Adaptation of the introductory control system laboratory apparatus for model based design

by

Dong Hyun Kim

B.S., Kansas State University, 2018

A THESIS

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

2021

Approved by:	Approved by:
Co-Major Professor Dr. J. Garth Thompson	Co-Major Professor Dr. Warren N. White

Copyright

Abstract

Model Based Design (MBD) has been enabling engineers to rapidly develop, test, analyze, and implement control system concepts. Many companies, especially in automotive and aerospace industries, have seen increases in design and development productivity and decreases in cost and time while retaining the quality of firmware in embedded systems. As MBD is becoming a trend in industries’ standard practice, it is important for the university education to adapt and integrate these tools into the students’ learning experience.

The MBD approach enhances the deployment of complex control systems by abstracting their complexity into graphical representation of system models. The models support an array of analysis and simulation tools that permit the designer to progressively evaluate alternative control structures and components to reach the required performance requirements. The tools ultimately lead to the auto-generation of the source code for the embedded systems firmware. But it is important in the education context that students understand the engineering concepts underlying the tools and to not obscure too much of the backend information. To appreciate the automation of firmware generation students should have a minimal understanding of basic coding practices to maximize the learning outcome.

This thesis presents the MBD methods used to automate the generation of new firmware of an existing laboratory apparatus called the MotorLab and to integrate MBD concepts into an undergraduate controls course. MotorLab is used in the introductory controls course at Alan Levin Department of Mechanical and Nuclear Engineering at Kansas State University. The updated firmware is carefully examined to ensure the full range of functionality of the original lab device to deliver the same effective lab exercises and to demonstrate the application and benefits of MBD.

Table of Contents

List of Figures

List of Tables

List of Software Listings

Acknowledgements

I am thankful for Dr. Schinstock for inspiring me to study automatic control theory, for Dr. White for continuing to educate me and agreeing to be my advisor, and for the graduate school for giving me the opportunity to pursue my thesis at Kansas State University. I would also like to thank Dr. Thompson and Dr. Eckels for their support in finishing my thesis, Dr. Brockhoff and Dr. Flippo for sharing insights regarding class objectives and research projects, and Mr. Patterson for helping me with IT related issues.

Also, I thank my loving parents and my brother for supporting me throughout my education and asking nothing in return while believing in me.

And last but not least, I would like to dedicate this thesis to my dearest Laura who supported me throughout my happiest and toughest times working on the thesis. This would not have been possible without you.

Nomenclature

MBD – Model based design

RTOS – Real Time Operating System

UAS – Unmanned Aerial Systems

MIL – Model in the loop

SIL – Software in the loop

PIL – Processor in the loop

HIL – Hardware in the loop

USART – Universal Synchronous/Asynchronous Receiver/Transmitter

CDT – C/C++ Development Tools

SWD – Serial Wire Debugger

GDB – GNU Debugger

VCP – Virtual Com Port

PWM – Pulse Width Modulation

IDE – Integrated Development Environment

HAL – Hardware Abstraction Layer

CCR – Capture Compare Register

USB OTG FS – Universal Serial Bus On-The-Go Full Speed

PID – Proportional, Integral, and Derivative

Kp – Proportional Gain

Ki – Integral Gain

Kd – Derivative Gain

Chapter 1 - Introduction

University engineering faculty members have always appreciated the importance of engaging engineering students in hands-on experiments with laboratory practices. This teaching method of applying theories taught in lectures into more tangible applications has given students better understanding of engineering principles and practices. It also results in increased confidence before heading into industry as engineers [1]. It has also been shown in pedagogical studies that young adults in higher education have greater comprehension and retention of learning experiences with interactive exercises ([2], [3], and [4]). But there are significant hurdles associated with engineering labs. Teaching laboratories are expensive to develop, operate, and maintain and there are few sources from which those resources may be obtained. Some institutions have added equipment fees to the students' tuition but there are many demands for those resources. Many institutions and faculty members may lack the resources for operation of specific teaching laboratories. This may especially be the case in the controls engineering discipline where the equipment often involves hardware and software pertaining to automotive and aerospace industries.

The application of digital computers to control physical systems revolutionized the controls industry. The use of a computer to control a physical system is called an embedded control system. The design and implementation of embedded control systems involve selection of the computer (often a microcontroller), the sensors and actuators that connect the computer to the physical system, and the creation of the computer software. The MathWorks company capitalized on this transformation by producing a suite of tools to support the development and implementation of these systems. Their tools make use of graphical user interfaces and mathematical models to represent the components of the systems. These tools reduce the cost and time to design and implement embedded control systems by supporting a progressive set of analysis and simulation tools.

The controls industry has adopted this methodology called Model Based Design (MBD) using MathWorks products and other third-party add-on products that integrate into MathWorks tools. MBD manages the complex nature of embedded control systems by using graphical abstractions and mathematical models. These tools provide an interactive environment to design, test, analyze, and deploy these systems. This is achieved by integrating the four main stages of MBD: Model-In-Loop (MIL), Software-In-Loop (SIL), Processor-In-Loop (PIL), and Hardware-In-Loop (HIL).

MIL is a model development and simulation stage. This stage depends on accurate computer models of the physical system being controlled, the controller, and interconnecting devices being developed. The models are used for simulation and analysis, and for tuning the controller. The MIL stage is important in saving costs since the computer model of the physical system can be used instead of setting up expensive hardware under a testing environment. MIL provide a means to evaluate various components of the system prior to expenditure for the physical devices. The same computer model is then used in MIL, SIL, and PIL to develop and test the controller software.

Once the analysis of the MIL simulation returns adequate performance, the next stage is SIL. SIL uses and extends the models from the MIL stage. The SIL stage enables testing the controller in a controller emulation process. The MIL and SIL stages are completed in a general-purpose PC without any special hardware. The software emulated processor mimics the constraints of a specific processors to allow various controllers to be evaluated before acquiring the microcontroller.

Following the SIL stage is the PIL stage which involves a real microcontroller. In contrast to SIL, this stage runs the simulation with the control loop running on an actual microcontroller. PIL requires a connection with adequate data transfer rate between a host PC and microcontroller.

The final stage is HIL, which is the verification of a deployed embedded control system in the most realistic setting using actual physical hardware. There can be various levels of HIL from only implementing the interconnecting sensors and actuators to including the entire functioning physical system. At any given stages of implementing MBD, reiteration back to prior stages is often necessary.

A major benefit of the MBD methodology is the auto-generation of optimized controller code. MBD saves a significant amount of time spent with the debugging process due to handwritten code being susceptible to human errors ([5], [6], and [7]). In the educational context, the time saved in code production can provide more time for testing, analyzing, and engaging students in learning more about the embedded control system design procedures. Instead of repeating the same process of writing and debugging code manually, students will be able to focus on the simulation, verifying the response of the hardware, and studying the effects of the controllers on the physical system.

Working with the mathematical and graphical representations of physical systems can aid students in understanding mathematical principles in control systems analysis and design. The simplified visual representation of each components block will help relating mathematical principles to the physical control system.

The first two stages of MBD can be accomplished on general-purpose PCs without the expenditure on any physical system hardware. Additionally, PIL can be accomplished with many inexpensive microcontroller boards. With the help of MBD, students can be exposed to designing various embedded control systems that might be very expensive or impossible to implement in a university laboratory. But in order for students to experience the full scope of the MBD methodology, it is critical to include some level of physical hardware. With more realistic HIL setups, students can have more accurate response data to analyze and compare with their simulated system from previous stages. And laboratory exercises should be designed carefully to maximize the student outcome from learning with the MBD. The lab should also help students with retaining the understanding of basic coding practices while reinforcing the control theory with more hands-on exercises.

A faculty member in the Mechanical and Nuclear Engineering Department at Kansas State University has developed an embedded system [8]. The system can control various operational modes of a small brushless motor. The device is called the MotorLab and has gone through several stages of development. The system is used in a set of experiments in a laboratory associated with an automatic controls course that is required in the curriculum. The controls laboratory consists of 10 experimental stations each with a MotorLab device and a general-purpose PC. Since the system was developed and is maintained internally using student employees, it is difficult to determine its cost. It remains unknown as if it will continue to be maintained since the faculty member who developed the hardware, and the embedded firmware is no longer with the University.

Other methods of modifying and updating the laboratory device have been explored ([9], [10], [11], and [12]) at Kansas State University. Most of the effort was aimed at lowering the cost and with increasing the distribution of the apparatus. It might produce better learning outcomes if a suitable device can be developed at a cost so that each student can own one.

Experience with a similar situation in a related course produced outstanding experiences for many of the students who became very engaged with the device and could spend as much time as they wished learning with the device.

The current set of laboratory assignments in the controls course do not address MBD explicitly. However, several of the assignments involve application of the MathWorks tools to model, analyze, design, and simulate the controller for the MotorLab system (essentially the MIL stage). But students are not exposed to the embedded MotorLab software or to the other stages of MBD. Students are encouraged to change the parameters of the controller and observe the effect on the response of the system. The MotorLab device seems well suited for a broader application of MBD and that is the subject of this thesis.

The alternative to internally developing laboratory equipment suitable for teaching the concepts of MBD is to purchase laboratory hardware and software. But this solution of purchasing all the necessary equipment comes with an exceptionally expensive price tag. Unlike the industries' implementation of MBD using the resources available to them, universities might not have the same capacity to purchase the industrial grade software and hardware. Especially when the materials are required for every student in the curriculum.

To minimize the financial burden of an important educational experience for our students, this thesis will focus on two main objectives. The first is to lower the cost associated with adopting MBD by not requiring acquisition of new hardware and software. It will assume the availability of the minimum requirement of essential software and hardware such as MATLAB Coder and Simulink Coder, and a microcontroller with a suitable physical system to control (the MotorLab system). Then, the thesis will explore creating additional tools to work with MATLAB and Simulink along with available free software packages. This will substantially reduce the software acquisition cost to enable MBD functionality. The second objective is to examine the quality of generated controller code to ensure that it is suitable for our intended educational purposes. It will be carefully compared with the current MotorLab firmware by running example labs. The objectives are to reduce the cost without compromising the quality so that the students will gain access to the industrial tools that are widely used. It will boost students' confidence and competence as they join the engineering workforce after graduation.

In the following chapters, the development and deployment of MBD capability will be covered as follows:

- description of the hardware currently used in the control course,
- modelling of the physical hardware,
- acquisition and development of additional tools and software and adapting to their limitations with workarounds, and
- verification using the HIL.

Finally, several lab exercises from the introductory controls course, ME570, will be performed using the MBD approach and compared to the results to the original MotorLab exercises. They will be used to demonstrate the practicality of using MBD for the development of embedded control systems in an educational setting.

Chapter 2 - Hardware

2.1 MotorLab

To investigate the implementation of MBD capability, the existing laboratory apparatus has been evaluated and modifications required to support the four stages of MBD identified. The existing device called MotorLab is pictured in Figure 2.1 and Figure 2.2. It has been in use for over a decade in the laboratory associated with the mechanical engineering automatic controls course. It features a rugged design and construction, as well as well written custom firmware. It also includes a custom graphical user interface which provides access to the setup the laboratory experiments and record system responses. To make the MBD approach more familiar for students and cost effective, this thesis aims to utilize the hardware that is already provided in the laboratory and to replicate the graphical user interface functionality.

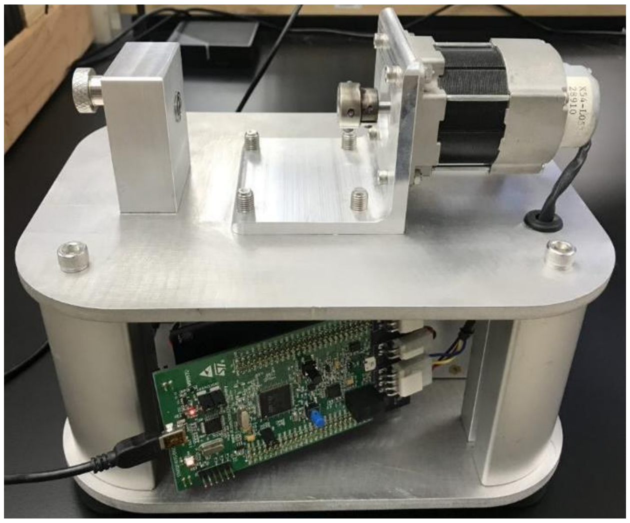


Figure 2.1 MotorLab Apparatus

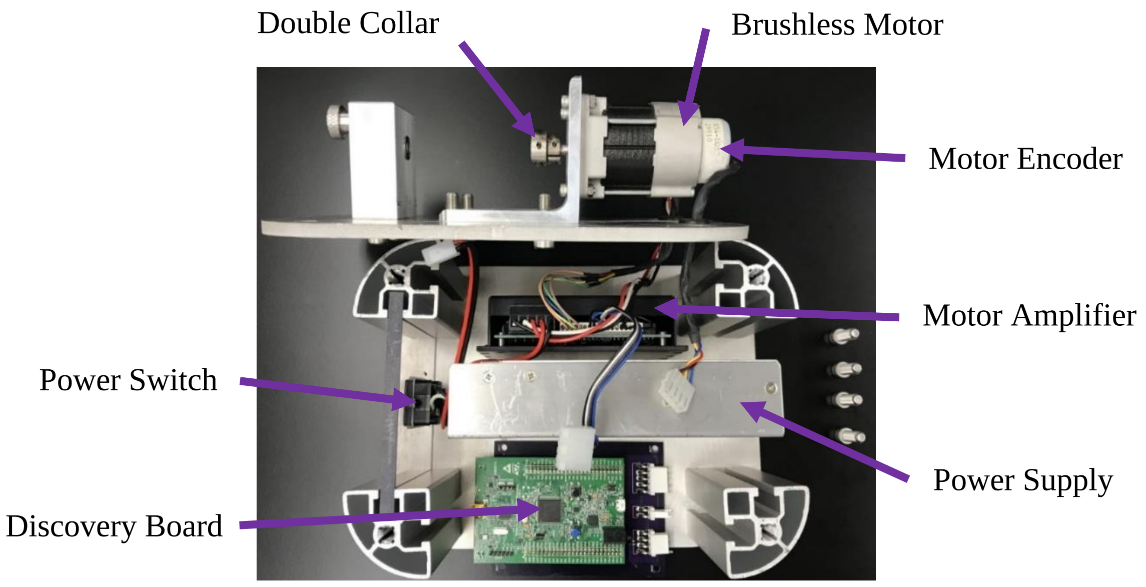


Figure 2.2 Components of MotorLab Apparatus

The apparatus is composed of four main components:

- a brushless motor with an encoder,
- an amplifier,
- a 24 Volt power supply, and
- a microcontroller development board.

The vendor of the microcontroller used is STMicroelectronics. It is a multi-national company that offers a variety of Micro-Controller Units (MCU) and Micro-Processor Units (MPU) used in a wide range of embedded control applications. Exposure to this type of hardware can help students become familiar with the capabilities and resources they provide, and the tools used to integrate these devices into physical systems. This exposure makes the students attractive to companies that seek employees with relative experience with specific MCU or MPU products. More details about the hardware, the driver libraries, and the Hardware Abstraction Layer driver will be covered in the software chapter.

2.2 STM32 Discovery Board

The development board used is the STM32F4 Discovery Board manufactured by STMicroelectronics. The embedded software for the MotorLab application was developed for this hardware and for this specific application. The primary task of the software is the interrupt driven control loop that is executed at 10kHz. Other peripheral tasks with lower priorities run in the background without compromising the primary task. The MBD processes are implemented on a general-purpose host PC. The PC must be capable of performing the MBD algorithms at the same control loop rate. In the PIL and HIL stages, the host PC and the microcontroller must be able to exchange data at this same loop rate.

Software for the Discovery board is developed in an application on a PC and ported to the Discovery board through a USB port. The USB port communicates with the onboard processor through the ST-LINK, a device on the board that manages that communication process. The ST-LINK also supports software debugging and limited data retrieval which is not fast enough to support the PIL and HIL applications.

Three options were pursued to obtain the required exchange data between the host PC and the Discovery board. The Discovery board has several Universal Synchronous-Asynchronous Receiver/Transmitter (USART) ports. The Transmit and Receive (TX/RX) lines are available on the board as designated pins. The first option considered was to utilize the VCP (Virtual Com Port) modification. This modification consists of the Transmit and Receive pins (TX/RX) of any available USART to be soldered to the ST-LINK TX/RX pins. The TX/RX pins are adjacent to the bottom left corner of the ST-LINK chip. A picture of this modification is shown below in Figure 2.3.

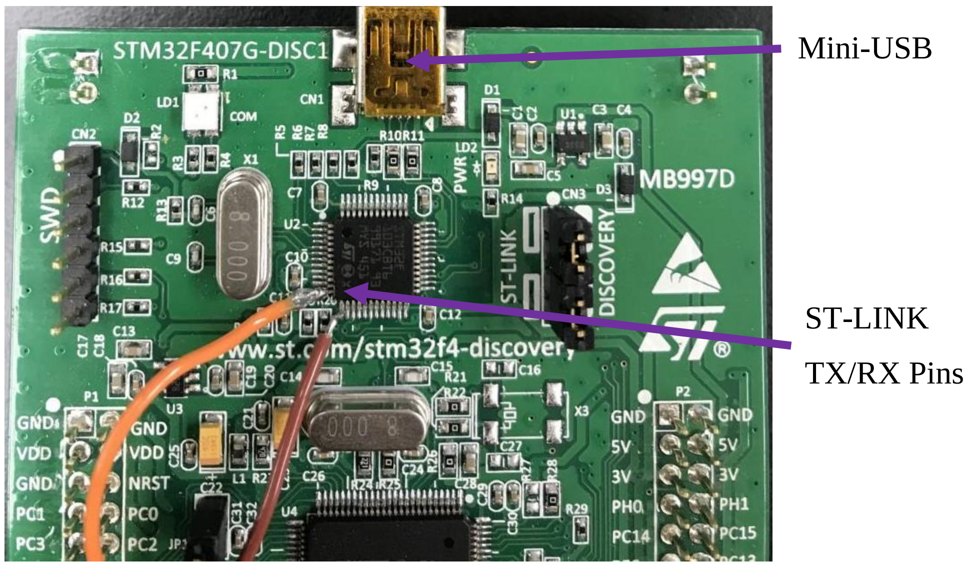


Figure 2.3 ST-LINK VCP Modification

The benefit of this modification is the simple interface. Only one USB cable is required to exchange data between the Discovery board and host PC. The same connection can be used to flash the firmware to the Discovery board as well.

This option is the simplest setup out of the three, but the data exchange rate is significantly throttled by the hardware. Data acquisition can be done but the speed at which this port is able to read and write data is too slow to support the PIL and HIL applications. Thus, making the first option not feasible for MBD purposes. To mitigate the physical limitation with the Discovery board, additional hardware is required moving forward.

2.3 FTDI Adapter

Future Technology Devices International (FTDI) is a semiconductor device company specializing in Universal Serial Bus technology [13]. It develops, manufactures, and supports devices and their related cables and software drivers for converting RS-232 or TTL serial transmissions to and from USB signals, in order to provide support for legacy devices with modern computer interfaces.

The second option to achieve the required communication rates was to create a new communication channel from the onboard USART TX/RX pins to a USB port on the host computer. The USART pins are TTL level and the FTDI adapter is used to convert them to be compatible with the host PC’s USB port. The hardware setup of the USART interface using FTDI converter is shown in Figure 2.4 below. This allows serial port communication between the microcontroller and host PC while running the SIL and PIL applications.

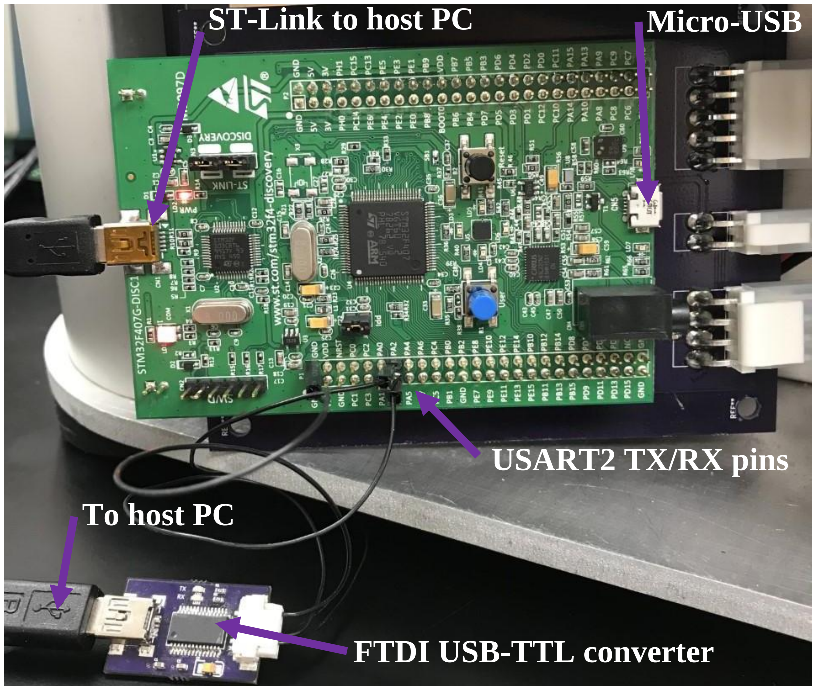


Figure 2.4 SIL and PIL Hardware Setup

The second option noticeably improves the data exchange rate. This is achieved by using a hardware with better data transmission capability. According to FTDI chip’s specification [14], the maximum baud rate achievable is 3M baud. This can be beneficial in theory, but the maximum speed is limited by MATLAB and Simulink software which has been tested to be below 1M (921600) baud. However, the second option’s data rate is still not sufficient for the HIL verification.

In contrast to the SIL and PIL, the HIL will accomplish data acquisition using the third option: USB On-The-Go Full-Speed (USB OTG FS). The implementation of this USB protocol further improves the data transmission rate for HIL. This option has the best data transfer rate when compared to the other two. Unfortunately, USB OTG FS cannot be utilized during SIL and PIL due to compatibility issues with Simulink. Simulink's proprietary code does not allow for a third-party patch to use different protocols. Only UART is supported for SIL and PIL. The second option of FTDI board will suffice for the operation of SIL and PIL using MotorLab. The USB OTG FS will be implemented using the micro-USB port on the Discovery board. The micro-USB port is on the opposite side of the board to the mini-USB as shown in Figure 2.4 above. The detailed steps on setting up, running the various simulations, and verifications with plots will be covered in Chapter 4, 5, and 6.

2.4 Motor, Encoder, and Amplifier

In this section, the characteristics of the motor, encoder and amplifier will be presented. In the next section the mathematical model of the system will be developed for position and speed control applications.

The motor is a three-phase DC brushless motor. It has a maximum speed of 4000 RPM with the 24 Volt power supply. The detailed specifications from the manufacturer of the motor, excerpted from Appendix B, are shown in Table 2.1 below.

Table 2.1 Motor Dynamic Specs from the Manufacturer

LA052-040E Motor Dynamic Specs from Shinano Kenshi		
	Units	**Value**
Rated Power	W	40
Rated Voltage	V_{DC}	24
Rated Speed	RPM	3,000
Rated Torque	$N \cdot cm$	12.7
	$kgf \cdot cm$	1.3
Rated Current	A	2.5
Torque Constant	$\frac{N \cdot cm}{A}$	5.0
	$\frac{kgf \cdot cm}{A}$	0.51
Back EMF Constant	$\frac{V}{kRPM}$	5.2
Phase Resistance	Ω	1.18
Phase Inductance	mH	4.4
Instantaneous Peak Torque	$N \cdot cm$	38.2
Max Speed	RPM	5,000
Rotor Inertia	$g \cdot cm^2$	110
Power Rate	$\frac{kW}{s}$	1.48
Mechanical Time Constant	ms	5.2
Electrical Time Constant	ms	3.7
Mass	kg	0.6

The motor position is measured by the incremental encoder attached directly to the motor shaft. The encoder is a two-channel encoder with a resolution of $0.225\frac{deg}{count}$ which is 1600 counts in a full revolution. The encoder signals are attached to a timer counter on the Discovery board. Motor position is measured by counting pulses from the encoder. The two channels provide indication of the direction of rotation so that encoder count is incremented or decremented depending on the direction of rotation. The measured position can be numerically differentiated in the software on the Discovery board to approximate the motor speed.

The position or speed of motor are controlled by the amplifier and microcontroller. The amplifier has a current control circuit that outputs a current to the motor windings proportional to the input command. This is known as a torque-controlled motor since the magnetic torque is proportional to the current in the windings. The command input to the amplifier is an analog signal. The microcontroller provides this signal by means of a ±3 Volt analog output connected to the amplifier. The microcontroller uses a Digital-to-Analog Converter (DAC) to achieve this output. By controlling this analog voltage signal, the current inside the motor can be controlled. The amplifier has a pin connected to the microcontroller's Analog-to-Digital Converter (ADC) where the current output to the motor is measured. This signal is not utilized in the microcontroller's control loop, but it is recorded for analysis purposes. The current command to the motor can be implied by scaling the microcontroller's analog voltage command with $1\,\frac{Amp}{Volt}$. Therefore, the unit of proportional gain in the position controller is $\frac{Volt}{deg}$. Similarly, the proportional gain in the velocity controller is $\frac{Volt}{RPM}$. The scaling of the amplifier current command and the encoder will be used in the next chapter to determine the scaling of the controller gains.

Chapter 3 - Physical System Modeling

This chapter will describe the parameters of the current MotorLab device and its electro-mechanical system characteristics. It will be used to model the system. This system model will be used in the first three phases of MBD: MIL, SIL, and PIL.

3.1 MotorLab System Block Diagram

Figure 3.1 below shows the schematic representation of MotorLab system in a closed-loop position or speed control configuration.

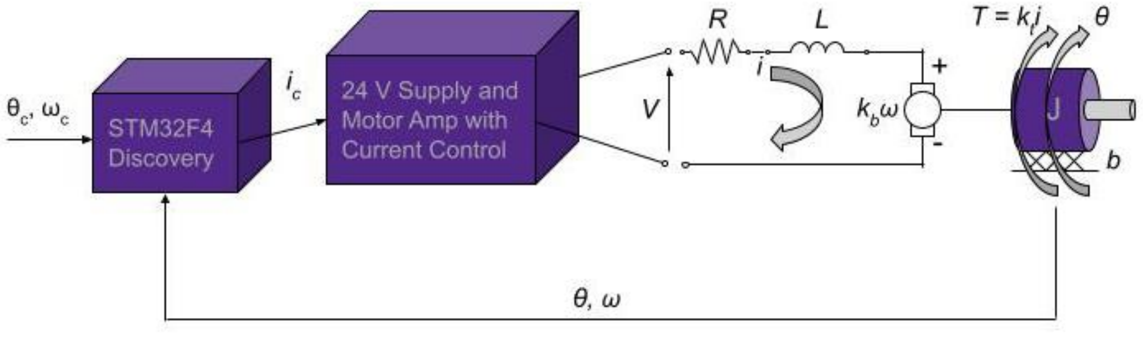


Figure 3.1 MotorLab Schematic Diagram

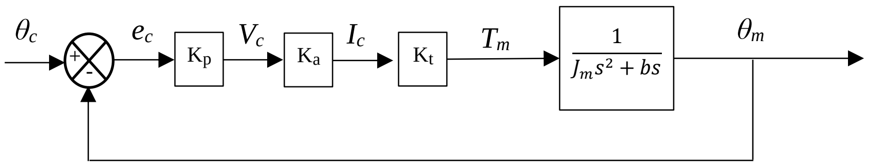


Figure 3.2 Proportional Position Control Block Diagram

For convenience, the desired motor command position, θ_c, will be represented in degrees. The measured position of the motor, θ_m, will also be in degrees as will the command error, e_c. K_p is the proportional gain of the controller ($\frac{Volt}{deg}$). The input signal to the amplifier, V_c, is the analog output from the microcontroller's DAC converter with a range of ±3 Volts. Since the amplifier is supplied with 24 Volts, the current input to the motor windings is changed with varying the duty cycle with ±3 Volts. Also known as Pulse Width Modulation (PWM), the average power with a fixed voltage delivered to the motor is varied by modulating the time duration of a pulse described by duty cycle. This drives the motor to change either its position or speed with current control. K_a is the amplifier gain and is denoted as $1\,\frac{Amp}{Volt}$. This amplifier gain is used to scale the voltage output into the current command, I_c, delivered to the motor. With the microcontroller's peak voltage of 3 Volts, the maximum current command is 3 Amps. As a note, the amplifier is mathematically assumed to be '1' meaning it is "fast" with no loss in the controller output. When a current command feeds into the motor windings, the torque produced by the brushless motor is proportional to the current input and is denoted by K_t and has a unit of $\frac{N\cdot m}{A}$. Then the applied torque drives the motor to the commanded position while the feedback loop provides the encoder reading of the motor shaft position, θ_m.

3.2 MotorLab System Parameters

The introductory lab exercises are designed for students to identify system parameters experimentally. Some hardware specific constants require further reading of the manufacturer's reference manuals and hardware specification data sheets. The most important parameters for modeling the dynamic system are in Table 3.1, excerpted from Appendix B. The numeric constants for the system model used in this thesis are gathered from it. The Table 3.1 below also contains the motor constants which the students identify during the second lab experiment.

Table 3.1 MotorLab System Parameters

	Value	Description
k_t	$0.05\frac{N\cdot m}{A}$	Motor Torque Constant
b	$3e^{-5} N\cdot m\cdot s$	Viscous Friction Coefficient
J_m	$1.29e^{-5} kg\cdot m^2$	Total Rotor Inertia
ω_{cf}	$300\frac{rad}{s}$	Speed Filter Cutoff Frequency

3.3 MotorLab Model Development

The motor shown in Figure 3.2 above can be modeled as a continuous time transfer function. The total inertia of the motor is given below:

$$J_m = J_1 + J_2$$

Where J_1 is the rotor inertia from Table 3.1 above and J_2 is the inertia of the double shaft collar which is $19 g\cdot cm^2$ taken from Appendix B. According to the Newton's law, the moment of inertia times the angular acceleration is equal to the sum of the torques which consist of the motor torque T_m and the friction torque:

$$J_m\dot{\omega} + b\omega(t) = T_m(t) \tag{3.1}$$

The motor torque is equal to the motor torque constant from Table 3.1 times the motor current:

$$T_m(t) = k_t i(t) \tag{3.2}$$

Now equating (3.1) and (3.2) then taking the Laplace transform to obtain the following:

$$J_m(s\omega(s) - \omega_0) + b\omega(s) = k_t i(s) \tag{3.3}$$

The time domain solution of (3.3) can be found for some arbitrary initial condition ω_0 and no input by setting the input current i to zero and solving for the speed:

$$J_m(s\omega(s) - \omega_0) + b\omega(s) = k_t i(s) = 0$$

$$(J_m s + b)\omega(s) = J_m\omega_0$$

$$\omega(s) = \frac{J_m\omega_0}{J_m s + b}$$

$$\omega(s) = \frac{\omega_0}{s + \frac{b}{J_m}}$$

Take the inverse Laplace transform to obtain the following exponential decay model to express the initial condition motor response:

$$L^{-1}\{\omega(s)\} = \omega(t) = \omega_0 e^{-\frac{b}{J_m}t}$$

To model the speed output from the motor using the current control, the following transfer function has been obtained by rearranging Equation (3.3):

$$\frac{\omega(s)}{i(s)} = \frac{k_t}{J_m s + b} \tag{3.4}$$

Since the position of the motor is the integral of the velocity, the position output from the motor is obtained by adding an integrator to Equation (3.4):

$$\frac{\theta(s)}{i(s)} = \frac{\omega(s)}{i(s)} \cdot \frac{1}{s} = \frac{k_t}{J_m s^2 + bs} \tag{3.5}$$

Equation (3.4) is the speed response model and Equation (3.5) is the position response model in MBD, specifically for MIL, SIL, and PIL.

Motor position can be measured directly with the incremental encoder. There is no device on the MotorLab system to measure its motor speed. To be able to operate as a speed control system, a software differentiation of the motor position is implemented in the microcontroller. To reduce the noise induced on the speed signal by the discrete nature of the encoder, a low-pass filter is implemented with the differentiator.

For the implementation of HIL, a block must be added to create the speed signal from the measured position signal. This will perform the same functionality as the derivative and filter in the original MotorLab firmware. In the MotorLab firmware the filter was specifically implemented with a lower frequency bandwidth (300Hz) that could be used to teach the students about higher-order dynamic effects in control systems. The intention of the MBD system being developed in this thesis is to retain the functionality of the original MotorLab system, so the block that is added must be created in such a way that the lower bandwidth characteristic of the original system is retained. So, the block that is created must include not only the derivative but also the filter characteristics. The second order continuous time transfer function for the filter with a derivative can be described as the following:

$$\frac{\omega_f(s)}{\theta(s)} = \frac{{\omega_{cf}}^2 s}{s^2 + 2\zeta\omega_{cf}s + {\omega_{cf}}^2} \tag{3.6}$$

Where $\omega_f(s)$ is the filtered speed output, $\theta(s)$ is the measured encoder position input, and ω_{cf} is the cutoff frequency to filter out large spikes in the speed approximation. Also note the free s in the numerator for differentiating the position. The natural frequency of the filter, which is the cutoff frequency, is chosen based on the 10x rule of thumb. This allows the system to ignore the effects of the high frequency dynamics from the filter itself. It is high enough to acquire clean signal from the motor but also low enough to help students realize this critical concept. When the students increase controller gains pushing the magnitude of this closed loop system closer towards the magnitude of the filter's open loop poles, or its cutoff frequency, it is expected to affect the system performance. The direct system outputs with the high frequency dynamics sometimes are not possible to predict or are complex to model. The effect of high frequency dynamics will be demonstrated later in Chapter 6 using both the speed and position responses of motor plant with Equation (3.4) and (3.5) above.

The block for implementing the speed filter is developed as a function in C code. The continuous time transfer function must be discretized. Discretization is accomplished using Tustin's method by setting

$$s = \frac{2}{T}\frac{z-1}{z+1}$$

Discretizing (3.6) symbolically results in the following expression:

$$\frac{\omega(z)}{\theta(z)} = \frac{2T{w_n}^2(z^2-1)}{T^2{w_n}^2z^2 + 4T\zeta w_n z^2 + 4z^2 + 2T^2{w_n}^2 z - 8z + T^2{w_n}^2 - 4T\zeta w_n + 4}$$

With the common z terms factored out in the denominator:

$$\frac{\omega(z)}{\theta(z)} = \frac{2T{w_n}^2(z^2-1)}{(T^2{w_n}^2 + 4T\zeta w_n + 4)z^2 + (2T^2{w_n}^2 - 8)z + T^2{w_n}^2 - 4T\zeta w_n + 4} \tag{3.7}$$

Then the following coefficient can be obtained to simplify (3.7):

$$(T^2{w_n}^2 + 4T\zeta w_n + 4)z^2 \tag{3.8}$$

Finally, dividing (3.7) with (3.8) above to find the filter coefficients:

$$\frac{\omega(z)}{\theta(z)} = \frac{\frac{2T{w_n}^2}{T^2{w_n}^2 + 4T\zeta w_n + 4}z^{-2} - \frac{2T{w_n}^2}{T^2{w_n}^2 + 4T\zeta w_n + 4}}{1 + \frac{2T^2{w_n}^2 - 8}{T^2{w_n}^2 + 4T\zeta w_n + 4}z^{-1} + \frac{T^2{w_n}^2 - 4T\zeta w_n + 4}{T^2{w_n}^2 + 4T\zeta w_n + 4}z^{-2}} \tag{3.9}$$

Implementing (3.9) in C code is included in Listing 3.1 below:

Listing 3.1 Low Pass Filter with a Derivative to get Velocity from Position Measurement

```c
void initLPF(float sampleT, float cutoffFreq, float dampingRatio) {
    float wn   = 2*3.14159f*cutoffFreq;
    float T    = sampleT;
    float zeta = dampingRatio;

    //set filter coefficients
    b0 = (T*T*wn*wn + 4*zeta*T*wn + 4);
    b1 = (2*T*T*wn*wn - 8)/b0;
    b2 = (T*T*wn*wn - 4*zeta*T*wn + 4)/b0;
    a0 = 2*T*wn*wn/b0;
    a1 = 0/b0;
    a2 = (-2*T*wn*wn)/b0;

    //init previous values
    Vk_1 = Vk_2 = Pk_1 = Pk_2 = 0;
}

float getVelocity(float pos) {
    float vel;

    vel = a0*pos + a1*Pk_1 + a2*Pk_2 - b1*Vk_1 - b2*Vk_2;

    // update values
    Pk_2=Pk_1; Pk_1=pos; Vk_2=Vk_1; Vk_1=vel;

    return vel;
}
```

Note the coefficient a1 is zero because the numerator of (3.9) does not have a z^{-1} term. The function getVelocity will take position measurement as an input argument and return the filtered velocity approximation. The verification of speed control using the C code above will be shown in Chapter 6 along with other response data comparisons using the HIL method.

Chapter 4 - Software

4.1 MathWorks

MATLAB has become a standard high-level modeling and analysis software for both the industry and academia. Its capabilities make the software very user friendly when coupled with Simulink. Simulink provides graphical user interface for simulation and code generation based on block diagrams. MathWorks Toolboxes such as Simulink Coder, MATLAB Coder, and Embedded Coder support automated code generation with optimization available [15]. In combination, they can use Simulink diagrams to produce firmware for specific microcontrollers. Similarly, hardware support packages provide microcontroller specific add-ons for Simulink and MATLAB. Once installed, these packages can configure Simulink diagrams to generate code for the targeted microcontroller to communicate with the host PC during simulation runtime. This is useful during PIL where the microcontroller computes controller output and the host PC computes the system response. Simulink labels this proprietary feature as External mode. It provides graphical representation of the simulation response and allows the user to pause the simulation and change simulation parameters using the host PC. Although hardware support packages are free for users with adequate MathWorks licenses, for the typical educational license there are limitations. The educational license provides some support for the popular Arduino processors but no support for the more capable Discovery board used in this thesis. These limitations and workarounds will be evaluated later in Chapter 5.

The following is a summary of all the necessary programs, toolboxes, and additional hardware support packages from MathWorks.

MathWorks base software with licensing requirement:

- Simulink and MATLAB

Toolboxes with licensing requirement:

- Simulink Coder and MATLAB Coder
- Embedded Coder
- Control System Toolbox

Required hardware support packages available for free:

- ARM Cortex-M Support from Embedded Coder
- ST Discovery Board Support from Embedded Coder

4.2 STMicroelectronics

In order to support MBD, STMicroelectronics provides software tools that integrate into MATLAB and Simulink to access the features of their microcontrollers. The following programs must be downloaded from the vendor's webpage:

- STM32-MAT/TARGET, STM32CubeIDE and STM32CubeMX

STM32-MAT/TARGET is a particular software package developed by STMicroelectronics to extend the target support for Simulink and MATLAB. This software comes with prebuilt peripheral driver blocks including but not limited to General Purpose Input Output (GPIOs) and Timers.

STM32CubeIDE is an Integrated Development Environment (IDE) based on Eclipse CDT [16]. This open-source IDE comes with support for the Arm toolchain as well as various helpful debugging tools such as Serial Wire Debug (SWD) interface based on the Gnu Debugger (GDB).

STM32CubeMX is a GUI based embedded C code project initialization generator which incorporates Hardware Abstract Layer (HAL) drivers into the base project [17]. The generated project file has an extension of .ioc. The generated project file then can be imported into the workspace of STM32CubeIDE or any other supported IDE such as the IAR Embedded Workbench or Keil MDK. STM32CubeIDE was chosen over these IDEs due to the fact that it is not a subscription-based product.

Detailed description of the workflow for each of the simulations will be included in the following chapter.

Chapter 5 - Code Generation with MBD

The original MotorLab microcontroller's firmware is a sophisticated custom Real Time Operating System (RTOS), a modified version of an OS developed for used in Unmanned Aerial System (UAS) flight controller. The system supports the level of complexity and computational bandwidth required for flight control systems so it will have more than adequate capacity for the introductory controls lab exercises. To allow students to continue to use a well performing controller, the MBD generated firmware must be required to run control loop rates at 10kHz. All the other peripheral tasks such as data acquisition must be intricately developed to not interfere with the main control loop rate. Available methods for verifying the system performance will be explored later in this chapter using HIL and their respective performance and limitations will be discussed in Chapter 6.

Figure 5.1 and Figure 5.2 captures the major differences between the MIL, SIL, PIL, and HIL stages of MBD development for MotorLab. The four stages will be described in the following subsections. The MIL and SIL are simulations performed only on the host PC and do not require microcontroller. The PIL and HIL stages require both the microcontroller and sometimes the actual physical system depending on the level of HIL verification. In advanced industry testing environment, especially in aerospace where HIL testing sometimes can be difficult, additional industrial grade hardware and software are used to test the plant response in real-time simulation. These real-time simulation solutions are provided from a partnership between MathWorks and Speedgoat. Speedgoat is the exclusive hardware manufacturer for Simulink Real-Time toolbox and supports much higher data bandwidth for detailed simulation in real-time [18].

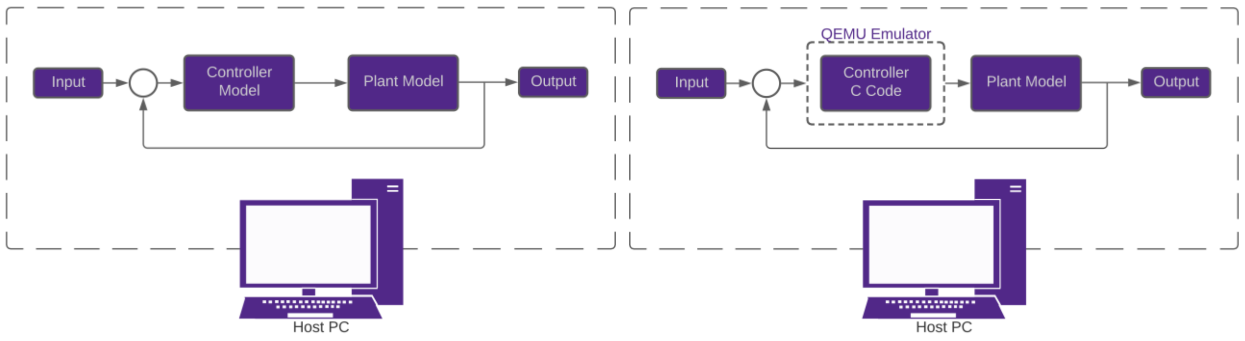


Figure 5.1 MIL and SIL Verification

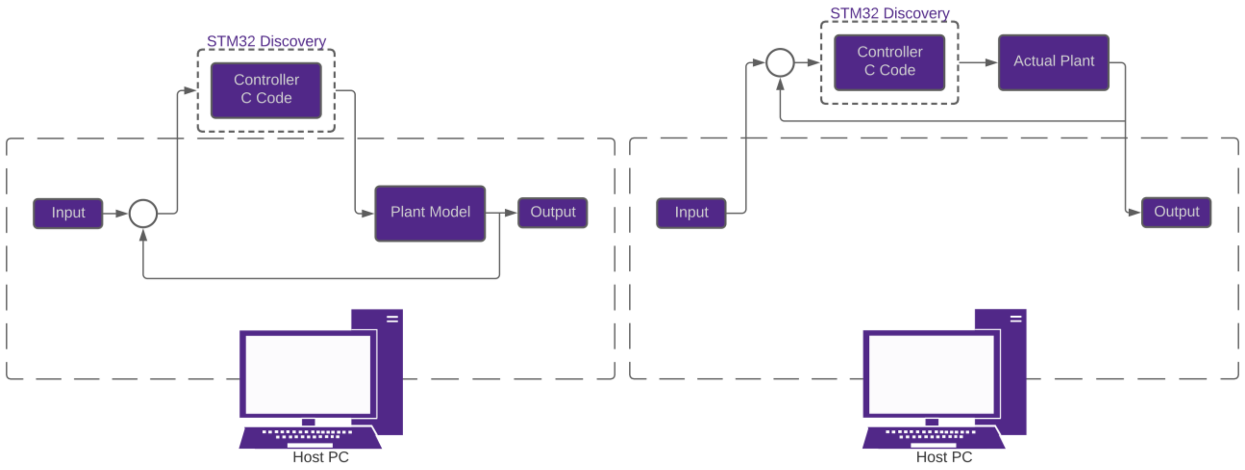


Figure 5.2 PIL and HIL Verification

5.1 Model-In-Loop

The very first step in MBD is the realization of the plant model to design a controller. Figure 5.3 below shows the Simulink model containing two identical system models. It is modified from the lab exercise #7, where students are taught to construct a system similar to the MIL phase. The block diagrams in Chapter 3 are used as components of this model. The closed loop system on top uses a continuous PID controller and the other uses a discrete PID controller. The two feedback gains named blow and bhigh denote the difference in friction coefficient values at high and low speed of the motor respectively. This will allow for the model to use a higher coefficient of friction at low speed and lower coefficient at high speed. Once the plant model is created, the controller model can be developed to verify the controllability of the plant. In the MIL simulation, only the top closed loop system model is required. The continuous PID controller will drive the simulated system response. This step is to make sure the controller logic can in fact control the plant model within the desired performance criteria.

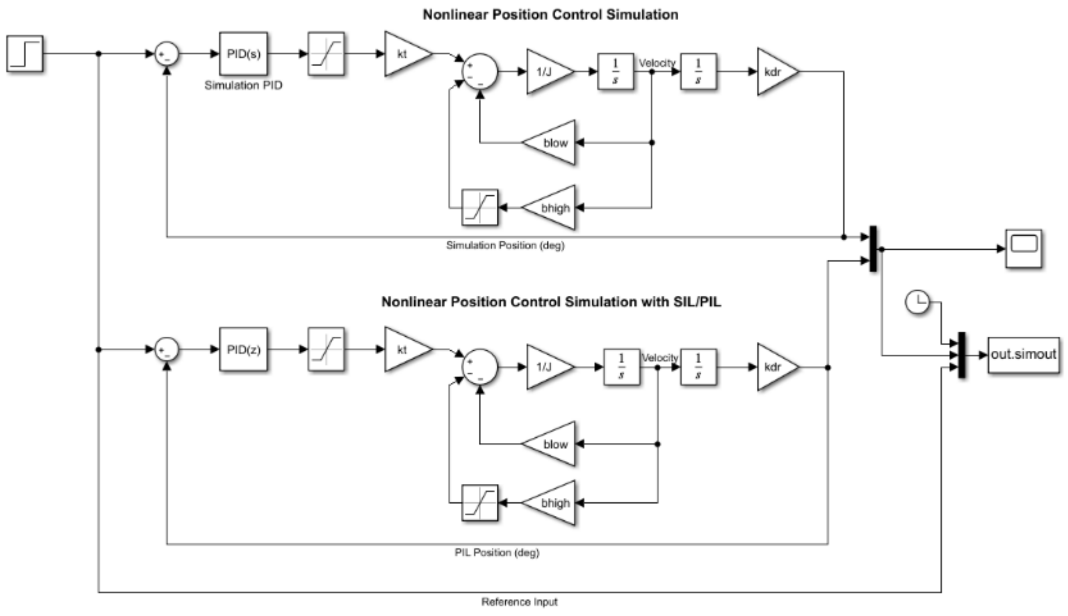
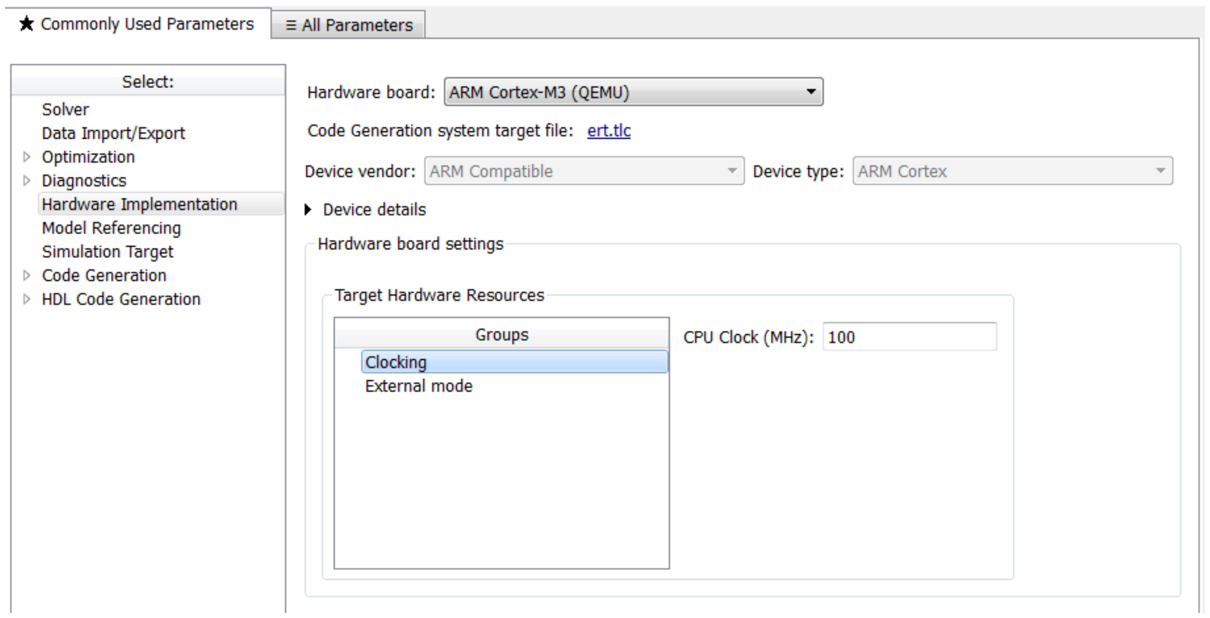

Nonlinear Position Control Simulation
PID(s)
Simulation PID
kt
1/J
Velocity
kdr
blow
bhigh
Simulation Position (deg)
Nonlinear Position Control Simulation with SIL/PIL
PID(z)
out.simout
PIL Position (deg)
Reference Input

Figure 5.3 Simulink Block Diagram for MIL, SIL, and PIL

The resulting Simulink data output will be exported back into the MATLAB workspace (to out.simout) using the export block as shown above. This workspace variable will contain the time series variable and position reading at each time step and can be used to plot the result. It is important to record simulation results to compare responses between the other simulations, as well as the final HIL verification stage. The resulting plots should show very similar responses between the continuous and discrete controllers driving the same plant model. The comparison plots are included in Chapter 6 and note that this step does not require any hardware and can be done with only the host PC running MathWorks products.

5.2 Software-In-Loop

After verifying the controller and plant model in the MIL step, the next step is to configure the Simulink diagram to run SIL. The difference between MIL and SIL is that the former uses the host PC to simulate everything, and the latter uses a microcontroller emulator to simulate the controller hardware. Figure 5.4 below shows the configuration required for the model to use ARM Cortex-M3 emulator. Note that the microcontroller on the Discovery board is an ARM Cortex-M4 based controller, but the emulator for M4 is not yet available with the hardware implementation package.



Hardware board: ARM Cortex-M3 (QEMU)

Code Generation system target file: ert.tlc

Device vendor: ARM Compatible

Device type: ARM Cortex

CPU Clock (MHz): 100

Figure 5.4 SIL Hardware Configuration

After configuring the Simulink diagram to run in SIL mode, the next step is to generate the controller block that will contain the C code. As shown in Figure 5.5 below, right click on the discrete controller block and navigate to the hardware deployment option.

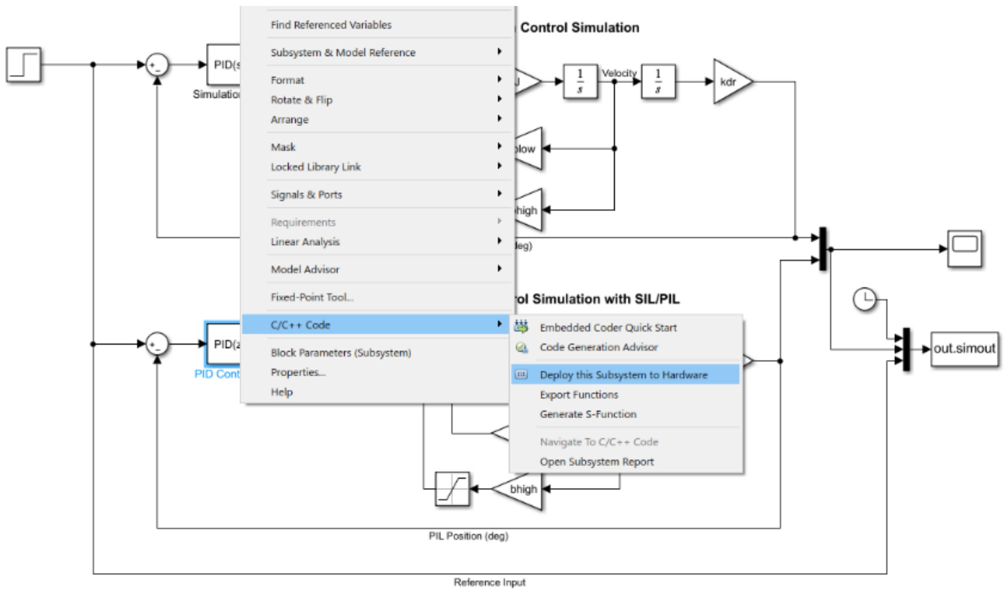
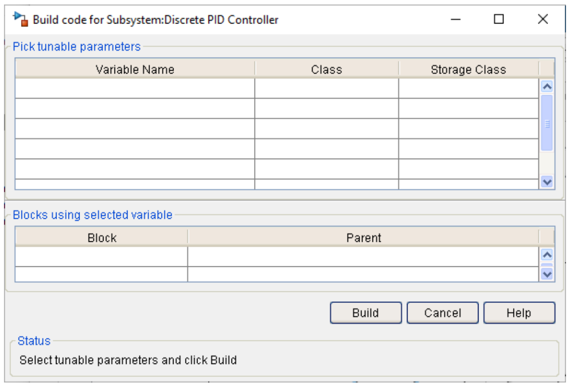


Figure 5.5 SIL/PIL Block Generation Dialog Box

After choosing the deployment option, a new dialog window will open as shown in Figure 5.6.


Build code for Subsystem:Discrete PID Controller
Pick tunable parameters
Blocks using selected variable
Build
Cancel
Help
Status

Variable Name	Class	Storage Class

Block	Parent

Select tunable parameters and click Build

Figure 5.6 Build Code Dialog Box

Click on the Build button and when the build process completes, a new Simulink diagram window opens with the C code controller block as shown in Figure 5.7. Replace the discrete controller block with this new SIL controller block and run the simulation. Similar to the previous stage, SIL does not require the actual hardware. Also, note the block is named PIL, but this is due to Simulink not differentiating the name between SIL and PIL steps. The Simulink diagram will run the controller based on the SIL configuration from Figure 5.4 above regardless of the block name.

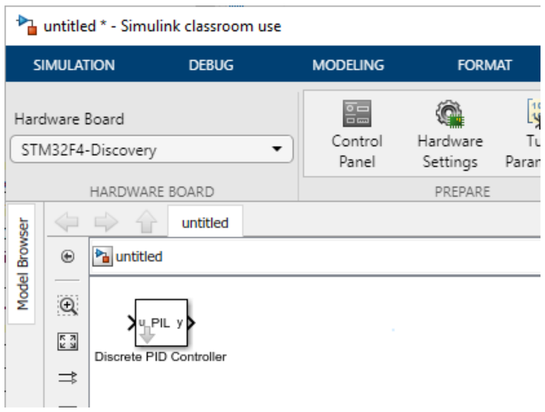


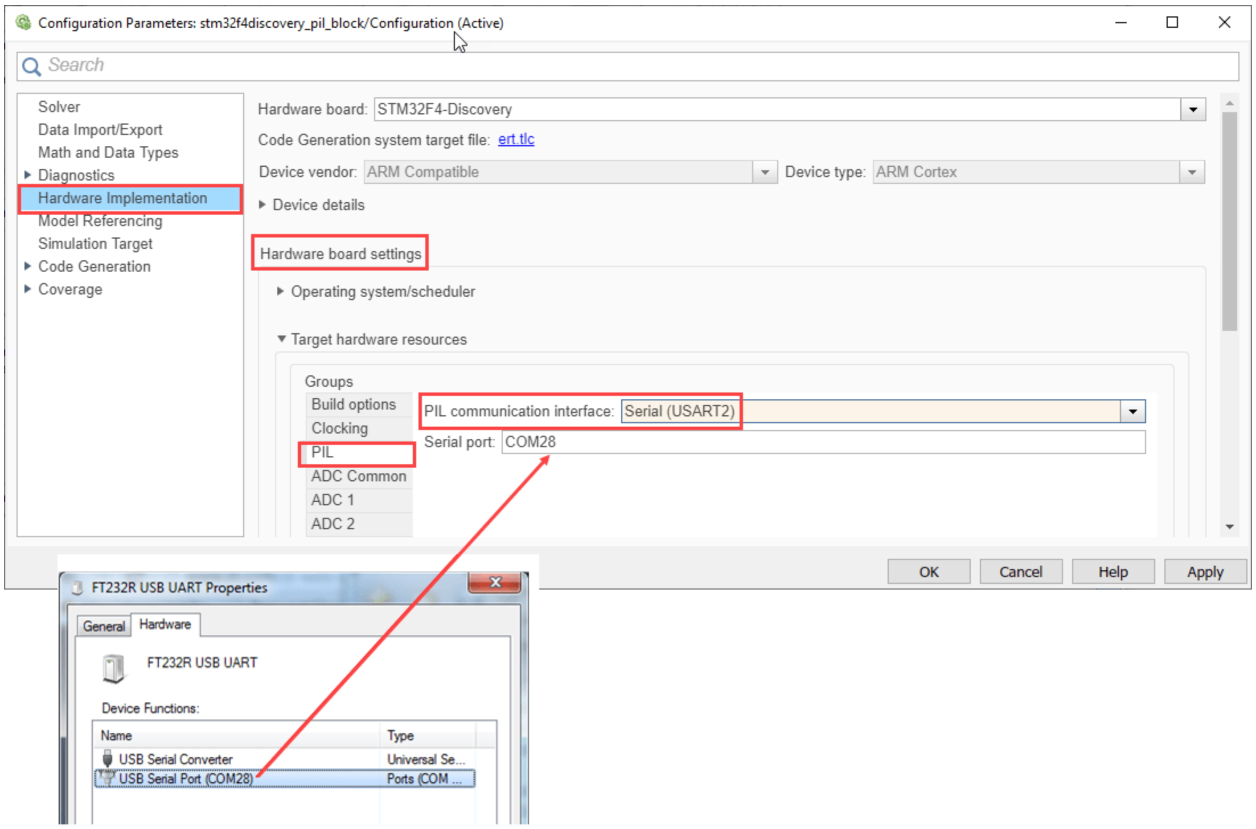
Figure 5.7 PIL Simulink Block

Once again, when the simulation completes make sure to save the simulation response and exported workspace variable for the analysis.

5.3 Processor-In-Loop

Running the simulation in PIL is similar to SIL from the previous subsection. To deploy PIL, configure the Simulink diagram to run the controller block outside of host PC with the hardware select dropdown menu. For this thesis, STM32F4-Discovery is chosen but other hardware boards are also available depending on which hardware support package is installed. Repeat the process from Figure 5.5 through Figure 5.7 and replace the SIL controller block with the new PIL block.

Unlike the SIL stage, it is critical at this point to setup the hardware to have correct communication between the host PC and microcontroller board. As described in Chapter 2, the FTDI cables are connected to the designated TX/RX GPIO pins which are configured as shown in Figure 5.8 from MathWorks webpage [19]. Note the COM port number is arbitrary and will depend on the port which the FTDI board is connected to.



Configuration Parameters: stm32f4discovery_pil_block/Configuration (Active)

Hardware board: STM32F4-Discovery

Code Generation system target file: ert.tlc

Device vendor: ARM Compatible

Device type: ARM Cortex

Hardware board settings

PIL communication interface: Serial (USART2)

Serial port: COM28

FT232R USB UART Properties

FT232R USB UART

Device Functions:

Name	Type
USB Serial Converter	Universal Se...
USB Serial Port (COM28)	Ports (COM ...

Figure 5.8 PIL Configuration for the Target Hardware

When the run button is clicked, Simulink will build, deploy, and run the simulation in series with the target microcontroller and manage the data exchange. Observe the exported data back in the MATLAB workspace and save the response data when the simulation terminates.

When the modified ST-LINK VCP is used instead of the FTDI solution as mentioned in Chapter 2, the ST-LINK will block until its serial communication task is completed, making the total simulation time extremely long and impractical. The use of FTDI is also recommended by MathWorks documentation.

5.4 Hardware-In-Loop

HIL is the final verification step of MBD development cycle. HIL may include the complete plant setup or be partially made up of simulation models. Factors such as safety, cost, time, and size may influence what physical devices may be included or simulated. Several repetition of HIL may be required with additional physical devices integrating into the system at each repetition. Following the HIL, further investigation and reiteration back to previous stages may become necessary as the real-world problems, such as the high frequency dynamics and nonlinearity, can start to occur. Analysis of the expected occurrence of high frequency dynamics from Chapter 3 are included in Chapter 6.

In the following subsections, limitations of the software used in the HIL development, specifically from Simulink and STM32-MAT/TARGET, are introduced and the method developed to mitigate the said limitation will be presented.

5.4.1 Simulink External Mode

External mode is a feature of the Simulink Coder package. External mode provides the user with real-time execution of I/O driver code blocks while exchanging parameter data between the host PC and the target microcontroller board. This feature uses a shared memory interface, enabling users to manipulate signals, such as PID gain values or step input value, during the simulation runtime. This can be helpful for tuning the controller while observing the system response. Below is a figure from MathWorks webpage describing External mode.

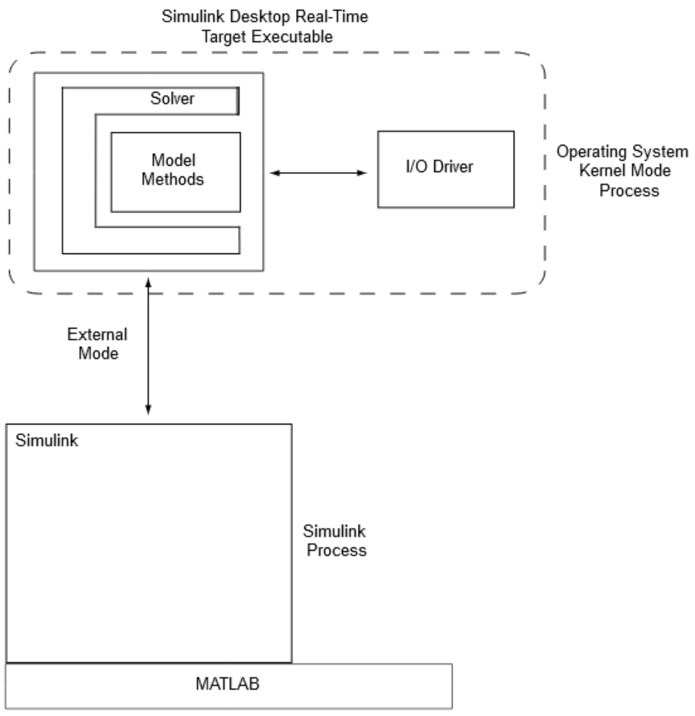

Simulink Desktop Real-Time Target Executable
Solver
Model Methods
I/O Driver
Operating System Kernel Mode Process
External Mode
Simulink
Simulink Process
MATLAB

Figure 5.9 Excerpt of External Mode from MathWorks Help Center

5.4.2 STM32CubeMX Configuration

The microcontroller on the Discovery board includes several peripherals and components that can be used in wide range of applications. For example, the Discovery board includes multiple counters, timers, registers, ADCs and DACs, and communications interfaces. Before the release of STM32CubeMX, all of the peripheral drivers had to be manually developed using their reference manual and their Application Programming Interface (API). To ease the burden of developers manually configuring each microcontroller board, the manufacturer developed STM32CubeMX that works with Simulink.

STM32CubeMX is an interactive software that supports Simulink as a base hardware setup tool. In this subsection, the detailed description of C code project initialization using the STM32CubeMX program is presented. With the help of STM32CubeMX, the hardware target and its peripherals can be configured as desired. Open STM32CubeMX then navigate to

ACCESS TO MCU SELECTOR under New Project and find a desired hardware target. Double clicking on the target selected will bring up Pinout & Configuration tab like Figure 5.10 below. If a dialog window asking about initialization of every peripheral pin appears, select no. This will prevent the project from including unnecessary initialization code and keep the project files streamlined.

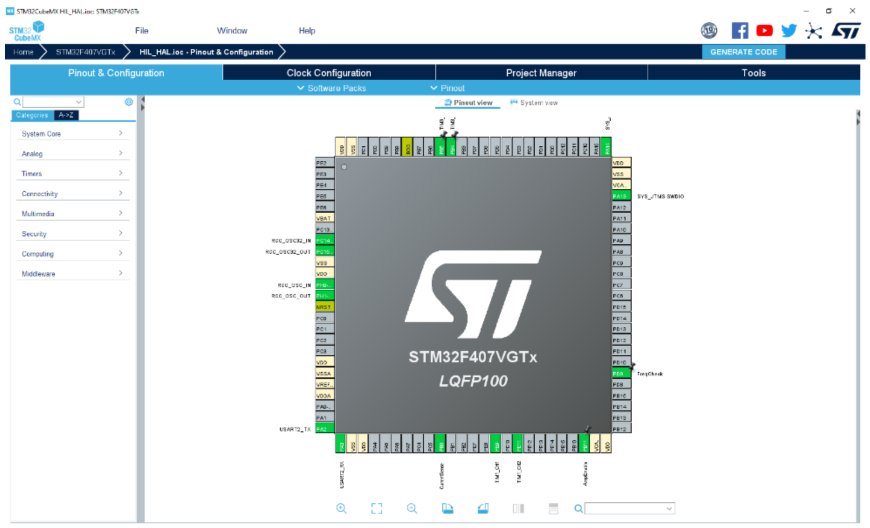


Figure 5.10 STM32CubeMX Pinout & Configuration Tab

The Pinout & Configuration tab will be used to setup required peripherals used for controlling the MotorLab device. Note the device category pane along the left of Figure 5.10. The first thing to configure is System Core. Within System Core, select the RCC section and set High Speed Clock to use Crystal/Ceramic Resonator. This will configure the clock to use the maximum possible clock frequency for driving peripherals. Other peripherals like Timer settings for generating PWM and reading encoder counts can be configured within this tab as well. Refer to Figure 5.10 above to see various GPIO pins and their names which reflect their respective functions. For example, TIM1 and TIM3 are used to drive PWM signals and count encoder angles respectively.

The next step is Clock Configuration. The specific parameter values for the clock configuration were found by studying the MotorLab device information files and the C-code file "system_stm32f4xx.c". This is the setup code that runs in the original MotorLab device which was created by Dr. Schinstock. The view of the complete Clock Configuration is shown in Figure 5.11. This will configure the microcontroller to use its maximum clock cycle for peripheral devices. This maximum clock cycle is 168MHz for the Discovery board.

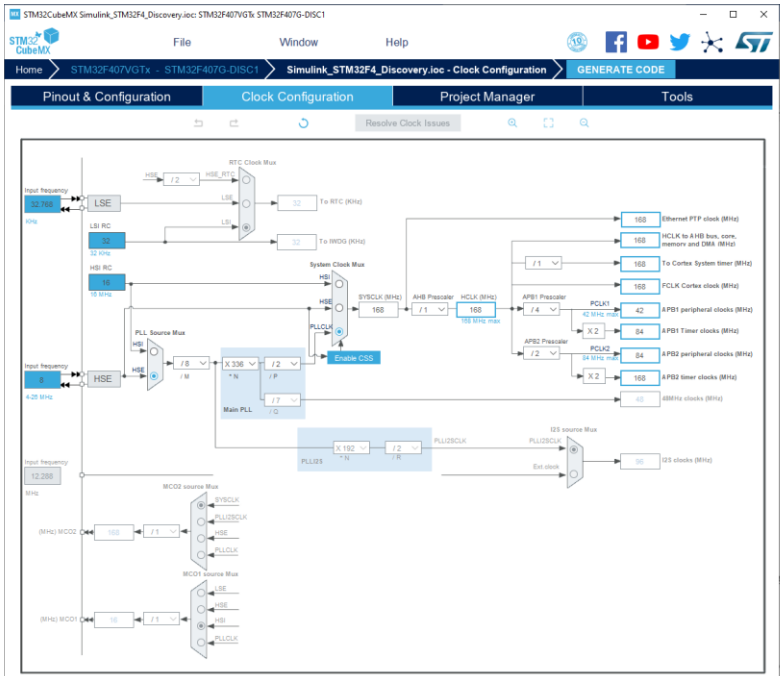
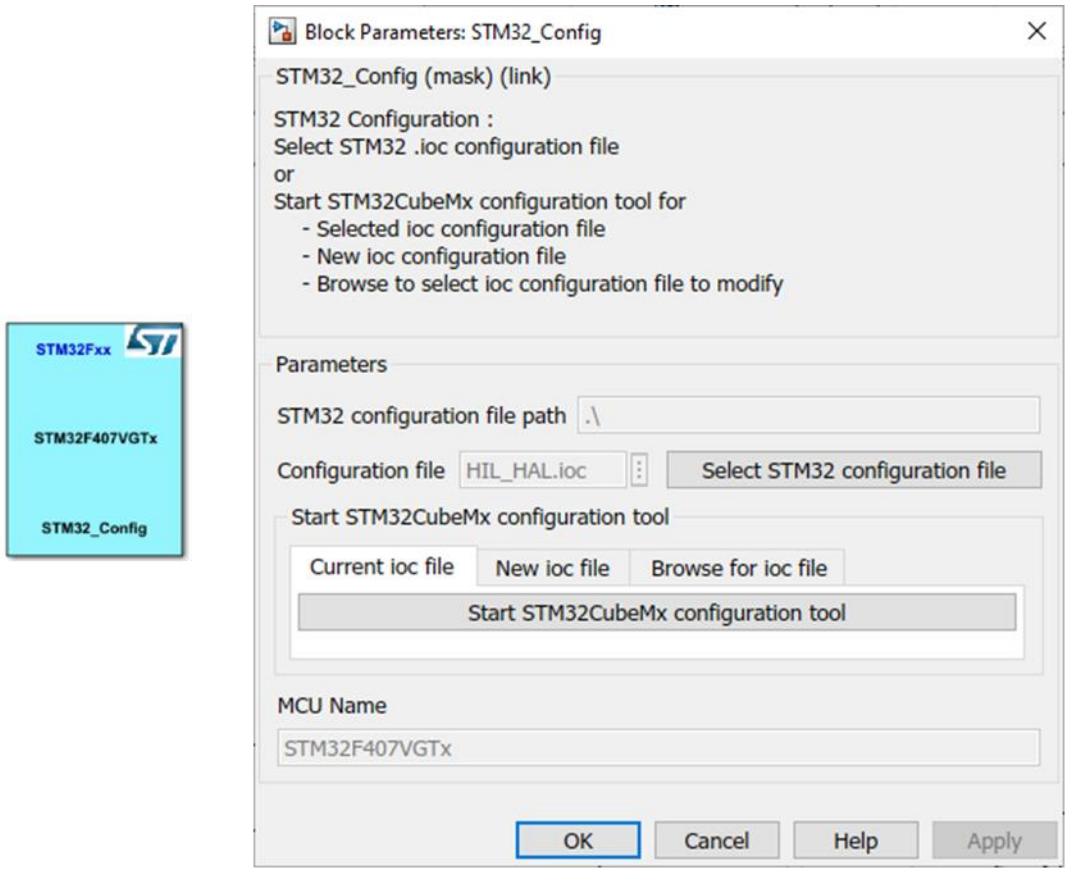


Figure 5.11 Project Clock Configuration

Once the configuration is complete and the parameters entered, select GENERATE CODE to autogenerate the initialization code for the selected peripheral devices. This will generate and export the project code (with the .ioc extension) to the IDE and toolchain specified under Project Manager tab. The firmware code that will be generated by Simulink will be appended to this base C project code.

5.4.3 STM32-MAT/TARGET Block Set

One of the limitations of Simulink and its External mode is the lack of target support for STM32 microcontrollers. The hardware support package provided by MathWorks includes only three microcontroller boards and lacks the ability to configure individual peripherals. To mitigate this issue, this thesis will make use of another support package from STM called STM32-MAT/TARGET. It will be used for the HIL verification. This support package provides the users with more advanced and customizable driver blocks. But the most beneficial feature from STM32-MAT/TARGET package is that it supports the integration of configuration code file generated by the STM32CubeMX in the previous subsection. In Figure 5.12 is the STM32 Configuration block and its parameter dialog box.



Block Parameters: STM32_Config

STM32_Config (mask) (link)

STM32 Configuration :
Select STM32 .ioc configuration file
or
Start STM32CubeMx configuration tool for
- Selected ioc configuration file
- New ioc configuration file
- Browse to select ioc configuration file to modify

Parameters

STM32 configuration file path: .\

Configuration file: HIL_HAL.ioc | Select STM32 configuration file

Start STM32CubeMx configuration tool

Current ioc file | New ioc file | Browse for ioc file

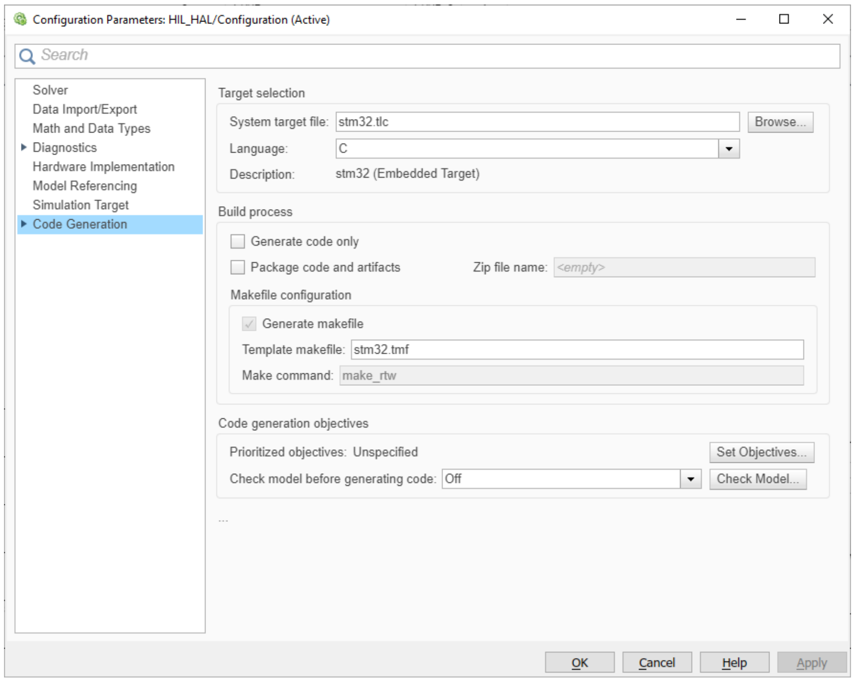
Start STM32CubeMx configuration tool

MCU Name: STM32F407VGTx

Figure 5.12 STM32 Configuration Block and Parameter Dialog Box

This block configures the Simulink and Simulink Coder to generate source files using the .ioc file specific to the target hardware. With the Simulink diagram using the STM32-MAT/TARGET configuration block, the development ecosystem now incorporates Simulink, STM32CubeMX, and STM32CubeIDE. This in essence extends the available target hardware from three boards that are included in the MathWorks' support package, to virtually every MCUs manufactured and supported by STM.

After setting up the hardware target using the block shown in Figure 5.12 above, the next step is to configure Simulink to use a different system target file. The Configuration Parameters dialog box within Simulink is shown below in Figure 5.13:


Configuration Parameters: HIL_HAL/Configuration (Active)

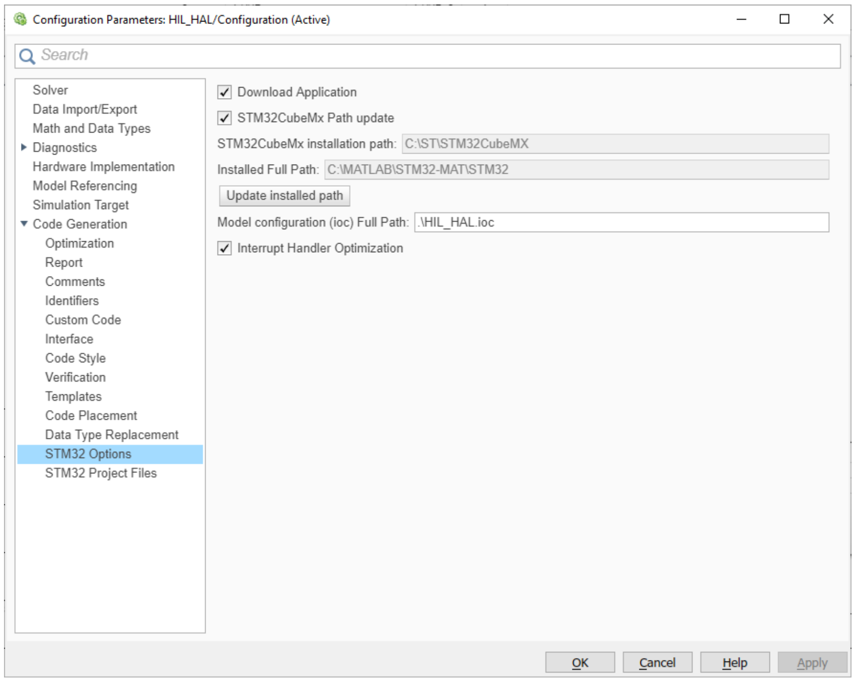
Target selection
System target file: stm32.tlc
Language: C
Description: stm32 (Embedded Target)

Build process
☐ Generate code only
☐ Package code and artifacts
Zip file name: <empty>
Makefile configuration
✓ Generate makefile
Template makefile: stm32.tmf
Make command: make_rtw

Code generation objectives
Prioritized objectives: Unspecified
Check model before generating code: Off

Figure 5.13 Simulink Target Selection using STM32-MAT/TARGET

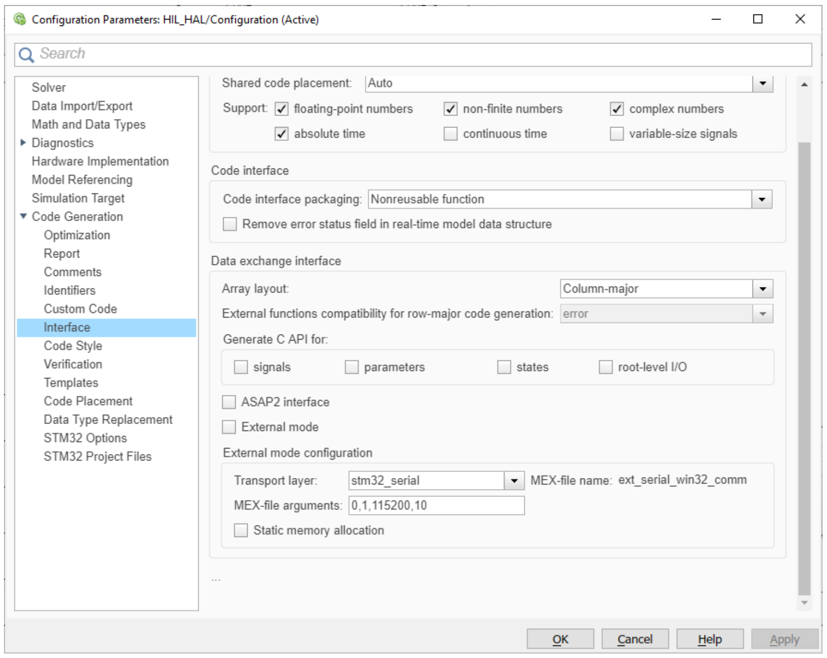
The selected system target file with the file extension of .tlc in the figure above is included in the STM32-MAT/TARGET package. This setup will instruct Simulink Coder to compile the block diagram and generate code using stm32.tlc to the target the hardware specified by the configuration block. After the system target file has been configured, the next step is to specify the installed path of STM32CubeMX. As shown in Figure 5.14 below, STM32CubeMx Path update box should be checked. This will search for a directory containing the software and autofill the empty installation path box.



☑ Download Application
☑ STM32CubeMx Path update
STM32CubeMx installation path: C:\ST\STM32CubeMX
Installed Full Path: C:\MATLAB\STM32-MAT\STM32
Model configuration (ioc) Full Path: .\HIL_HAL.ioc
☑ Interrupt Handler Optimization

Figure 5.14 STM32 Options for STM32CubeMX Installation Path

The last step of configuring the Simulink diagram for External mode is to set the verification interface method. Navigate to Interface under Code Generation within Configuration Parameters dialog box. Select External mode as highlighted in Figure 5.15 below. Also note that inside the Interface section, there are other options such as External mode configuration to setup communication port, but this will be ignored during the build as described in the following subsection as a bug.


Configuration Parameters: HIL_HAL/Configuration (Active)

Shared code placement: Auto

Support: ✓ floating-point numbers ✓ non-finite numbers ✓ complex numbers ✓ absolute time ☐ continuous time ☐ variable-size signals

Code interface

Code interface packaging: Nonreusable function

☐ Remove error status field in real-time model data structure

Data exchange interface

Array layout: Column-major

External functions compatibility for row-major code generation: error

Generate C API for: ☐ signals ☐ parameters ☐ states ☐ root-level I/O

☐ ASAP2 interface

☐ External mode

External mode configuration

Transport layer: stm32_serial MEX-file name: ext_serial_win32_comm

MEX-file arguments: 0,1,115200,10

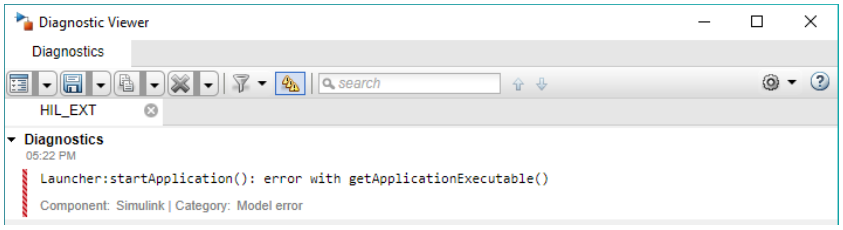
☐ Static memory allocation

Figure 5.15 Selection of Verification Interface

Now the Simulink diagram is configured to be verified using External mode along with the hardware in real-time as the HIL.

5.4.4 Limitations of STM32-MAT/TARGET

At the time of the development, an existing bug in the system prevents the program from launching STM32CubeIDE at the end of the code generation in External mode.


Diagnostics
05:22 PM

```
Launcher:startApplication(): error with getApplicationExecutable()
```

Component: Simulink | Category: Model error

Figure 5.16 The Diagnostics Window with the Error Message

To bypass this bug, the Build button under DEPLOYMENT should be used instead of Build, Deploy & Start as shown in Figure 5.17.

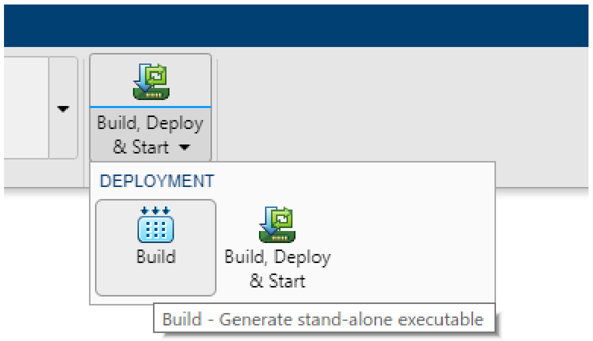
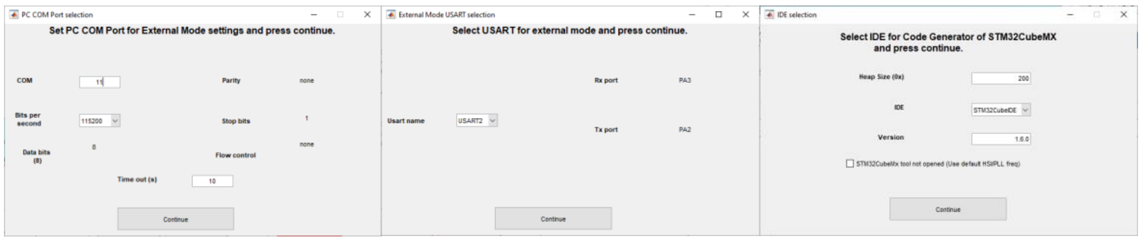


Figure 5.17 Build Button

Refer to Figure 5.18. When the build is initiated, a series of dialog windows will appear to configure the host PC COM port where the FTDI converter is connected and to select IDE. This will also override any values set within Figure 5.15.


PC COM Port selection

Set PC COM Port for External Mode settings and press continue.

COM	11	Parity	none
Bits per second	115200	Stop bits	1
Data bits (8)	8	Flow control	none
Time out (s)	10		

Continue

External Mode USART selection

Select USART for external mode and press continue.

Usart name	USART2	Rx port	PA3
		Tx port	PA2

Continue

IDE selection

Select IDE for Code Generator of STM32CubeMX and press continue.

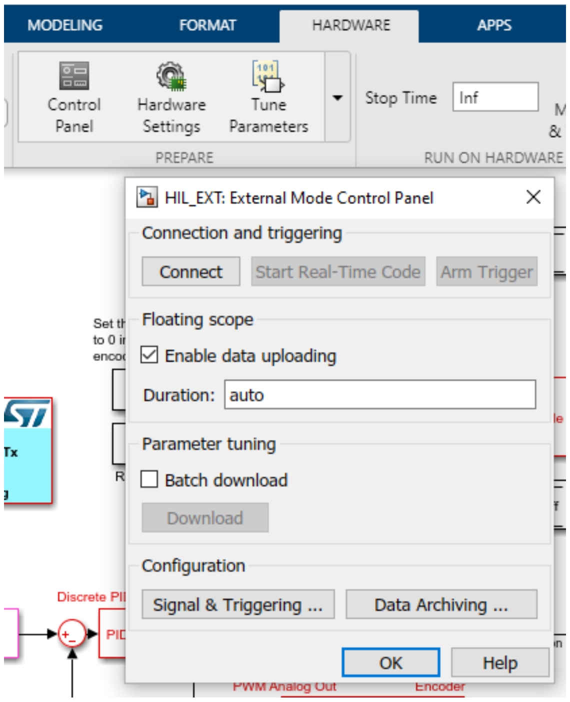
Heap Size (0x)	200
IDE	STM32CubeIDE
Version	1.6.0

☐ STM32CubeMx tool not opened (Use default HSI/PLL freq)

Continue

Figure 5.18 External Mode Dialog Windows

When the build completes, the generated project can be opened using the IDE selected with STM32CubeMX in the previous step. Run the debugger within the IDE, then open External Mode Control Panel from the Simulink diagram toolbar. While the debugger is running in the IDE, the hardware can be connected to the Simulink diagram using Connect button as shown in Figure 5.19.



Stop Time: Inf

HIL_EXT: External Mode Control Panel

Connection and triggering

Floating scope

[✓] Enable data uploading

Duration: auto

Parameter tuning

[ ] Batch download

Configuration

Figure 5.19 External Mode Control Panel

The method described in this subsection bypasses the error message shown in Figure 5.16 and connect to the hardware while communicating in real-time using External mode. The complete Simulink diagram for the motor control apparatus utilizing the STM32-MAT/TARGET, MATLAB, and Simulink block sets and detailed view of subsystems is shown in Figure 5.20 and Figure 5.21 respectively.

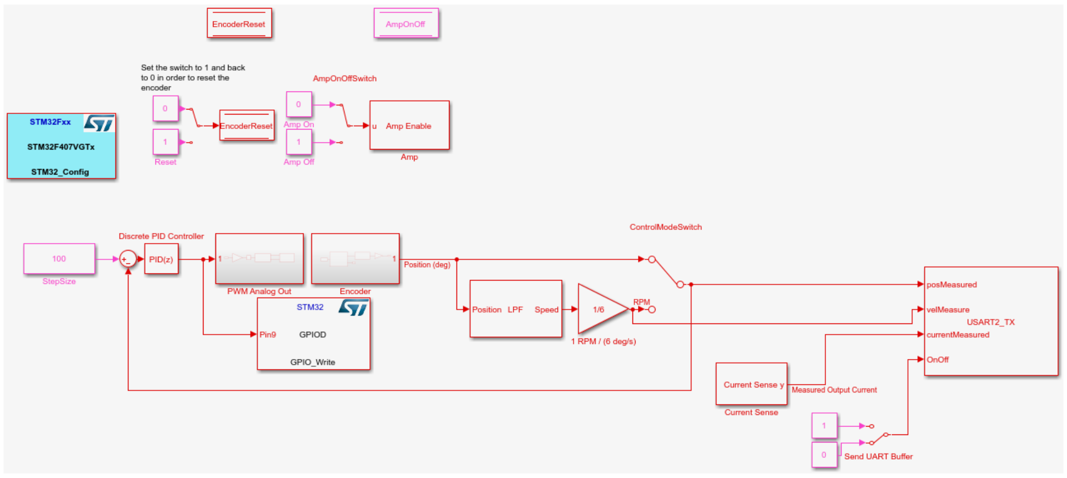


Figure 5.20 The Simulink Diagram of MotorLab

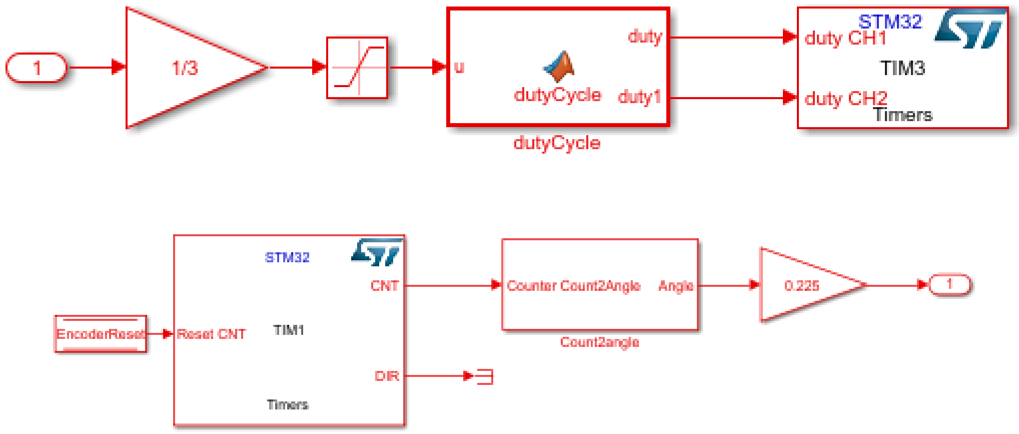


Figure 5.21 Timer Blocks for PWM and Encoder Subsystems

At this point of the development, the microcontroller can exchange data with the host PC to edit parameters of the motor controller to test and verify the system response. However, as soon as the data acquisition block is implemented within the diagram, running the simulation in External mode becomes impractical. The increase in required bandwidth for data acquisition is too much when added on top of the External mode's overhead. The low-cost microcontroller board cannot physically handle the required data transmission rate. For instance, when External mode is used to interact with the hardware in real-time while transferring velocity data back to the host PC, the control loop will deviate from its fixed loop frequency of 10kHz. This results in instability of the motor control. The physical setup of measurement is shown in Figure 5.22. The measured abnormal control loop frequency is shown in Figure 5.23.

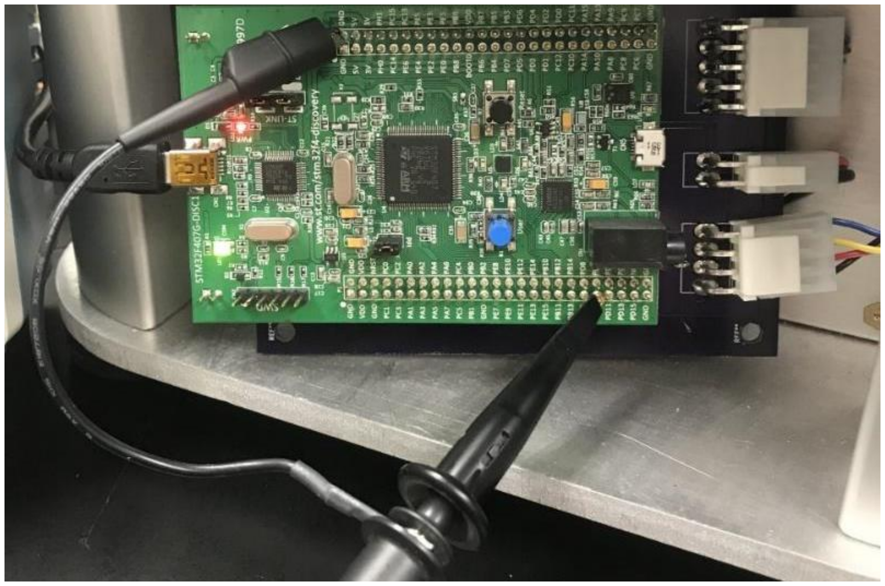


Figure 5.22 Control Loop Rate Verification using Oscilloscope

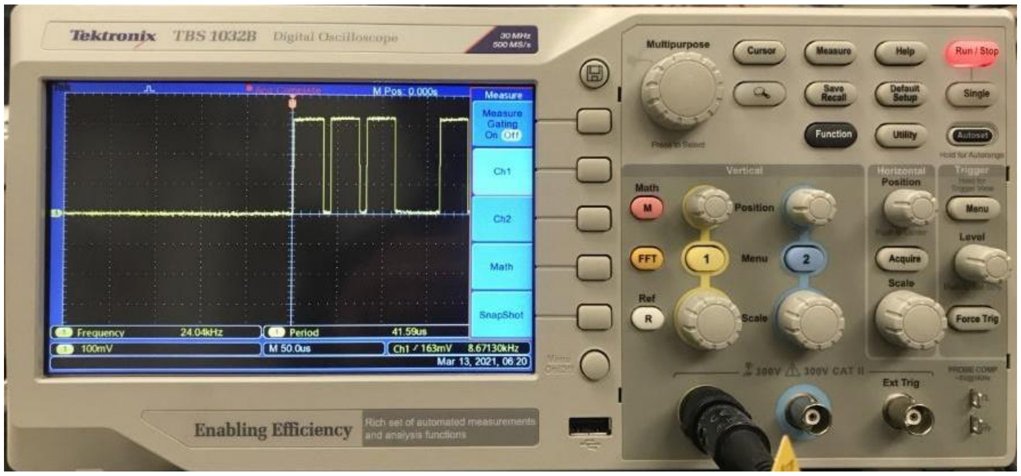


Figure 5.23 Abnormality in Control Loop Frequency

This problem arises from the generated External mode source code that utilizes interrupt-based communication between the hardware and host PC. The affected speed control plot is shown in Figure 5.24 below and is identified to occur near the same interval.

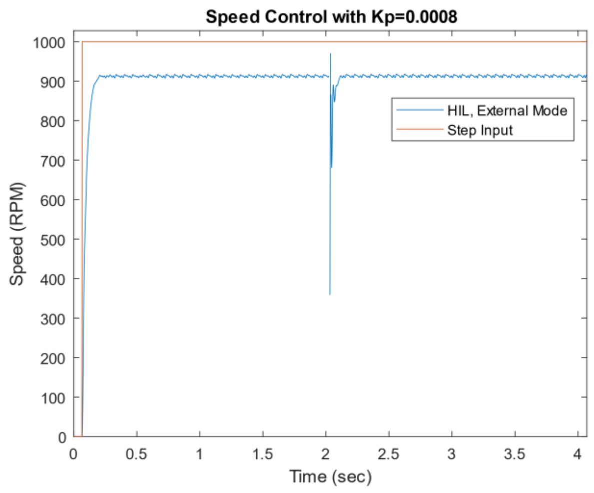


Figure 5.24 Resulting Plot of Speed Control with External Mode

Also, HAL_UART_RxCpltCallback function is hardcoded within the External mode's source code. This is a callback function that is called when the data receive is completed on a UART port. The MathWorks' usage of the function, as shown in Listing 5.1, prevents any new custom blocks to define a new behavior of the callback function during receive interrupts. The compilation will fail with the multiple definition error. This severely hinders the performance of the serial communication, and the user is forced to utilize communication with blocking.

Listing 5.1 STM32SerialRtiostream.c Interrupt Callback Function Snippet

```
/*****************************************************************************
***
* Function Name  : HAL_UART_RxCpltCallback
* Description    : Rx Transfer completed callbacks.
* Input          : UART handle
*****************************************************************************
**/
void HAL_UART_RxCpltCallback(UART_HandleTypeDef * huart)
{
    HAL_StatusTypeDef status;

    /* Increment pointer on receive buffer. */
    ptSet++;

    /* Increment number of receive char. */
    nbRcv++;
```

Another issue coupled with the External mode code is the usage of HAL functions by STM32-MAT/TARGET peripheral driver blocks. When the debugger first initializes the project before connecting to the Simulink diagram, the CCR value within the register viewer is observed to be at an arbitrary value instead of zero. This is undesirable as the CCR register is responsible for the PWM generation. It will result in indeterministic behavior, and the system response will be random. The last bug observed within the Simulink diagram using External mode is the usage of dashboard blocks. To make the simulation more user friendly, implementing visual blocks without complex signal lines can be beneficial in contrast to the diagram shown in Figure 5.20 above. But when the diagram is running, either the dashboard blocks do not update or disappear, giving no control of the device in real-time.

5.4.5 Workarounds Using STM32-MAT/TARGET

To keep the integrity of MBD approach with user friendliness from start-to-end development cycle and to avoid students from having to debug code manually, Simulink External mode will not be employed for HIL verification. Instead, a USB serial communication protocol will be developed to allow students to interact with the hardware in real-time. This approach will alleviate the bandwidth issue as well as establishing a simple and swift connection with the microcontroller. Thus, making it an efficient method for the hardware testing in HIL.

To enable a serial communication protocol over the micro-USB port, STM's USB Device middleware will be used to implement USB OTG FS. Once the firmware is flashed, the Discovery board only requires the micro-USB for data acquisition. This simpler connectivity also makes it a desirable approach to hardware testing without cluttering up the apparatus with additional wiring. In contrast, FTDI converter requires minimum of three wires and a USB cable. Listing 5.2 below shows the transmit function that will send string data with a length of 2048 lines. Each string contains time, position, velocity, and the current measured from the amplifier.

Listing 5.2 USB OTG FS Transmit Function

```
for (uint16_t i=0; i < 2048; i++) { // send over the float data
    __disable_irq();
    snprintf(strBuffer, 51, "t:%0.3f|p:%0.3f|v:%0.4f|a:%0.3f|",
             data[i].floatVals[0],data[i].floatVals[1],
             data[i].floatVals[2],data[i].floatVals[3]);
    __enable_irq();
    while(CDC_Transmit_FS((void*)strBuffer, sizeof(strBuffer)));
}
return;
```

To use the custom source code files within the Simulink context, MATLAB System Block can be implemented. MATLAB System Block is like an interpreter between the two programming languages. It will translate C functions to be used within Simulink's code generation. Listing 5.3 is a snippet of the MATLAB System Block source code. It is a template file that can be modified to fit the use. The MATLAB function coder.cinclude() is used to include the header file during its initialization as the function name suggests. Then, another MATLAB function coder.ceval() is used to call the C function in the main control loop. The copyStr function contains the code snippet in Listing 5.2. Additional variable names such as posMeasured are the input arguments to the function.

Listing 5.3 MATLAB System Source Block

```
methods (Access=protected)
    function setupImpl(obj)
        if coder.target('Rtw')
            % Call C-function implementing device initialization
            coder.cinclude('ReceiveTransmit.h');
        else
            % Place simulation setup code here
        end
    end
    function stepImpl(obj,posMeasured,velMeasured,currentMeasured)
        if coder.target('Rtw')
            % Call C-function implementing device output
            coder.ceval('copyStr',posMeasured,velMeasured,currentMeasured);
        else
            % Place simulation output code here
        end
    end
```

Chapter 6 - HIL Verification and Analysis

This chapter will cover a few lab exercises to compare the results and performance between the original MotorLab and the MBD deployed embedded controller. Figure 6.1 below depicts the full HIL Simulink diagram developed for this thesis.

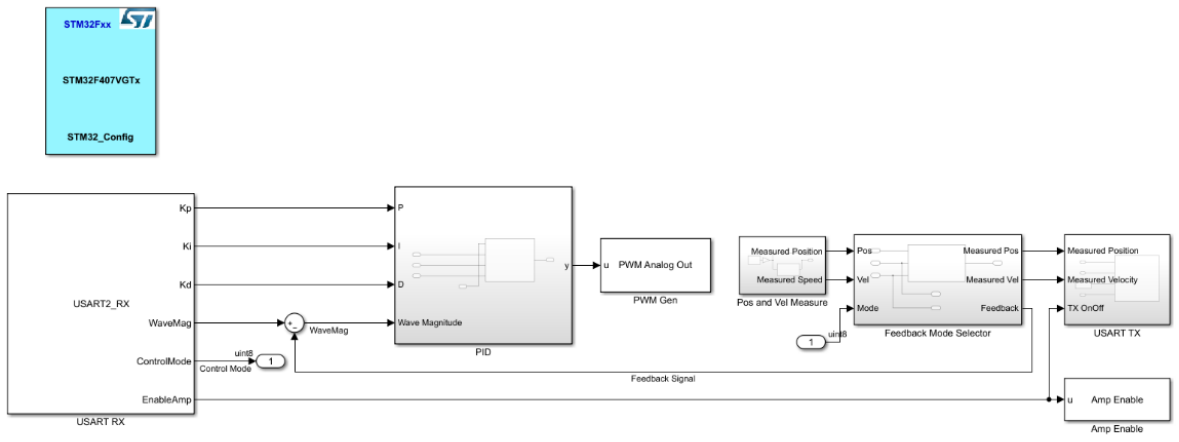


Figure 6.1 HIL Simulink Diagram

Starting from the left, the blue block is the target hardware configuration block. Below that is the communication-receive block. It will listen to the host PC's commands such as PID gains or what type of input wave to be used. Next block is the PID subsystem. It has an additional input parameter which is the input wave magnitude. Following the PID is PWM Analog Out block. As the name suggests, this block sends the PWM signal to the amplifier. Note there is no signal coming out of the block. This is because the feedback sensor is the encoder, which is the next block. Encoder block will read encoder counts to measure the position of the motor. The position values are also used to calculate the speed using the low pass filter inside. The next block named Feedback Mode Selector is used to send back either the position or velocity depending on the controller mode. The last two are the communication-transmit and Amp Enable blocks. The communication-transmit block will send the data back to the host PC when the transmit buffer is full. More details on the data acquisition will be covered in the following subsection.

Shown in Figure 6.2 below highlighted in red is the control loop timing verification using an oscilloscope. Comparing the measured frequency to Figure 5.23 in the previous chapter, the final embedded controller developed by this thesis shows a uniform interval at 10kHz.

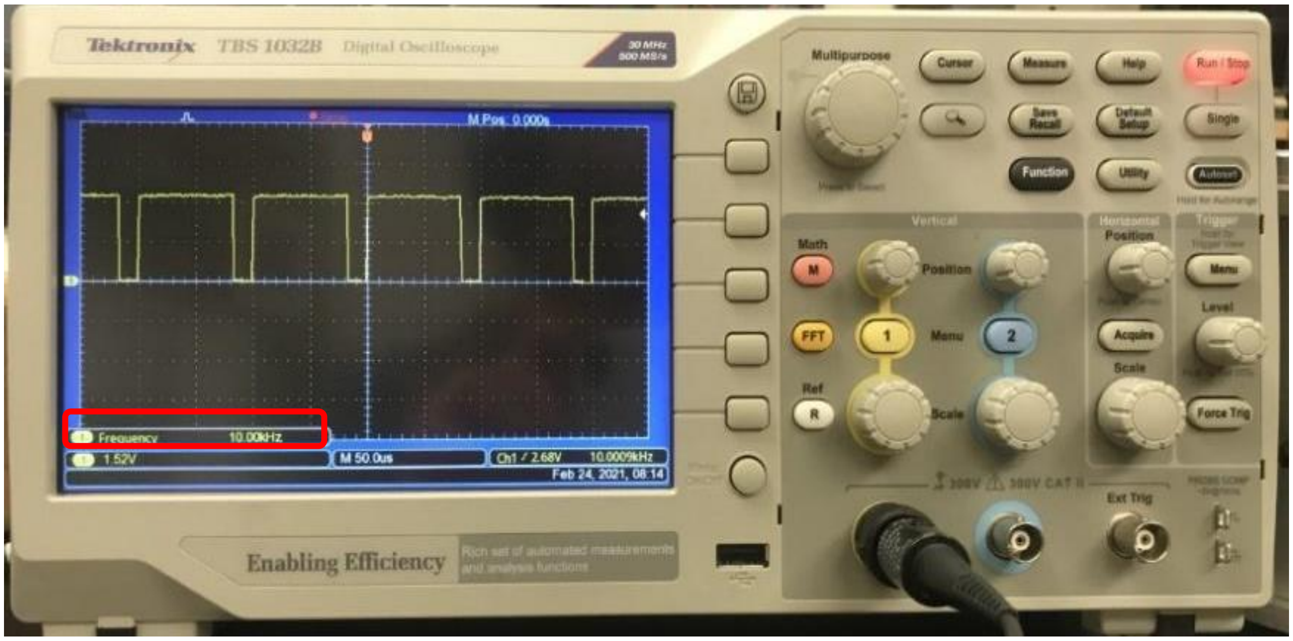


Figure 6.2 Oscilloscope reading of GPIO pin at 10kHz

6.1 Data Acquisition

Table 6.1 below shows the structure of the incoming data stream from the microcontroller. The Command value will depend on the controller mode. The rest is always sent from the microcontroller to be used for analysis.

Table 6.1 Data Acquisition Stream Format

Column	**1**	**2**	**3**	**4**	**5**
Description	Time	Command	Motor Position	Motor Speed	Motor Current
Variable	t (sec)	θ_c (deg), w_c (rpm), i_c (amp)	θ (deg)	ω (rpm)	i (amp)

6.2 Velocity Control

As introduced in Chapters 3 and 5, the high frequency dynamics affecting the system will be discussed using the speed control responses. Refer to Figure 6.4 below. The legend name HIL denotes the embedded controller generated from the MBD, Nominal model is from the MATLAB simulation code without the low pass filter, and High Freq. Model from the same MATLAB simulation but with the low pass filter added.

The nominal plant is described as the following from Equation (3.4) where k_t is the motor torque constant, k_{dr} is the conversion from radian to degree, and k_{rd} is the output conversion to RPM:

$$G_{system} = \frac{k_t k_{dr} k_{rd}}{J_m s + b}$$

Multiplying the above equation with a proportional controller $G_c = K_p = 0.0032 \frac{Amp}{RPM}$, the first order closed loop system response can be obtained using the MATLAB syntax as shown below:

Listing 6.1 Nominal Closed Loop System Response

```
Tnominal = feedback(kp*Gs, 1);
```

For the higher order system response with the low pass filter, the following equation is used:

$$G_{hf} = \frac{{\omega_{cf}}^2}{s^2 + 2\zeta\omega_{cf}s + {\omega_{cf}}^2}$$

Similar to the nominal response, the closed loop response is obtained:

Listing 6.2 High Frequency Closed Loop System Response

```
Thf = feedback(kp*Ghf*Gs, 1);
```

For the actual system response using the embedded controller, the GUI based on Simulink function blocks and various dashboard blocks are used as shown in Figure 6.3 below. Users can edit various parameters to observe and collect system output data.

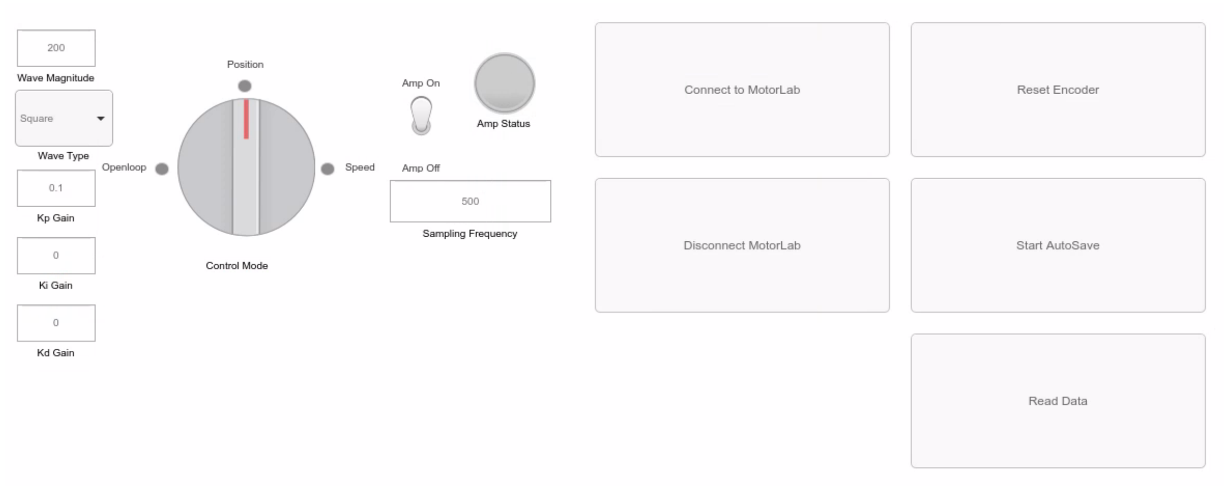


Figure 6.3 HIL Graphical User Interface

The HIL GUI is developed to reflect the current MotorlabGUI parameters in order to offer students the same required functionality to conduct lab exercises. The resulting plot of a speed control using a specific Kp gain is shown below in Figure 6.4.

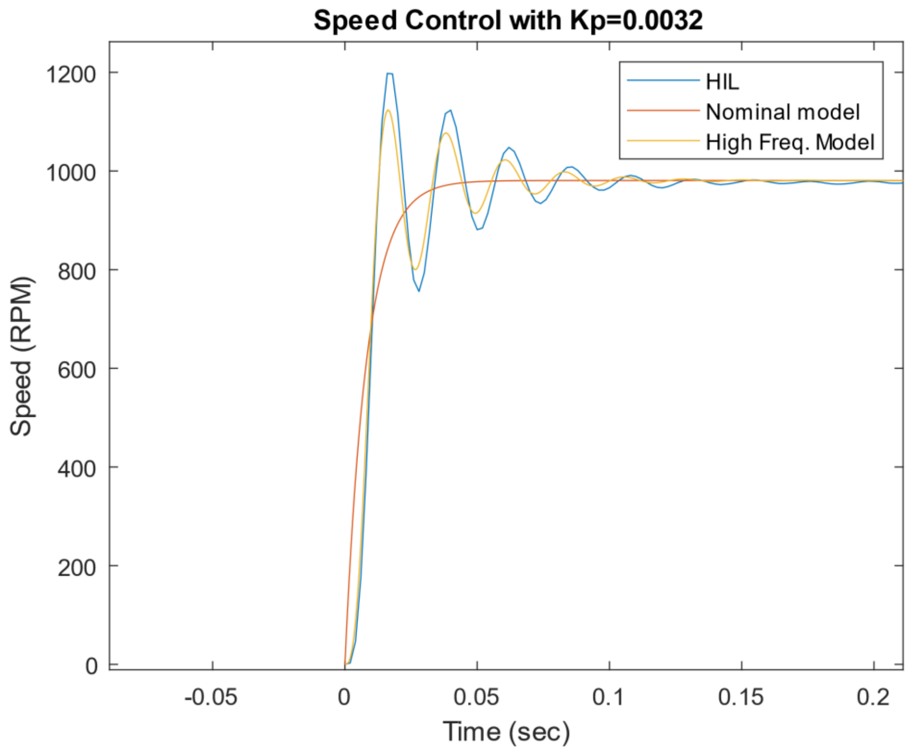


Figure 6.4 High Frequency Dynamics in the System Responses

As the plot above clearly shows, there are some differences between the plots induced from the high frequency dynamics of the filter. The filter's cutoff frequency, or its natural frequency, is set at 300$\frac{rad}{s}$. Applying the 10x rule of thumb, this gives the upper limit for the closed loop poles of the system at around 30$\frac{rad}{s}$. With a proportional gain of $0.0032\,\frac{Amp}{RPM}$, the magnitude of the closed loop pole from the nominal model is 120.8$\frac{rad}{s}$, and is close to the real pole of the system with the filter at -136.1$\frac{rad}{s}$. These are well over the rule of thumb limit and resulted in the oscillations with noticeable amplitude. Moreover, the other two poles from the model with the filter have a magnitude of 282.6$\frac{rad}{s}$ which is much closer to the magnitude of the real pole and therefore the complex poles affect the system response noticeably. And further deviation can be observed from the actual system response. This phenomenon will be observed again in position control using the embedded controller.

6.3 Position Control with P Controller

When a proportional gain is 0.01$\frac{Amp}{deg}$, the responses between the HIL and MotorlabGUI are the same, but the difference between the two and the MIL, SIL, and PIL responses can be recognized as shown in Figure 6.5 and Figure 6.6 below. This is due to the high frequency dynamics or other non-linear dynamics that are not modeled in the simulations as Kp gets higher. To compare this higher Kp to the lower Kp of 0.005$\frac{Amp}{deg}$, the noticeable difference in plot vanishes.

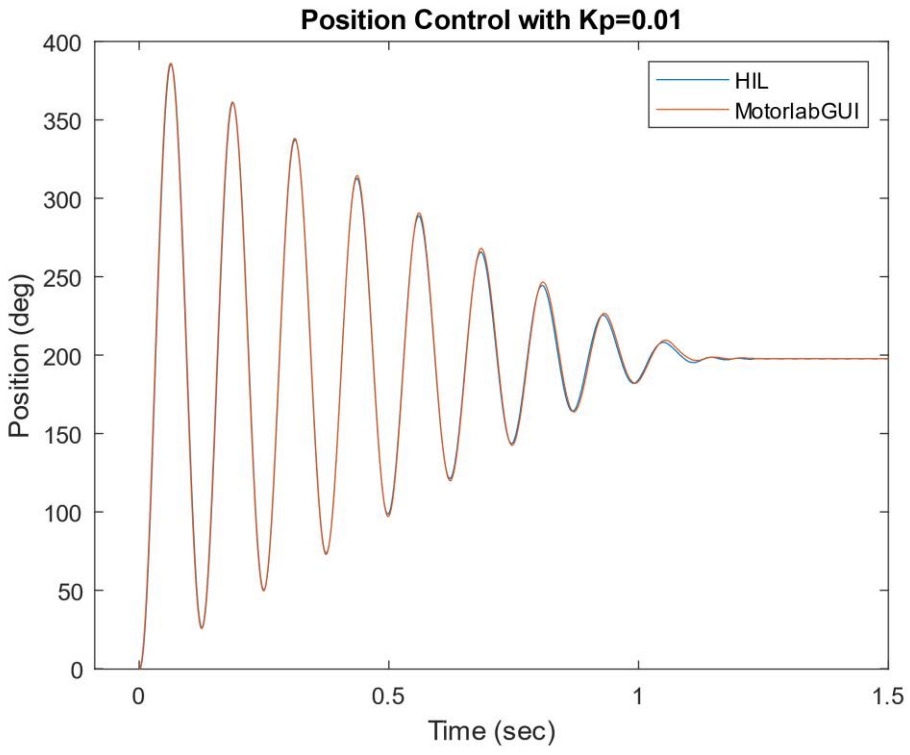


Figure 6.5 Proportional Position Control with Higher Gain

The above figure is the position control comparison between the MotorLab device and the HIL response. To minimize the variables in system response, same motor was used with different microcontrollers.

Figure 6.6 below compares the same system response using the two different MBD steps, PIL and HIL. The PIL utilizes microcontroller in loop with the host PC to run the embedded controller with the simulated plant. In contrast, the HIL is a full verification method with the actual plant and the embedded controller.

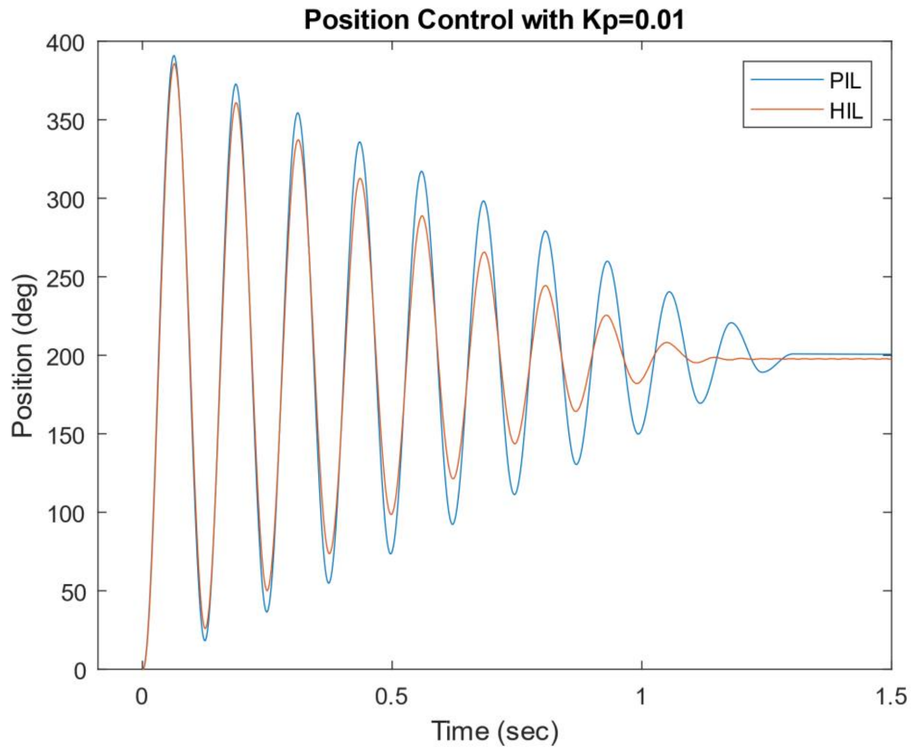


Figure 6.6 PIL vs. HIL Proportional Control Comparison

The visible difference closely resembles the speed response plot above when the high frequency dynamics start to affect the system, but the frequency of the oscillations are in phase.

In order to confirm that the higher gain is affecting the system response, the following figures were obtained using a lower gain of $0.005\frac{Amp}{deg}$.

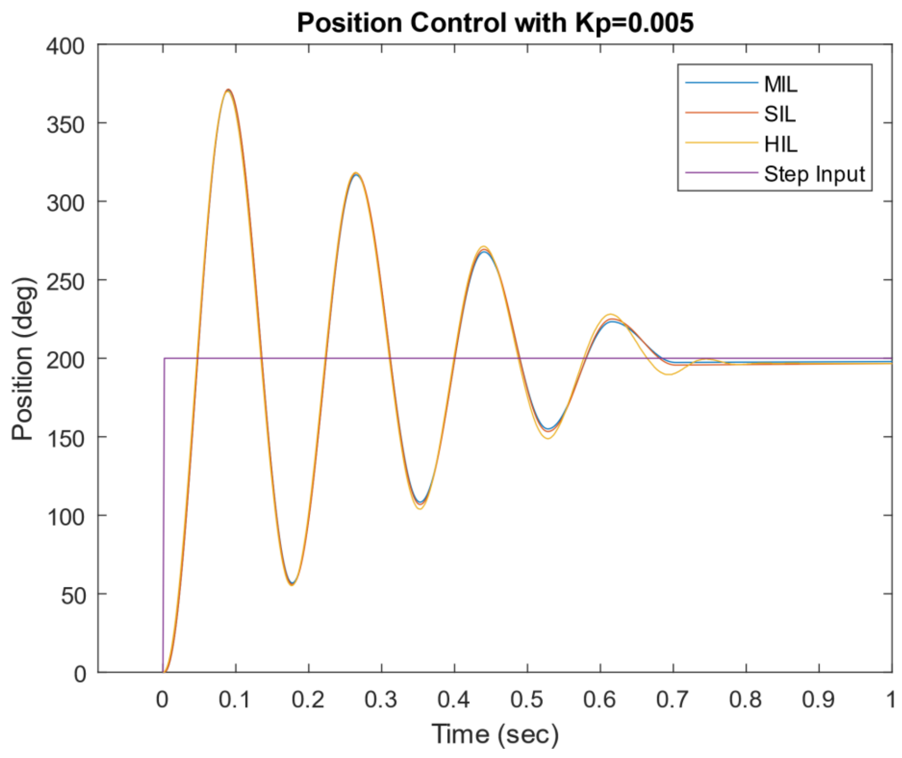


Figure 6.7 Comparison Between the MBD Simulations and the HIL Verification

Unlike the response resulting from the higher gain, the system response above shows well matching plots. This can be expected as the gain gets lower, the magnitude of higher order poles gets further away from the magnitude of the real pole. The following Figure 6.8 and Figure 6.9 are comparisons between the various MBD stages.

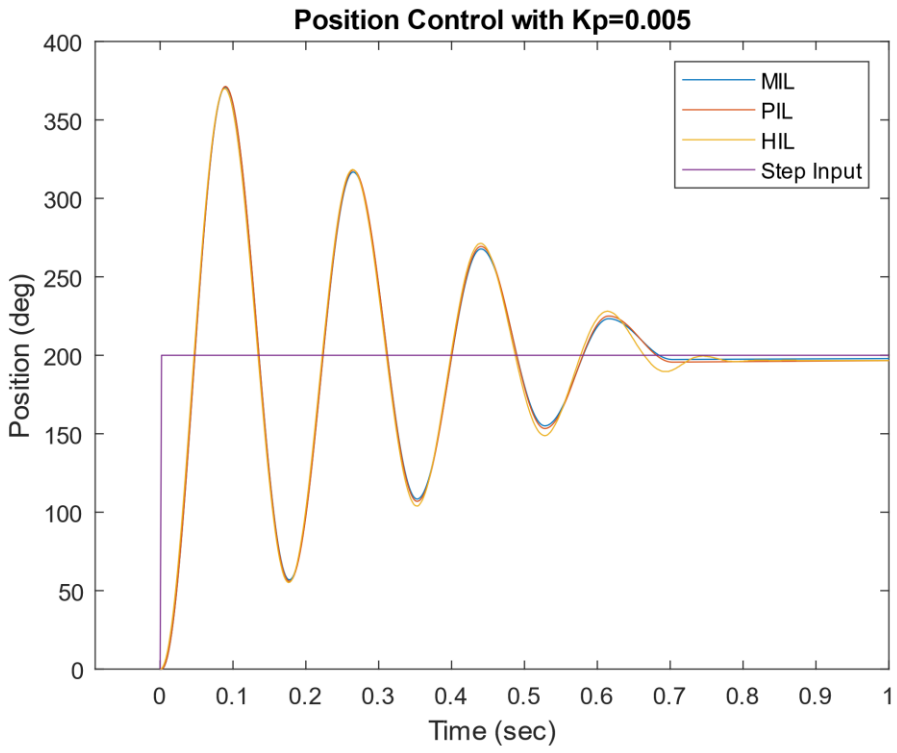


Figure 6.8 Comparison Between the MBD Simulations and the HIL Verification

Figure 6.8 above replaces the SIL response from Figure 6.7 with the PIL response. Again, the plots appear to be a good match with the HIL response slightly larger in amplitude. And the final MBD response figure below is the pure simulation without the actual motor in the loop.

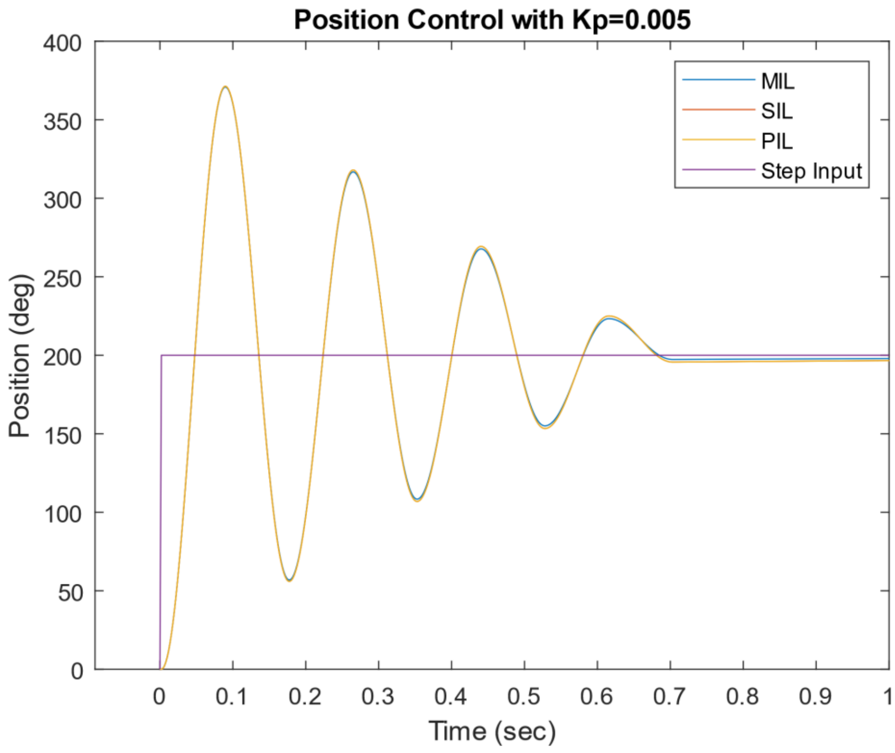


Figure 6.9 Comparison Between the MBD Simulations

6.4 Position Control with PD Controller

The lab #6 students conduct is the utilization of Proportional-Derivative controller. This lab is designed to teach students the limitations of using the square wave input to test systems. Up until this lab, students are only required to use step input to the system. To establish a diverse understanding of control theory coupled with the limitations of physical system, students learn about the output saturation and why different input wave types are necessary. For example, suppose a step input of 2000 deg is used with a PD controller as the following:

$$\theta_c = 2000deg$$

$$K_d = 0.0001\frac{amp \cdot sec}{deg}$$

$$G_c(s) = K_p + K_d s$$

With the initial error of 2000 deg at the first timestep, then the output from the controller becomes:

$$e(t) = 2000deg$$

$$L\{e(t)\} = E(s) = \frac{2000}{s}$$

$$I_c(s) = E(s) \cdot G_c(s) = \frac{K_p \cdot 2000}{s} + K_d \cdot 2000$$

Then, taking the inverse Laplace transform to get the step and impulse functions:

$$L^{-1}\{I_c(s)\} = K_p \cdot 2000 \cdot u(t) + K_d \cdot 2000 \cdot \delta(t)$$

Here, impulse function has an area of $K_d \cdot 2000 = 0.2amp \cdot sec$. Since the embedded controller has a finite step size of 10kHz, or 0.0001sec, this suggests that the amplifier needs to output 20,000,000 amp or extremely small time steps to correct the initial error. For obvious reasons, this is not feasible for lab exercises or even for the most industrial applications.

This control problem highlights that the actual energy which drives the plant is much smaller than predicted by the linear model due to the amplifier saturation at ±3amps. To mitigate this inadequate PD controller response, triangle input wave function will be implemented. It is also a good idea to test systems with various input signals, not just step inputs.

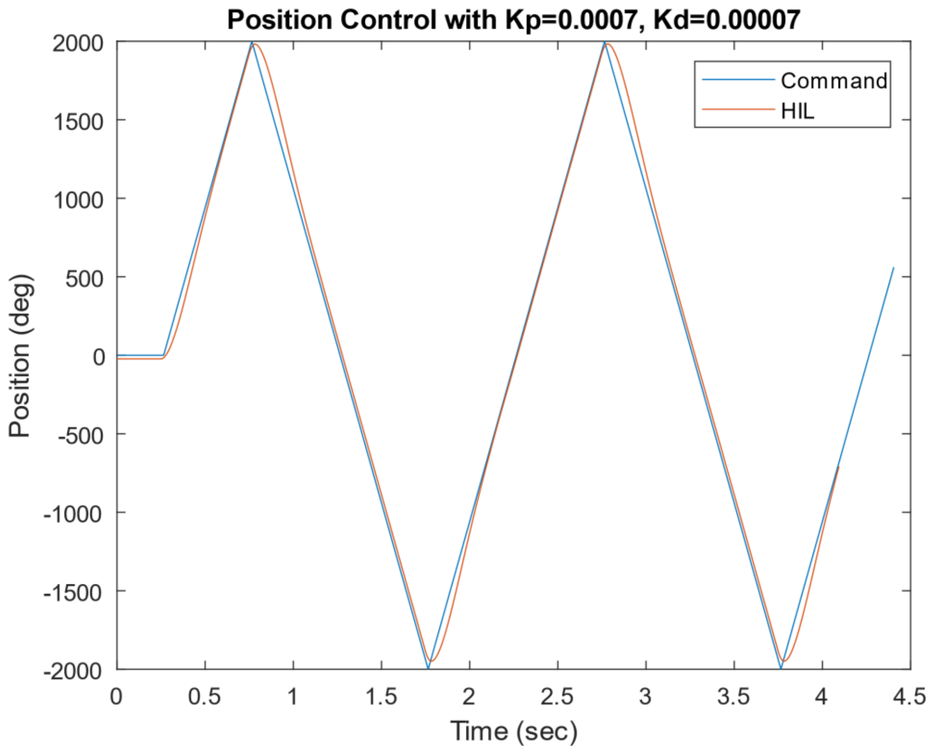


Figure 6.10 Smaller PD Gains with Large Command Input

Figure 6.10 above shows the system response when triangle input wave is used against lower PD controller gains. There exists a small deviation from the command tracking, but it is sufficient in correcting the error signal. In contrast, Figure 6.11 below shows the system response with higher PD gains. Comparing the two different PD controllers, the higher PD gains show better tracking to the input command.

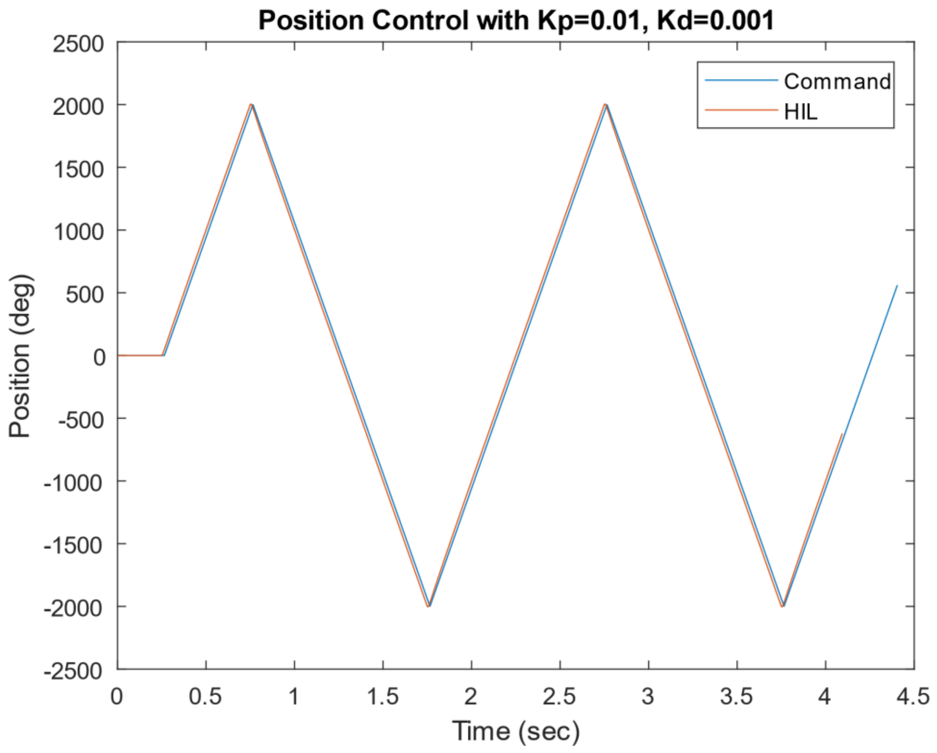


Figure 6.11 Higher PD Gains with Large Command Input

Finally, the embedded controller produced from the MBD is compared to the original MotorLab device using the same gains as Figure 6.11. Figure 6.12 below shows very closely matching plots and Figure 6.13 shows a zoomed in window of Figure 6.12. Note the difference is miniscule, showing that the MBD deployed control system matches well with the current device.

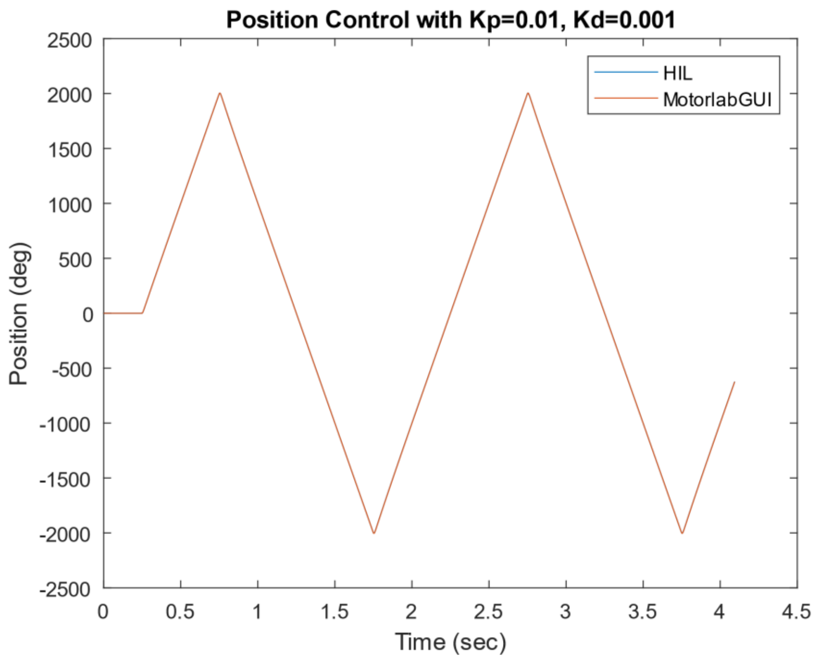


Figure 6.12 HIL vs. MotorlabGUI with Higher PD Gains

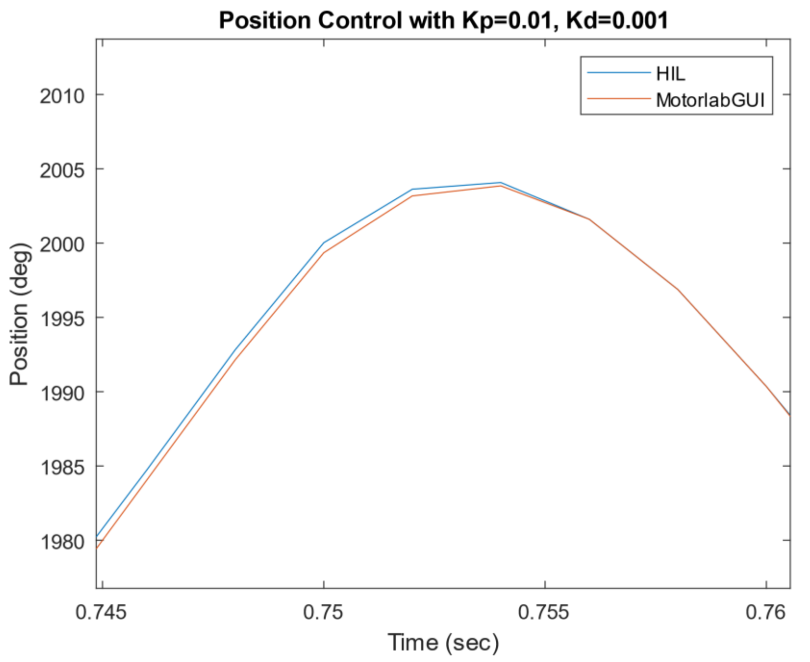


Figure 6.13 Closer Look at Figure 6.12

Chapter 7 - Additional MBD Applications

As stated in the previous chapter, taking advantage of the software used in this thesis can assist with extending the MBD approach in embedded control applications. This chapter will introduce the basic MBD deployment of an embedded control using the Arduino based robot car. The robot car is equipped with a distance measuring sensor as its feedback signal. Below is the hardware image. It is composed of a basic motor driver, Arduino Uno, and distance sensor. The distance sensor has ultrasonic transmitter and receiver. The distance is calculated by measuring the time it takes for the sound to reflect from an object back to the receiver. The range of distance it can measure is between 2 cm to 400 cm.

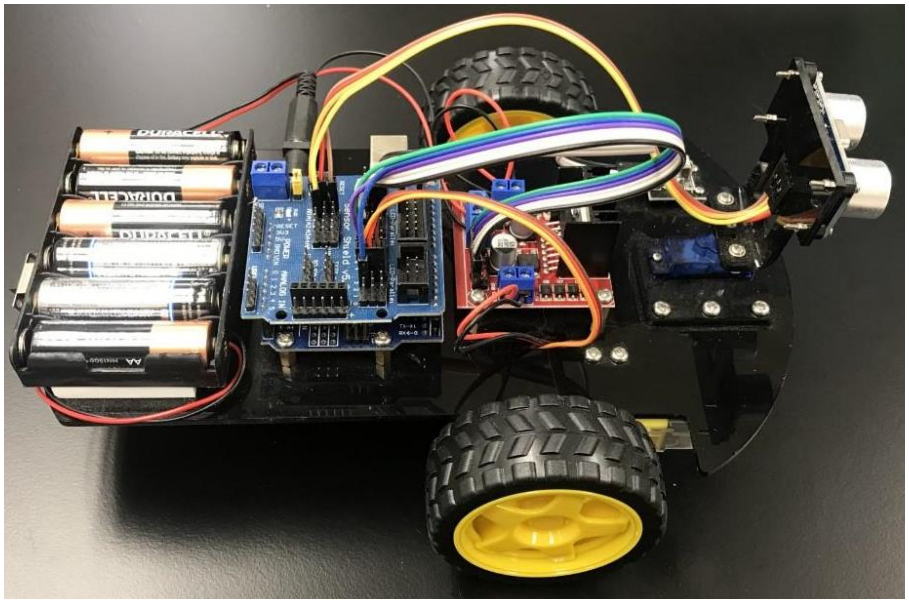


Figure 7.1 Arduino Robot Car with Ultrasonic Distance Sensor

The Simulink diagram is shown below in Figure 7.2. For this Arduino target board to work with Simulink, additional hardware support packages must be downloaded. To note, there are many communities and forums dedicated to setting up Arduino with Simulink. This can be helpful with troubleshooting the setup. Unfortunately, this was not the case for the Discovery board since it requires steeper learning curve when compared to Arduino boards.

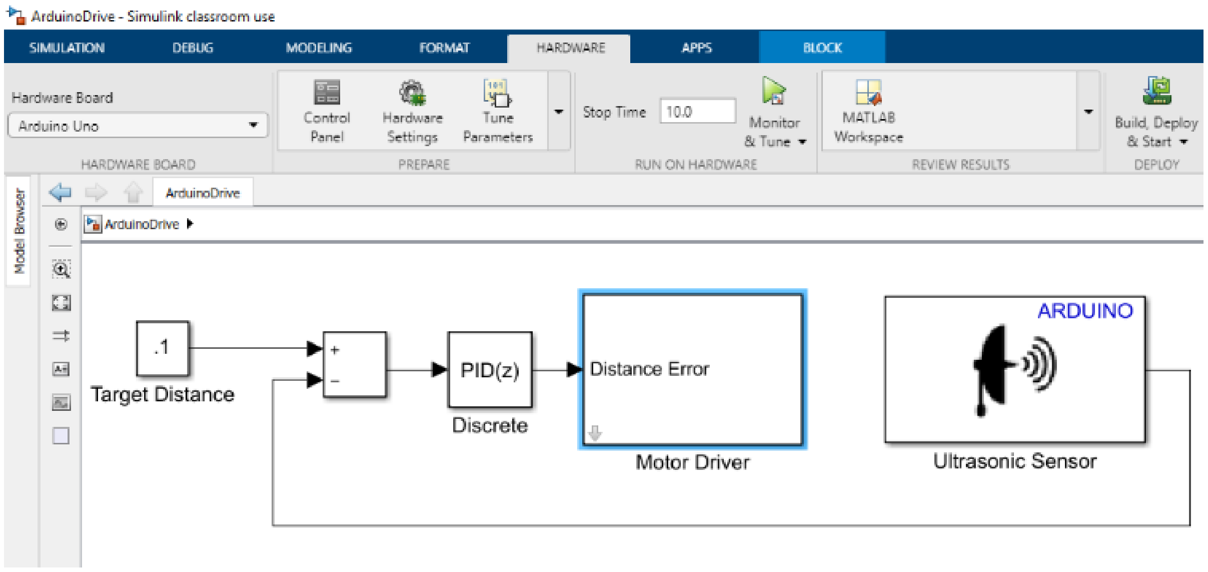


Figure 7.2 Visual Deployment of Embedded Controller using Arduino Uno

The diagram is designed to control the distance in front of the robot car, which is a simple position control application. This was chosen because the concept of control system can be more obvious when it relates to something students can grasp. In this case, an overly simplified adaptive cruise control using PID. The block named Target Distance is the reference input by a user. The vehicle will try to keep this distance. It is measured in meter. The block named Motor Driver is a Simulink subsystem which can help with simplifying the topmost view of the diagram. The detailed view of the subsystem block is shown in Figure 7.3 below. Additional logic was required to drive the system forward or backward depending on the distance sensor readings.

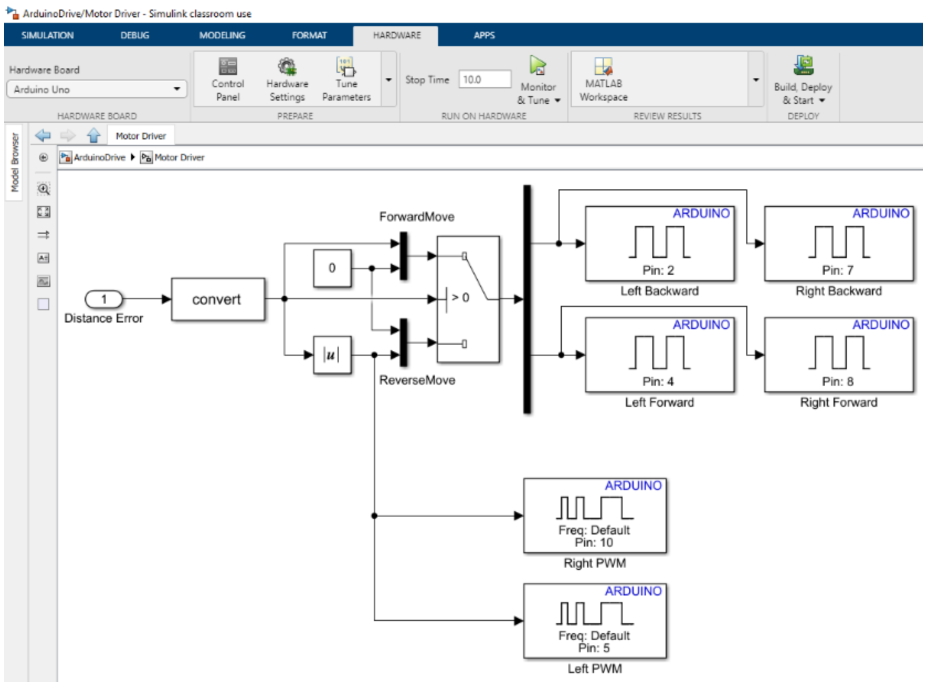


Figure 7.3 Motor Driver Subsystem

As this vehicle was inexpensive and intended for only visualizing the concept of PID controller, there is room for performance improvements such as adding wheel encoders. Wheel encoders can be used as an additional feedback sensor, making the system more controllable. But nevertheless, the entire system costs less than $30 and is capable of integrating a basic MBD development. It is also modular where students can take it around and work with it at their own leisure.

Chapter 8 - Conclusions

The main objective of this thesis was to demonstrate the application of MBD to expand and advance the current motor control apparatus. This was achieved by adopting the modern methodology used in embedded controller development. The primary goal of this research was to develop an easy-to-use graphical interface for developing and verifying the embedded control system for students. This would allow students to drag and drop proper blocks within Simulink to design and implement an embedded controller. For the verification using HIL, a few example lab exercises from the introductory control theory course were examined and analyzed using the rapidly deployed embedded controller based on MBD. The degree to which the newly created controller matched the performance of the original device was very encouraging. Furthermore, an additional inexpensive MBD application based on the Arduino robot car was briefly introduced.

Many difficulties were encountered trying to minimize the cost associated with the MBD implementation. The free to use software had limited documentation or resources were not available from the supplier for it to function properly. Thus, to integrate the MBD required a very lengthy process of trial and error for some parts of the program to work. Other parts had to be completely redeveloped and substituted for the MBD to function as intended.

When the rapidly deployed embedded controller was compared to the original MotorLab device, the difference in performance was insignificant and satisfactory. The operating procedure is very similar to that used by the MotorLab software, making the embedded controller familiar to instructors and students. Applying the researched method of creating new blocksets to more complex systems, a new course on teaching students the MBD process could be created. The outcome of the course could include the MBD designing process, testing with hardware, analysis of auto-generated firmware, and the controller tuning process to drive the system more efficiently.

Realistically, the solution developed by combining many tools from marginally supported sources create a significant problem for continued use. Maintaining a functioning system in the face of uncoordinated software updates and the need to update locally created tools to retain compatibility would require support from well-trained individuals. The elegant option is to acquire the more extensive MathWorks tool set that are designed specifically for MBD . The problems confronted in this thesis highlight many of the basic issues associated with establishing, maintaining, and operating effective teaching laboratories in an educational environment.

References

[1] L. D. Feisel and A. J. Rosa, "The Role of the Laboratory inUndergraduate Engineering Education," *Journal of Engineering Education,* 2005.

[2] A. Bruckman, "Can Educational Be Fun?," *Game Developer's Conference*, vol. 99, pp. 75-79, 1999.

[3] D. J. Tanis, "Exploring Play/Playfulness and Learning in the Adult and Higher Education Classroom," The Pennsylvania State University, 2012.

[4] R. Krauss, "Combining Raspberry Pi and Arduino to form a low-cost, real-time autonomous vehicle platform," in *American Control Conference*, Boston, 2016.

[5] MathWorks, "MathWorks," 2014. [Online]. Available: https://www.mathworks.com/content/dam/mathworks/mathworks-dot-com/campaigns/portals/files/adopting-model-based-design/why-adopt-model-based-design-for-embedded-control-software-development.pdf. [Accessed 2021].

[6] M. Utting and B. Legeard, Practical Model-Based Testing: A Tools Approach, Morgan Kaufmann Publishers, 2007.

[7] K. Henderson and A. Salado, "Value and benefits of model-based systems engineering (MBSE): Evidence from the literature," *The International Council on Systems Engineering,* 2020.

[8] J. Wieneke, D. Schinstock, W. N. White and G. Hu, "Redesign of an Undergraduate Controls Laboratory with an Eye toward Accommodating Future Upgrades," in *American Control Conference*, Baltimore, 2010.

[9] S. R. Smith, "Demonstrating introductory control systems concepts on inexpensivehardware," Kansas State University, Manhattan, 2015.

[10] N. Uzzaman, "A comparative analysis between field-oriented control and uncontrolled current operation of a brushless DCmotor," Kansas State University, Manhattan, 2016.

[11] D. Schinstock, W. White and S. Smith, "Does Inexpensive Hardware Obfuscate Simple Experiments for Control Systems Laboratories?," *American Control Conference,* 2017.

[12] D. Schinstock and W. White, "Micro-controller based update of inexpensive undergraduate control systems laboratory hardware," *American Control Conference (ACC),* no. 10.1109/ACC.2015.7171160, pp. 2807-2812, 2015.

[13] FTDI Chip, "Corporate Profile," FTDI Chip, [Online]. Available: https://ftdichip.com/corporate-profile/. [Accessed 12 December 2021].

[14] Future Technology Devices International, "FTDI Knowledgebase," 2010. [Online]. Available: https://www.ftdichip.com/Support/Knowledgebase/index.html?whatbaudratesareachieveabl.htm. [Accessed 2021].

[15] MathWorks, "Model-Based Design," MathWorks, [Online]. Available: https://www.mathworks.com/solutions/model-based-design.html. [Accessed 12 December 2021].

[16] ST, "Integrated Development Environment for STM32," ST, [Online]. Available: https://www.st.com/en/development-tools/stm32cubeide.html. [Accessed 12 December 2021].

[17] ST, "STM32Cube initialization code generator," ST, [Online]. Available: https://www.st.com/en/development-tools/stm32cubemx.html. [Accessed 12 December 2021].

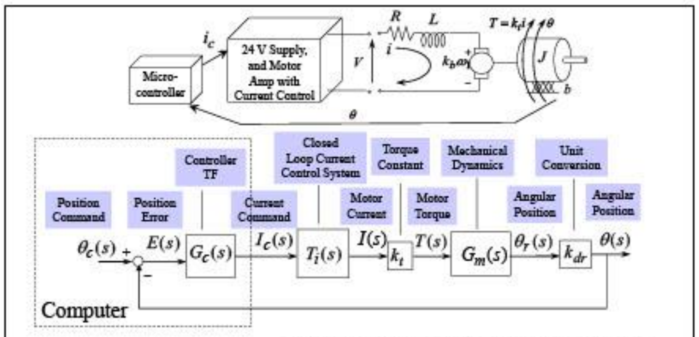
[18] MathWorks, "Simulink Real-Time," MathWorks, [Online]. Available: https://www.mathworks.com/products/simulink-real-time.html. [Accessed 12 December 2021].

[19] MathWorks, "Code Verification and Validation with PIL and Monitoring and Tuning," MathWorks, [Online]. Available: https://www.mathworks.com/help/supportpkg/stmicroelectronicsstm32f4discovery/ug/code-verification-and-validation-with-pil-and-external-mode.html. [Accessed 12 December 2021].

Appendix A - ME570 MotorLab Laboratory Assignments

Laboratory #3

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



$$G_m(s) = \frac{\theta_r(s)}{T(s)} = \frac{1}{Js^2 + bs}$$

$G_c(s) = K_p + K_d s$, PD controller

and CLTF

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

J – motor and double collar inertia (from motor lab handout)

b – viscous friction coeficient (estimated to be 3×10^{-5} N·m·s/rad)

k_t – motor torque constant (from motor lab handout)

$k_{dr} = 180°/\pi\,(rad)$ – angular postion unit conversion

$T_i(s) = 1$ (amplifier assumed "fast")

K_p – proportional gain of controller

K_d – derivative gain of controller

$\theta_r(t)$ – angular position (measured in radian)

$\theta(t)$ – angular position (measured in degrees)

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 Lab 3 Narrative in Canvas.

Closed loop position control system model (a TF).

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

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.

$E(s) = \theta_c(s) - \theta(s)$

$I_c(s) =$ _____$E(s)$

$I(s) =$ ______$I_c(s)$

$T(s) =$ ______$I(s)$

$\theta_r(s) =$ ______$T(s)$

$\theta(s) =$ _______$\theta_r(s)$

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 $K_d = 0.001\ (Amp \cdot s/\deg)$.
- 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. Set the ***Data Collection* ▸ *Sample Rate*** to 500Hz, then press the ***Calculate Step Response Timing*** button. Notice the waveform frequency automatically changes.
- 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.
- Don't forget to do the lab narrative on canvas - individually

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

In Lab #2 we found a linear estimate of the friction in the motor of the Motorlab system. It was obvious from the data that there were some nonlinear effects in the friction. In this lab you will try to capture these nonlinear effects in a simulation of the system using Simulink. You will use a model with a high coefficient of friction at low velocity and lower coefficient at high velocity as shown in the second figure to the right. Using a simulation of the nonlinear system you should be able to generate a nonlinear initial condition response for the velocity of the motor that is very similar to the experimental data in the first figure.

In other labs you have/will experiment with a position control system that uses a strictly proportional controller. Although you will not collect experimental data for this system in this lab, you will simulate it using your nonlinear model. In this model you will assume the closed loop current control system is very fast (i.e. $T_i(s) = 1$).

Intro to Simulink

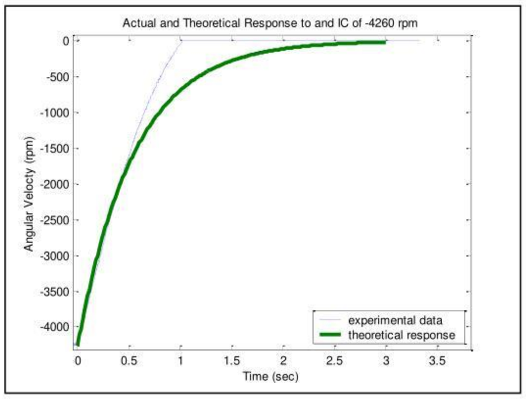
Simulink is a graphical simulation tool that is a companion to MATLAB. Your lab instructor will walk you through the process of building a simple model.

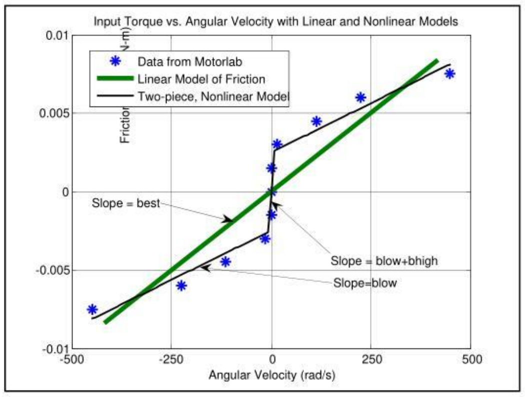
Generating the initial condition response.

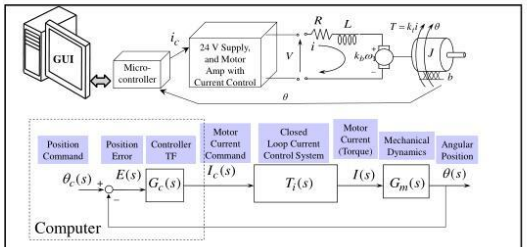
The first Simulink diagram given in this handout should be used to generate a response like that shown in the first figure. The model in this Simulink diagram is contained in the file "InitialConditionSimStart.mdl." This file should run after you have run the "setup.m" file. However, you will have to correct one thing (you figure it out) in order for the linear model to work correctly. With this simulation you only need to validate that the nonlinear model reproduces something very similar to the experimental data from the actual system shown in the first figure. Then you should move on to the closed loop simulation.

Generating the closed loop position control responses.

The second Simulink diagram given in this handout should be used to generate closed loop position control responses. The model in this Simulink diagram is contained in the file "PosCntrlSimStart.mdl." You should finish building the middle model (the linear model) and the bottom model. The bottom model should take into account that the actual Motorlab system will only allow +/- 3 Amps of current to be commanded to the motor amplifier. USE A SATURATION BLOCK TO LIMIT THE MOTOR CURRENT TO +/- 3 AMPS. After you have all three models working, HAVE YOUR LAB INSTRUCTOR CHECK THE OUTPUT OF YOUR SIMULINK MODELS.







J = motor inertia (from motor lab handout)

b = viscous friction coeficient (estimated to be 3×10^{-5} N·m·s/rad)

$G_m(s) = \frac{\theta(s)}{I(s)} = \frac{k_t k_{dr}}{Js^2 + bs}$

k_t = motor torque constant (from motor lab handout)

$G_c(s) = K_p$, proportional controller and CLTF

$k_{dr} = 180^\circ / \pi\,(rad)$ = angular postion unit conversion

$T_i(s) = 1$ (amplifier assumed "fast")

$\frac{\theta(s)}{\theta_c(s)} = \frac{K_p k_t k_{dr}}{Js^2 + bs + K_p k_t k_{dr}}$

K_p = proportional gain of controller

K_d = derivative gain of controller

$z = K_p / K_d$ = zero of controller

$\theta(t)$ = angular position (measured in degrees for this problem)

Then use your closed loop model(s) to generate step responses for the following three StepSizes:

1. StepSize = 200 deg
2. StepSize = 3000 deg
3. StepSize = 9000 deg

Each time you run the Simulink model you should be able to use the "GenStepPlot.m" to generate a plot in MATLAB to compare the responses of the three different models.

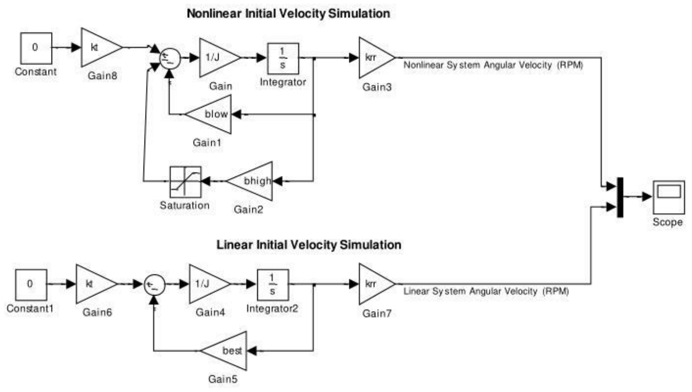
Things to Turn In

- Printout of your Simulink diagram for the closed loop simulation – no plots necessary
- Lab 5 narrative completed individually

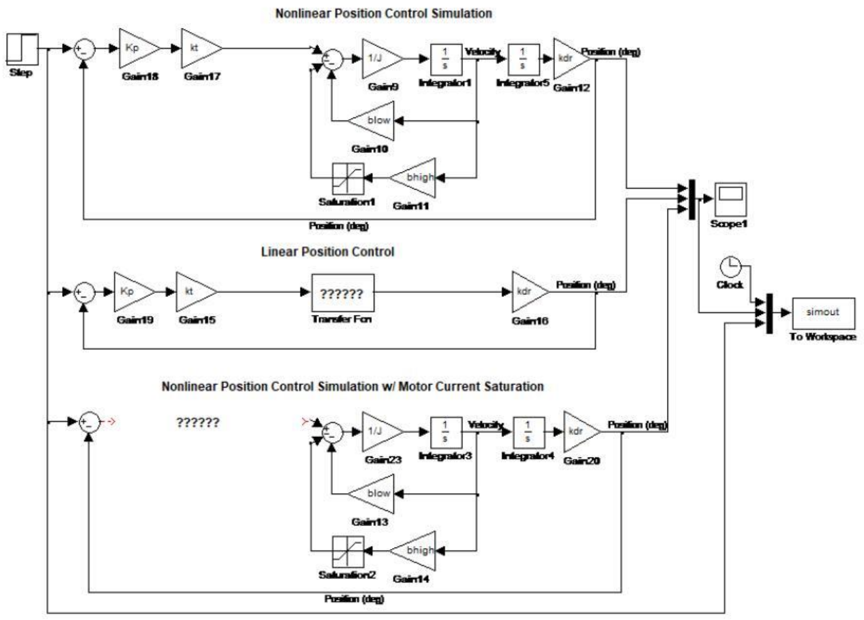
m-file code

```
%setup file
%setup file
i=[-0.15 -0.12 -0.09 -0.06 -0.03 0 0.03 0.06 0.09 0.12 0.15]; %vector of
current data
rpm=[-4270 -2150 -1100 -150 -0 0 0 150 1100 2150 4270];  %vector of velocity
data

krr = 60/2/pi;          % radians/s to rpm
kt = 0.05;              % you know this one ;)
T=kt*i;                 % convert vector current to torquer
w=rpm/krr;              % convert rpm to rad/s
w2=[-450:900/100:450];

west=[-4000 4000]/krr;
J=1.1e-5;
best=2e-5;            % friction estimate for linear model
Test=best*west;       % time constant estimate

blow = 1.25e-5;       % low friction (higher speed)      - nonlinear model
bhigh = 1.75e-2;      % high friction (lower speed)      - nonlinear model
fsat = 2.5e-3;        % Largest torque contribution from bhigh  - NL model

clear Test2
for i=1:length(w2)
    fhigh=0.0175*w2(i);
    if (fhigh > fsat)
        fhigh = fsat;
    elseif (fhigh < -fsat)
        fhigh = -fsat;
    end
    Test2(i)=fhigh + blow*w2(i);
end

plot(w,T,'*',west,Test,w2,Test2);
ylabel('Friction Torque (N-m)');
xlabel('Angular Velocity (rad/s)');
title('Input Torque vs. Angular Velocity with Linear and Nonlinear Models');

VIC=-4250/krr; % Initial velocity
kdr = 180/pi;  % Degrees to Radians
Kp = 0.01;     % controller Kp
StepSize=200;
Imax=3;

%Generate step plots from simulation output
t=simout(:,1); th1=simout(:,2); th2=simout(:,3); th3=simout(:,4); com=simout(:,5);

plot(t,th2,t,th1,t,th3,t,com);
ylabel('Angular Position (deg)');
xlabel('Time (sec)');
title('Step Response of Linear and Nonlinear Models with StepSize of 3000');
legend('Linear Model','Nonlinear Friction','Nonlinear Friction and Current Limit')
```

Simulink diagram for IC response


Nonlinear Initial Velocity Simulation
0
Constant
kt
Gain8
1/J
Gain
1/s
Integrator
krr
Gain3
Nonlinear System Angular Velocity (RPM)
blow
Gain1
bhigh
Gain2
Saturation
Scope
Linear Initial Velocity Simulation
0
Constant1
kt
Gain6
1/J
Gain4
1/s
Integrator2
krr
Gain7
Linear System Angular Velocity (RPM)
best
Gain5

Simulink diagram for closed loop response

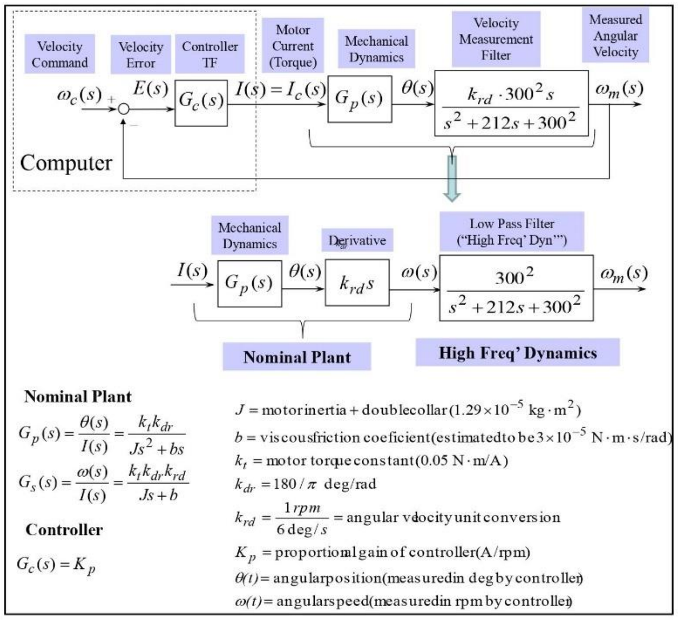


Laboratory #7

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.



Velocity Command
Velocity Error
Controller TF
Motor Current (Torque)
Mechanical Dynamics
Velocity Measurement Filter
Measured Angular Velocity
$\omega_c(s)$
$E(s)$
$G_c(s)$
$I(s) = I_c(s)$
$G_p(s)$
$\theta(s)$
$\frac{k_{rd} \cdot 300^2 s}{s^2 + 212s + 300^2}$
$\omega_m(s)$
Computer
Mechanical Dynamics
Derivative
Low Pass Filter ("High Freq' Dyn"")
$I(s)$
$G_p(s)$
$\theta(s)$
$k_{rd} s$
$\omega(s)$
$\frac{300^2}{s^2 + 212s + 300^2}$
$\omega_m(s)$
Nominal Plant
High Freq' Dynamics
Nominal Plant
$G_p(s) = \frac{\theta(s)}{I(s)} = \frac{k_t k_{dr}}{Js^2 + bs}$
$G_s(s) = \frac{\omega(s)}{I(s)} = \frac{k_t k_{dr} k_{rd}}{Js + b}$
Controller
$G_c(s) = K_p$
J = motor inertia + double collar (1.29×10^{-5} kg·m^2)
b = viscous friction coeficient (estimated to be 3×10^{-5} N·m·s/rad)
k_t = motor torque constant (0.05 N·m/A)
$k_{dr} = 180/\pi$ deg/rad
$k_{rd} = \frac{1\,rpm}{6\,deg/s}$ = angular velocity unit conversion
K_p = proportional gain of controller (A/rpm)
$\theta(t)$ = angular position (measured in deg by controller)
$\omega(t)$ = angular speed (measured in rpm by controller)

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.

Laboratory #7

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 (include units) copy of your MATLAB code.
- Complete the Lab 7 narrative – individually.

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);

pl = 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(pl,'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');
```

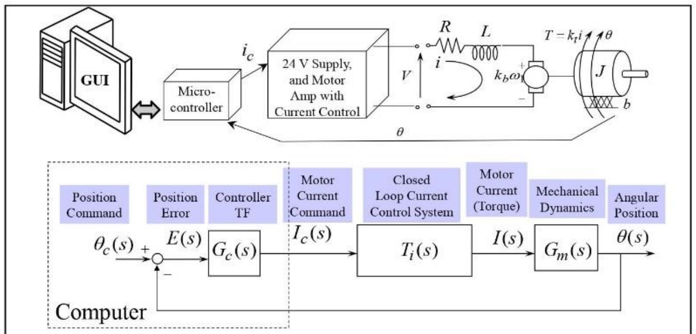
```
 xlabel('time (sec)'); ylabel('speed (rpm)');
```

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)');
```

Laboratory #8

In previous labs you experimented with a proportional (P) controller for position control in the Motorlab. We found that as we raised the gain of the P controller that we could obtain somewhat better control of the system, but that the improvement was limited. We could only raise the gain so much before the response became very oscillatory. Furthermore, the settling time could not be improved. Now you are to compare the proportional controller to a proportional-derivative (PD) controller. The PD controller adds a zero to the open-loop TF, changing the shape of the root locus. To obtain experimental data you should use the following three controllers.



$$G_m(s) = \frac{\theta(s)}{I(s)} = \frac{k_t k_{dr}}{Js^2 + bs}$$

$G_c(s) = K_p$, proportional controller

and

$G_c(s) = K_p + K_d s$, PD controller

$= K_d(s + K_p / K_d) = K_d(s+z)$

$J =$ motor inertia and collar (from motor lab handout)

$b =$ viscous friction coeficient (estimated to be 3×10^{-5} N·m·s/rad)

$k_t =$ motor torque constant (from motor lab handout)

$k_{dr} = 180^\circ / \pi\,(rad) =$ angular postion unit conversion

$T_i(s) = 1$ (amplifier assumed "fast")

$K_p =$ proportional gain of controller

$K_d =$ derivative gain of controller

$z = K_p/K_d =$ magnitude of controller zero

$\theta(t) =$ angular position (measured in degrees for this problem)

Three controllers for experimental data

1. P controller, $K_p = 0.001$ Amp/deg
2. PD controller, $K_d = 0.00007$ Amp·s/deg, $z = 10$ rad/s
3. PD controller, $K_d = 0.001$ Amp·s/deg, $z = 10$ rad/s

Obtaining Data From The Motorlab

One problem with real systems (rather than mathematical models) is saturation. When we use a derivative term in the controller, a step input will saturate the output of the controller in a real system. Therefore, we often do not get a good match between experimental results and theoretical models for a step response when derivative control is used. We will discuss this in the lab preparation. **DO NOT LET THIS DISCOURAGE YOU FROM USING DERIVATIVE CONTROL**. It can be very important in obtaining an optimal closed-loop system. Remember that step inputs are usually used as test signals – not as the actual commands in the operation of real systems such as machine tools, aircraft, etc. Because of this problem we will be using a triangle wave for the test command in this lab. **You should obtain the triangle wave response for the three different position control systems. Use a triangle wave with amplitude of 2000 degrees and a wave frequency of 0.5 Hz. You should obtain three different plots.**

Using MATLAB – *Complete Lab8Start.mlx first*

Using the "Sisotool" you should play with the systems to get a feel for the responses as the gains change. You should also use the Sisotool to get a feel for root locus. We will do an introduction to the Sisotool in lab. Another related MATLAB function is "rlocus" – try it.

- In the Sisotool, in the MATLAB workspace, or in the m-file, you should find the closed-loop poles and zeros of each of the three systems.
- Outside of the Sisotool, you should also obtain a single step response plot with the response of all three systems.

By hand, using the basic rules, you should draw two root locus plots.

- One plot should be for the P controller.
- The other should be for the PD controller with the zero at -10.
- On the plots clearly indicate the closed loop poles for the three different systems. (i.e. Where on the root locus are you?)

<u>Things to Turn In</u>

- A plot with all three step responses from the models.
- The triangle wave responses (use mlposplots or add code to the m-file).
- The root locus plots (by hand).
- Closed-loop poles and zeros for all three systems.
- M-file with comments
- Narrative on Canvas - Individually

Lab 8

Position control with PD controllers

Setup & Constants

These constants are duplicated from lab 7 since at many of you will have difficulty finding them. Not all of them will be used. You will need to recall these for the rest of the semester.

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

kt   = 0.05;                 % N-m/A
J    = 1.1e-5 + 0.19e-5;     % kg-m^2 or N-m-s^2/rad
b    = 3e-5;                 % N-m-s/rad
kdr  = 180/pi;               % deg/rad
krd  = 1/6;                  % rpm/(deg/s)
```

Transfer functions

Create the plant transfer function, G_m, and create the three required controllers - G_{c1}, G_{c2}, and G_{c3} using the form $G_c = K_p + K_d * s$. Study the equation block in the handout carefully to discover how to manipulate a given Kp and z into Kp and Kd. Then, apply feedback to close the loop on the three open loop systems, $G_m * G_{cx}$, assigning them to $T_1, T_2,$ and T_3.

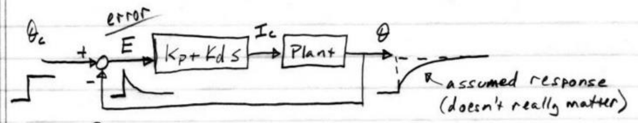
Step responses and plotting

Generate a 1 second step response for each closed loop system, T_x, and keep the resulting θ and associated time vectors in variables named th_x and t_x. Plot the three responses on the same figure. Use the help command if you have forgotten how to use step() or plot().

Difficulty with Step responses and controller derivatives

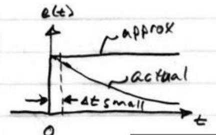
- The difficulty is that we often don't get a good match between the step response predicted by the model and the step response of the actual system when we use a derivative term in the controllor.
- DON'T let this stop you from using derivative control.
- Example: Suppose we use a PD controller

$$G_c = K_p + K_d s = K_d (s+z), \quad z = K_p/K_d$$

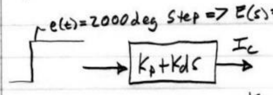


suppose $\begin{cases} -\theta_c = 2000\text{deg step} \\ -K_d = 0.0001\ A\cdot\text{sec/deg} \end{cases}$

Now approximate $e(t)$ (error) by a step to capture the initial part of $e(t)$

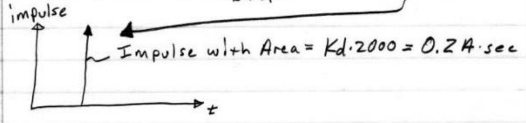


$e(t) = 2000\text{deg Step} \Rightarrow E(s) = \frac{2000}{s}$



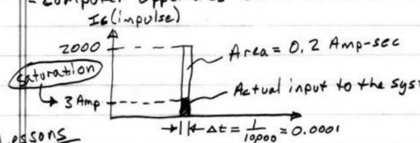
$$I_c(s) = E(s)\cdot G_c(s) = \frac{K_p \cdot 2000}{s} + K_d \cdot 2000$$

$$\mathcal{L}^{-1}\{I_c(s)\} = \underbrace{K_p \cdot 2000\, u(t)}_{\text{step}} + \underbrace{K_d \cdot 2000\, \delta(t)}_{\text{Impulse}}$$



Computer's Approximation of the Impulse

- Computer opperates on a clock with 10KHz frequency,



Lessons

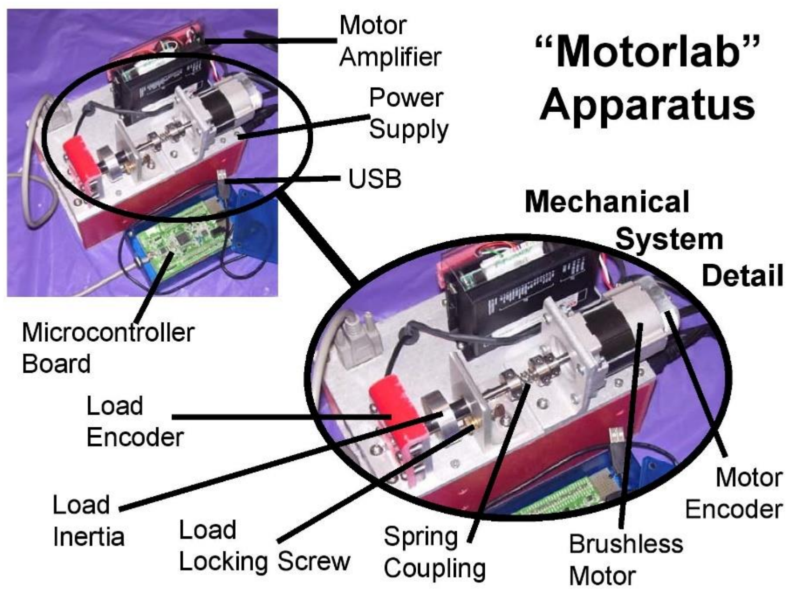
- Unless we use very small steps the actual energy that makes it to the system is much smaller than that predicted by the linear model, because of saturation.
- Saturation can also be a factor with large steps and proportional control. It is much easier to calculate when this will happen. $K_p \cdot \text{stepsize} = \text{saturation value}$
- Sometimes we want to use signals other than steps as the test signals.

Example C function for PID control

```
float simple_PID_controller(float error, float delta_time)
{
  float output, Kp=1, Ki=2, Kd=3, max_output=100;
  static float integral, last_error;                      // static vars for memory

  integral = integral + Ki*error*delta_time;              // numerical integration
  if (integral < -max_output) integral = -max_ouput;      // anti-integral windup
  if (integral >  max_output) integral =  max_ouput;      // anti-integral windup

  output = Kp*error + integral + Kd*(error-error_last)/delta_time;      //PID

  error_last = error;                                     // remember for next call

  return(output);
}
```

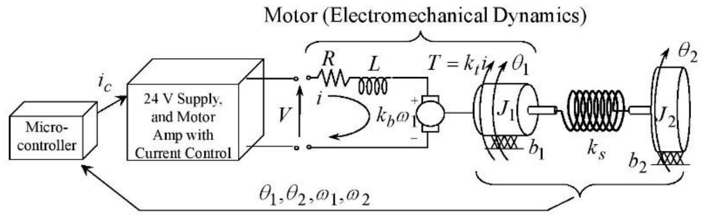
Appendix B – ME570 MotorLab Hardware Specifications

“Motorlab” Dynamics and Controls System



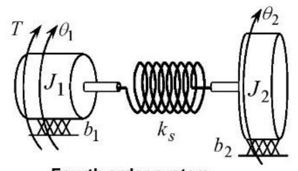
System Description

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 +/-10V 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.

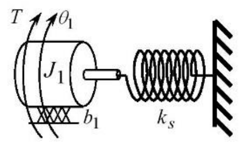

Motor (Electromechanical Dynamics)
R
L
$T = k_t i$
θ_1
θ_2
i_c
24 V Supply, and Motor Amp with Current Control
Micro-controller
V
i
$k_b \omega_1$
J_1
J_2
b_1
k_s
b_2
$\theta_1, \theta_2, \omega_1, \omega_2$

1

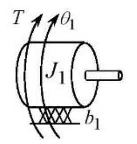
Several different configurations of the system can be utilized in experiments. Either sensor, the motor or load encoder, can be used for the feedback of the control loop. The selection is made in the software interface. The motor encoder is known as a "collocated" sensor since it is co-located with the 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.



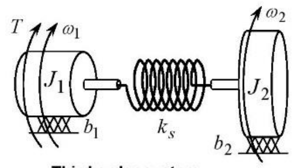
Fourth order system with a free integrator



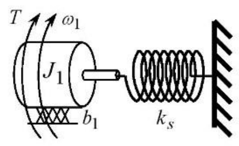
Second order system



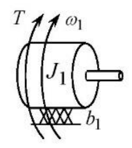
Second order system with a free integrator



Third order system



Second order system with a free differentiator



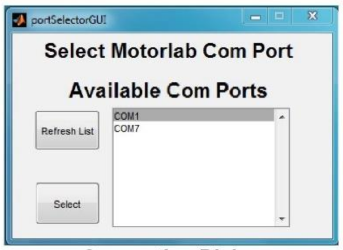
First order system

Software

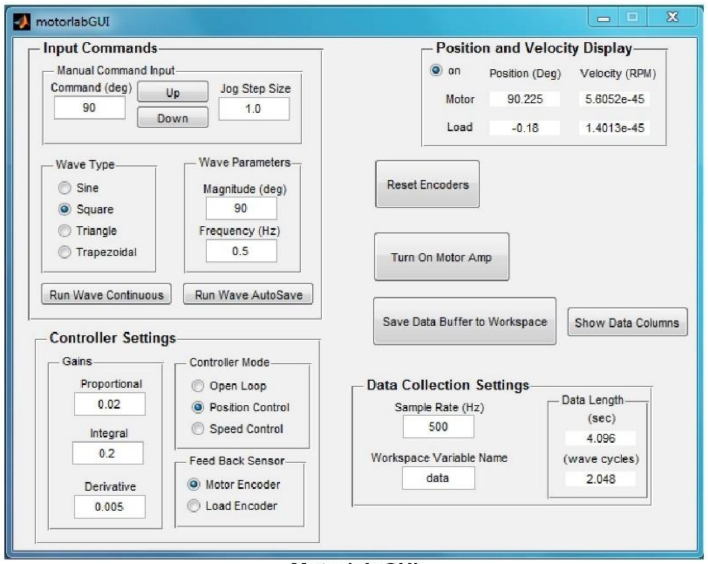
The software for the system can be found in the "c:\Motorlab" directory on the laboratory machines. All the needed Matlab functions can be found there. The software that is on the microcontroller is included in this directory in the motorlabRepo.zip file. This program is burned into the flash memory 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.



Connection Dialog


motorlabGUI

Input Commands

Manual Command Input: Command (deg) 90; Jog Step Size 1.0

Wave Type: Sine, [✓ Square], Triangle, Trapezoidal

Wave Parameters: Magnitude (deg) 90; Frequency (Hz) 0.5

Position and Velocity Display: [✓ on]

	Position (Deg)	Velocity (RPM)
Motor	90.225	5.6052e-45
Load	-0.18	1.4013e-45

Controller Settings

Gains: Proportional 0.02; Integral 0.2; Derivative 0.005

Controller Mode: Open Loop, [✓ Position Control], Speed Control

Feed Back Sensor: [✓ Motor Encoder], Load Encoder

Data Collection Settings: Sample Rate (Hz) 500; Workspace Variable Name data

Data Length: (sec) 4.096; (wave cycles) 2.048

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: mlolplots.m function: mlolplots(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: mlolplots(data); Does not scale the current command.
example: mlolplots(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);

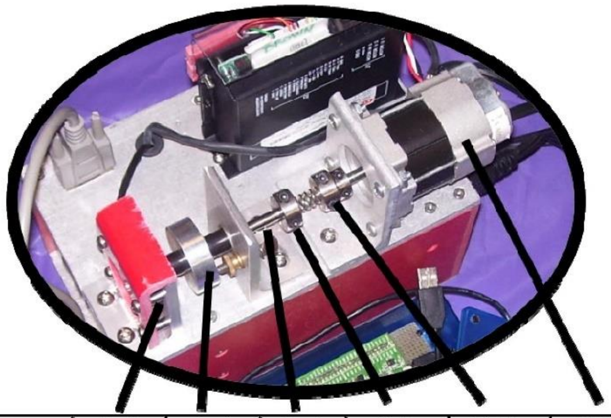
File: trapprof.m function: [x,v,t] =trapprof(DX,Vmax,Amax,DT) Trapezoidal-velocity motion profile generation
Outputs: x=position vector, v=trapezoidal velocity vector, t=time vector
Inputs: DX=distance to move, Vmax=maximum velocity, Amax=maximum acceleration, DT=time step for outputs
example: [x,v,t]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



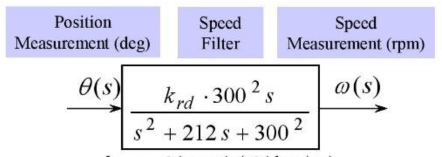
Object	Load Encoder	Load Wheel	Load Shaft	Single Shaft Collar	Double Shaft Collar	Motor Rotor and Motor Encoder
Inertia(g-cm^2)	0.8	32	1.4	15	19	110

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.


Position Measurement (deg)
Speed Filter
Speed Measurement (rpm)
$\theta(s)$
$\frac{k_{rd} \cdot 300^2 s}{s^2 + 212 s + 300^2}$
$\omega(s)$

$$k_{rd} = 1(rpm)/6(\deg/\ s)$$

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.

$$T_i = \frac{\omega_n^2 (s+z)}{z(s^2 + 2\varsigma\omega_n s + \omega_n^2)} \quad \text{or} \quad \begin{cases} T_{idelay} = \dfrac{\omega_n^2 (s+z)}{z(s^2 + 2\varsigma\omega_n s + \omega_n^2)} e^{-t_d s} \\ T_{ipade} = \dfrac{\omega_n^2 (s+z)}{z(s^2 + 2\varsigma\omega_n s + \omega_n^2)} \cdot \dfrac{s^2 - 6s/t_d + 12/t_d^2}{s^2 + 6s/t_d + 12/t_d^2} \end{cases}$$

where :

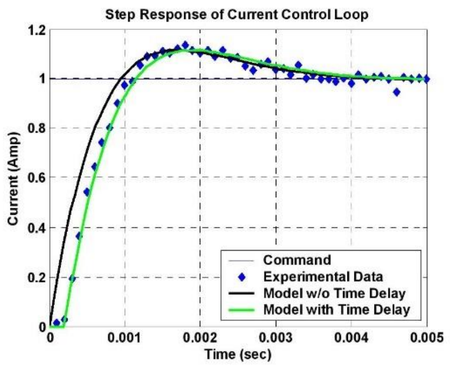
$z = 170 \cdot 2\pi$ *(rad/sec)*

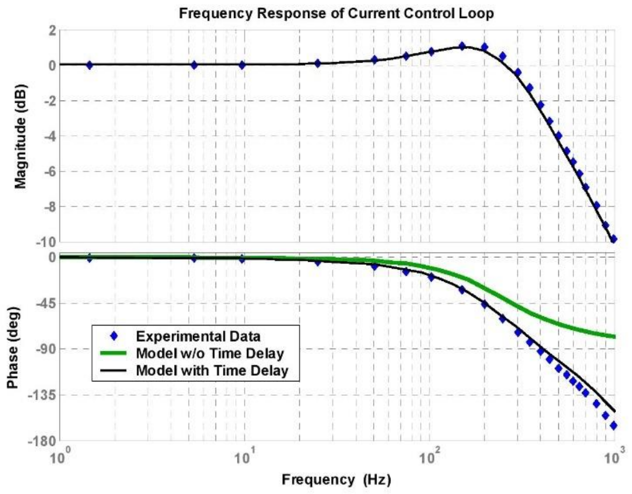
$\omega_n = 230 \cdot 2\pi$ *(rad/sec)*

$\varsigma = 0.8$

$t_d = 0.0002$ *(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

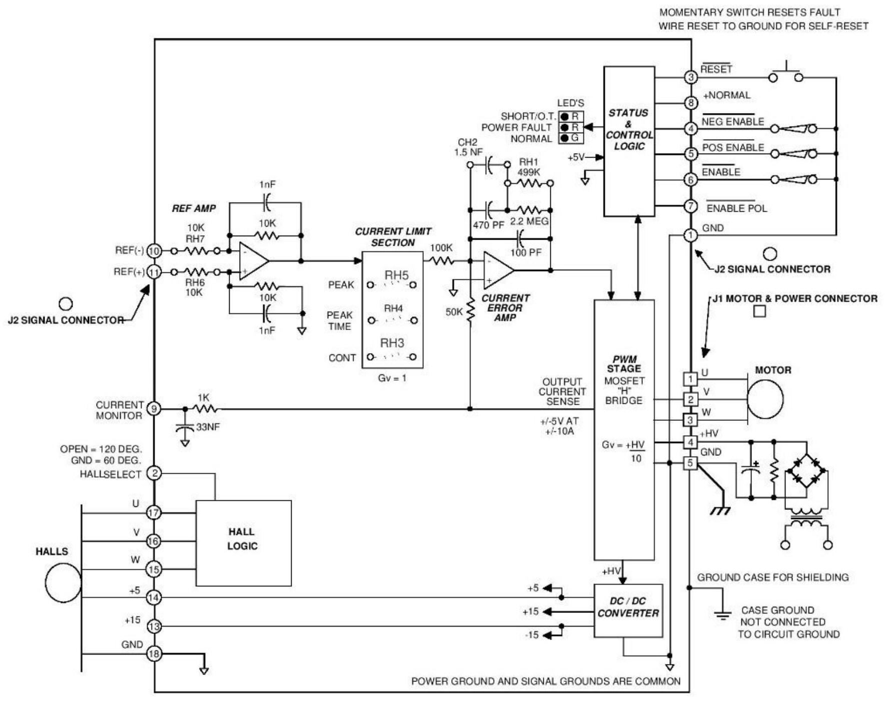
- **NO POTS: Internal component header configures amplifier for applications**
- **Conservative design for high MTBF**
- **Low cost solution for small brushless motors to 1/3 HP**

PRODUCT DESCRIPTION

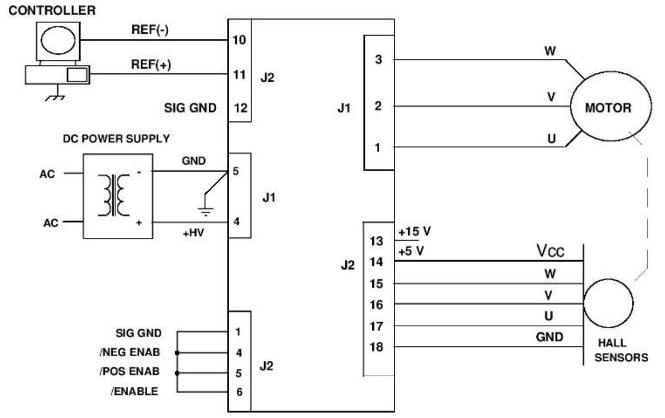
Model 503 is a complete pwm servoamplifier for applications using DC brushless motors in torque (current) mode. It provides six-step commutation of three-phase DC brushless motors using 60° or 120° Hall sensors on the motor, and provides a full complement of features for motor control. These include 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.

FUNCTIONAL DIAGRAM



TYPICAL CONNECTIONS



Copley Controls Corp.

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

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 10kΩ/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.

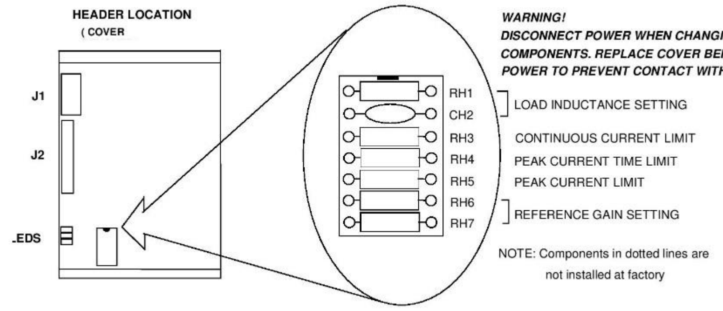
	J1-1	J1-2	J1-3
#1	U	V	W
#2	V	W	U
#3	W	U	V
#4	U	W	V
#5	W	V	U
#6	V	U	W

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.

APPLICATION INFORMATION (CONT'D)

COMPONENT HEADER



WARNING!
DISCONNECT POWER WHEN CHANGING HEADER COMPONENTS. REPLACE COVER BEFORE APPLYING POWER TO PREVENT CONTACT WITH LIVE PARTS.

CONTINUOUS CURRENT LIMIT (RH3)

Icont (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

Ipeak (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

Tpeak (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.

Copley Controls Corp.

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

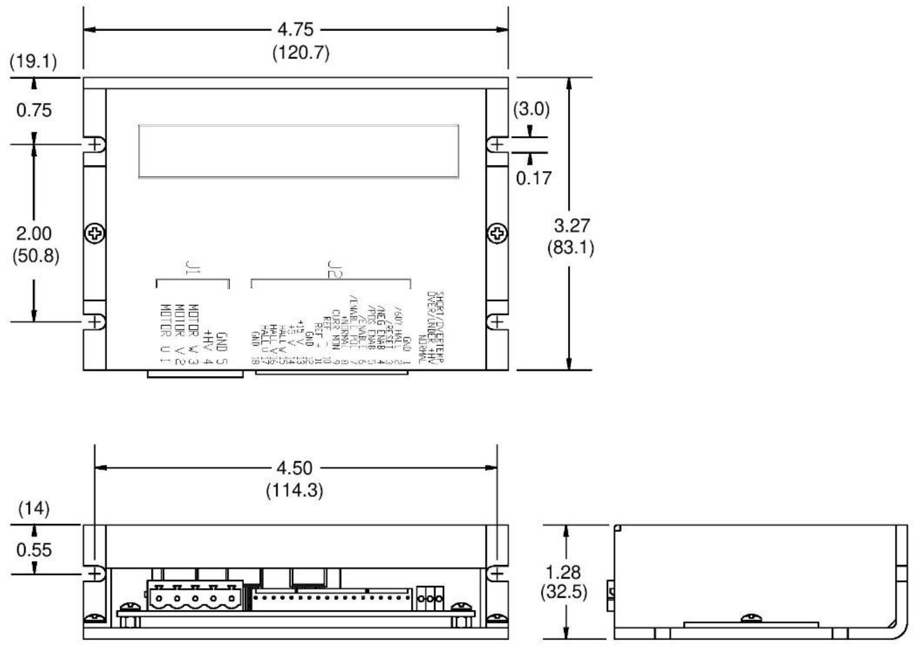
TECHNICAL SPECIFICATIONS

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

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

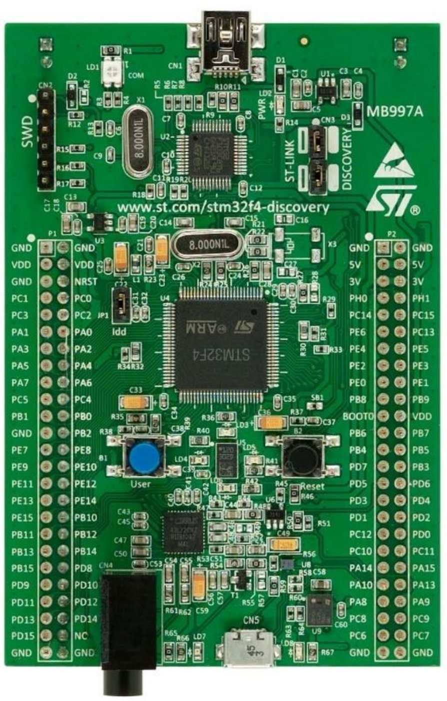
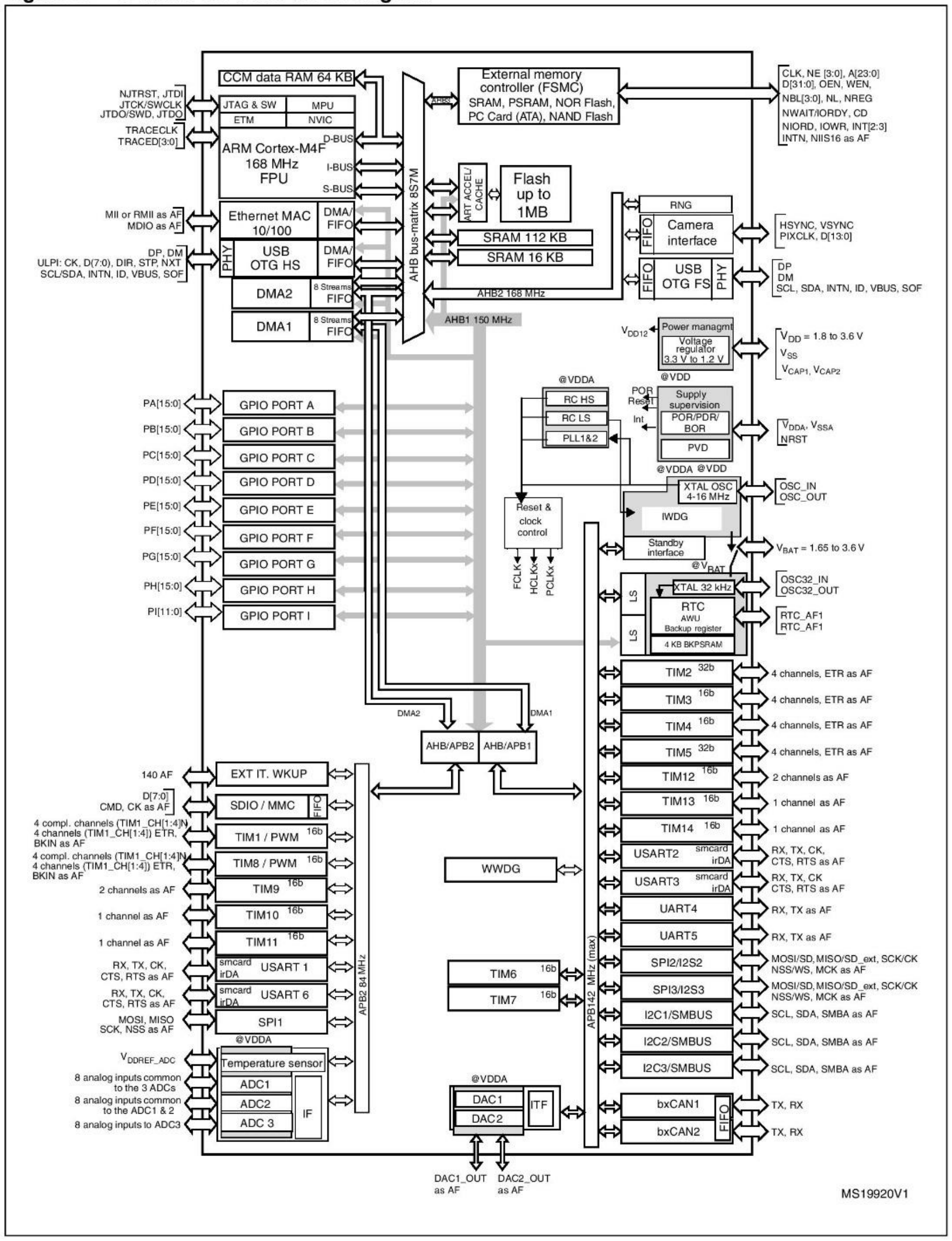


Figure 6. STM32F407VGT6 block diagram

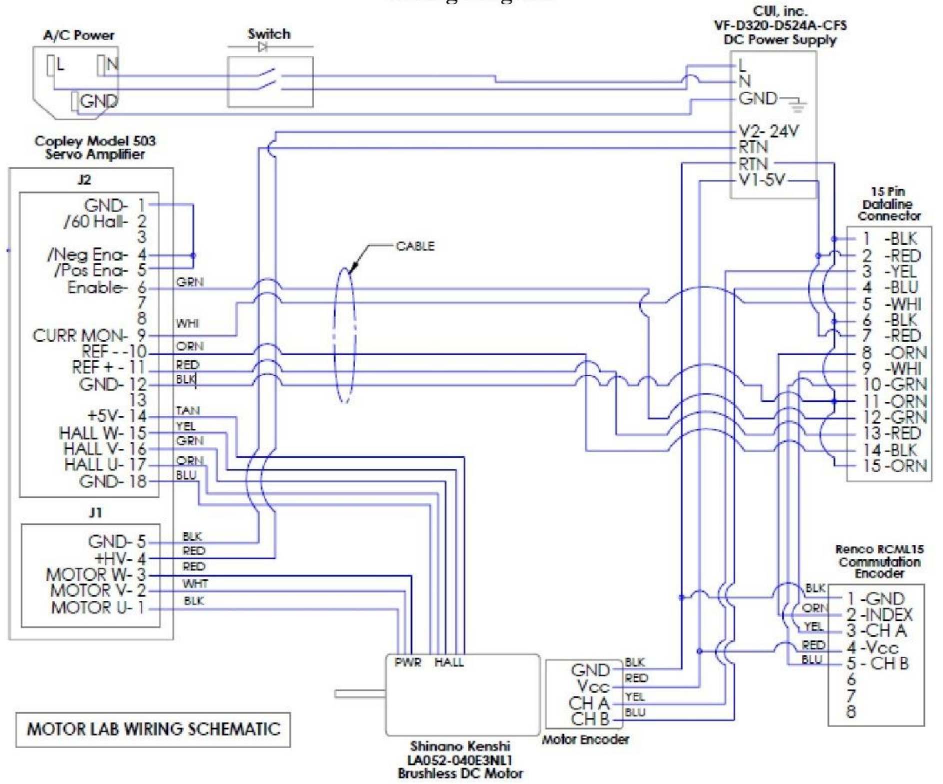


Wiring for the "Motorlab" Apparatus

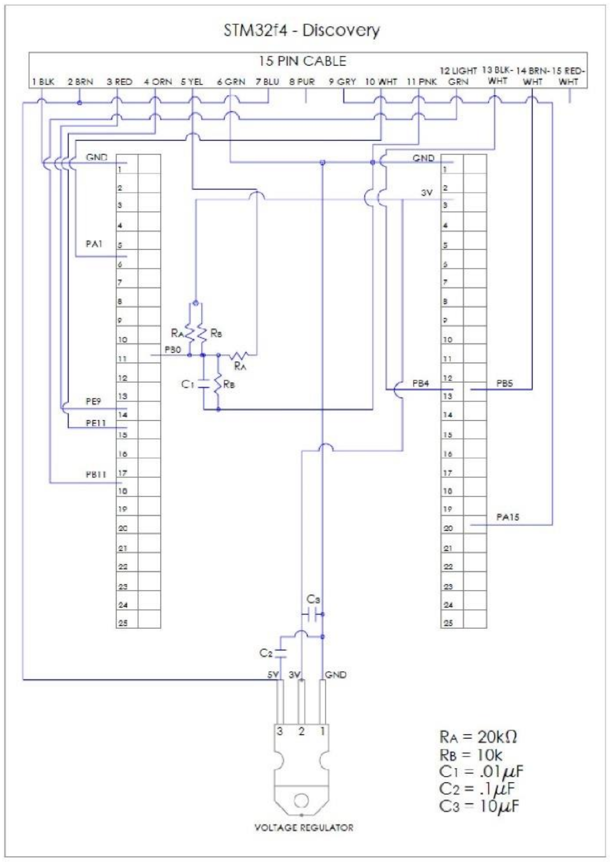
15 Pin Microcontroller Connections

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	9	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



STM32f4 - Discovery
15 PIN CABLE
1 BLK
2 BRN
3 RED
4 ORN
5 YEL
6 GRN
7 BLU
8 PUR
9 GRY
10 WHT
11 PNK
12 LIGHT GRN
13 BLK-WHT
14 BRN-WHT
15 RED-WHT
GND
PA1
PE9
PE11
PB11
PB0
RA
RB
C1
GND
3V
PB4
PB5
PA15
C3
C2
5V
3V
GND
3 2 1
VOLTAGE REGULATOR
RA = 20kΩ
RB = 10k
C1 = .01μF
C2 = .1μF
C3 = 10μF