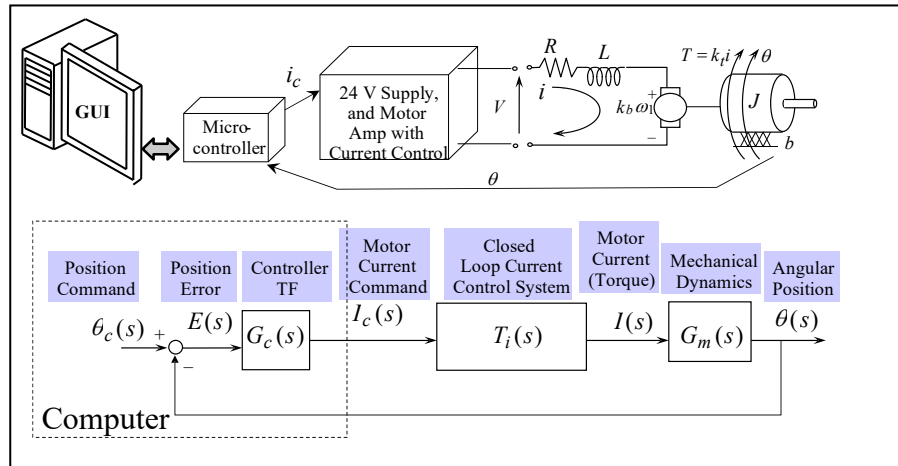
```
tfinal = .6;
[speedN,timeN]=step(1000*Tnominal(3), tfinal); % step response of nominal model
[speedHF,timeHF]=step(1000*Thf(3), tfinal);    %      ""      of higher order model
speed= data3(:,5);    %extract the first speed column of the data matrix
time=data3(:,1);      %extract the time column of the data matrix

h = figure(4);
p = plot(timeN,speedN,timeHF,speedHF,time,speed);
title('Plot of CL step response for Kp=0.0032');
legend('model with nominal dynamics','model with hi freq
dynamics','actual','location','southeast');
xlabel('time (sec)'); ylabel('speed (rpm)');
```

Arduino Laboratory

Laboratory #8

In previous labs you controlled the Motorlab with the onboard STM32F407 microcontroller board, which uses MATLAB for its user interface. The STM32F407 does the heavy work of interfacing with the other hardware such as reading the encoder and commanding the amplifier to produce a specific current. It also implements a controller loop and speed filter. Today you will be responsible for coding the controller portion with an Arduino.



A. Open the quiz in Canvas to grab screenshots and do the quiz questions as you go!

B. Getting Started

1. Download **MotorlabArduino-Setup.bat** from canvas and unzip it to your DESKTOP
2. Open Visual Studio Code – the following steps are all done inside Visual Studio Code.
3. Select File -> Close Workspace (if you already have a workspace you are working from)
4. Select File -> Open Workspace,
5. Open from your desktop **MotorlabArduino\Code\Arduino.code-workspace**
6. Click on Arduino_motorlab.ino on the left to open the code editor
7. Choose the appropriate COM port (bottom right of Visual Studio Code) (DO NOT choose Comm 1)

C. The Code

Skim through the code. It may look daunting at first, but the controller portion is only a handful of lines. These few lines are what we have been talking about all semester. The rest of the code is all setup, GUI and hardware interfacing. Read the comments starting at line 177 and complete the controller. No other code modification is necessary. Once you have completed your controller, upload it to the Arduino.

D. Instructor Verification

Ask an instructor to verify that your Motorlab behaves properly before conducting the following experiments.

E. The Experiments

Open Terraterm by going to **U:\TTERMPRO** and clicking on **ttermpro.exe**. Choose to open a Serial connection to the appropriate port (same port you pick for step A6). Read the key definitions at the bottom of the GUI, then enable the Motorlab (E) (ML Status will change to 1). Then type in 1500 (you do not need to click enter after entering numbers). Take a screen capture and then record your observations in quiz.

READ THIS PARAGRAPH BEFORE EXECUTING! Reset the Motorlab (**R**) and set it to speed mode (**S**) (values should all be zero except ML Status and red header now says Mode: S). Set the speed target to 9000 RPM and let the motor run for 5-10 seconds. Take a screen capture. Take note of the integral term value. Do **NOT** reset, type in 900 RPM and observe the speed (wait until the speed reaches 900RPM and then take screen capture). As you run this section pay attention to: Is 9000 RPM faster than the motor can go? What happened to the integral term when you tried to reach 9000 RPM? What happened when you changed the RPM to 900?

Turn on integral windup control (**W**) (after enabling, the header row with show Windup Control: 1) and then repeat the previous paragraph. What was different for each step?