'Helicopter' Control Theory Demonstrator

E90 Final Report

Swarthmore College Department of Engineering

Emery Ku, Class of '05

Advised by Professors Cheever and Orthlieb

05/2005

Table of Contents

Abstract

This final report summarizes the work done by Emery Ku ’05 on a ‘Helicopter’ Control Theory Demonstrator in the spring of 2005. Much of the content of this document is dedicated to the discussion of the design procedure and the documentation of system components. The system was built to resemble a helicopter and requires a two-input, two-output controller. One of the primary goals was to construct a new and unique structure which would ultimately be used as an educational tool for those studying control theory; it is currently in working condition. A fully successful controller has yet to be implemented, but a great deal of progress has been made in understanding the system’s inherent complications.

Introduction

Hundreds of years before Leonardo da Vinci was even born, the idea of vertical flight had already been implemented. A Chinese toy known as a "bamboo dragonfly" became widespread circa 400 BC: "the earliest versions of the Chinese top consisted of feathers at the end of a stick, which was rapidly spun between the hands to generate lift and then released into free flight," (http://centennialofflight.com/history/helicopter.html). More than two thousand years later, this simple toy would serve as a great inspiration for several key contributors to early modern helicopter design.

A component common to all modern helicopters is an advanced control system. The propeller that is responsible for 'lift' simultaneously generates a torque which causes the body of the aircraft to rotate. This is counteracted by the tail rotor. To have a pilot try to manually compensate for this constantly varying characteristic of the system while trying to stay in the air would be incredibly dangerous. Thus, a controller is required to counteract the natural tendency to horizontal rotation or 'yaw'. This challenge was the inspiration for my E90 Senior Design Project.

The primary goals of my E90 project are 1) to produce a system which emulates the functionality of a helicopter to be used as an educational tool for future classes of engineers, 2) to implement a two-input/two-output controller to govern the motion of the 'helicopter', and 3) serve as a demonstration of control theory principles.

While the system has been fully constructed and developed, a number of issues have come up along the way. Some have been addressed and resolved while others remain a barrier to the utmost realization of my project goals. At this point in time, parallel PID controllers have been implemented for both *Theta* (yaw) and *Phi* (pitch) directions. The system went through a serious redesign after which progress has been difficult. Before that modification, results were respectable for the *Phi* direction. However, due to time varying behavior in the system, consistent results have yet to be achieved.

System Design Overview

The salient features of a helicopter are its ability to hover vertically and rotate horizontally. In keeping with a system analogous to a real helicopter, I decided to use two motors/propellers, the larger of which would be responsible for generating lift. Given that the primary purpose of this project is to create a system which will be applied in an educational setting, safety and especially system hardiness are of great concern.

The idea of a counter-balanced arm which is free to rotate horizontally and vertically is an appealing adaptation of a true helicopter for a number of reasons: 1) the two degrees of freedom available in my system create the necessity of a two-input/two-output controller; 2) because the lever-arm rotates about a fixed point, there is no danger of the apparatus 'flying away' and doing damage to itself or to a student; 3) counter-balancing the lever allows for reduced power requirements for the motor responsible for generating lift.

With the general vision of a counter-balanced swinging arm propelled by ‘fans’ in mind, the first step to realizing the project was to order motors and propellers. Once acquired and tested, the mechanical capabilities these components dictated the design and weighting of the arm. In addition, the electrical requirements and characteristics of the motors determined the specifications of the amplification component. The details of the rotating parts of the arm were finalized when rotary sensors were purchased. The electrical characteristics and requirements of the electrical components (amplifier and sensors) determined the functionality of the microcontroller; this chip facilitates communication to and from the computer. Only with all of these components in place was it possible to begin designing a controller in MATLAB.

A graphical outline of the components’ prerequisites in the design process is shown in the flow chart below:

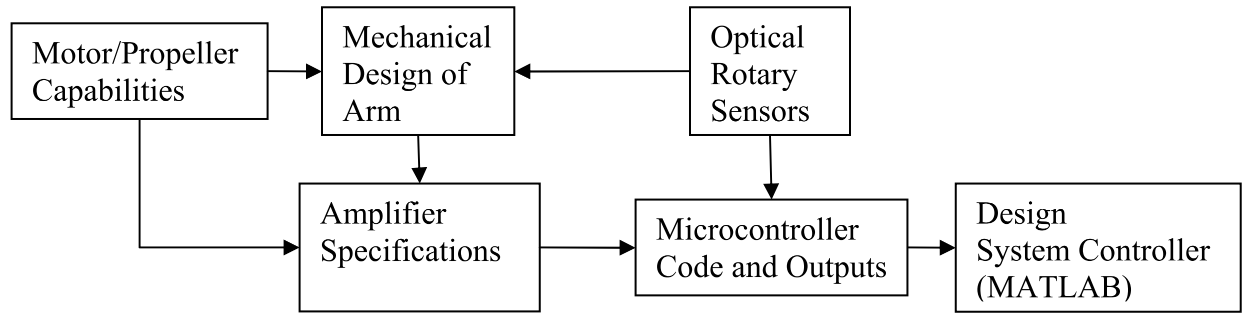

Motor/Propeller Capabilities
Mechanical Design of Arm
Optical Rotary Sensors
Amplifier Specifications
Microcontroller Code and Outputs
Design System Controller (MATLAB)

Figure 1. Flow chart outlining system design.

System Components

The system can be broken down into three primary domains: Mechanical, Electrical and Computer (or software). Each component is outlined here in terms of its function within the context of the entire system. The following flow chart summarizes the path of information through the system. Note that there are two arrows into and out of each component because the system moves in both *theta* and *phi* directions.

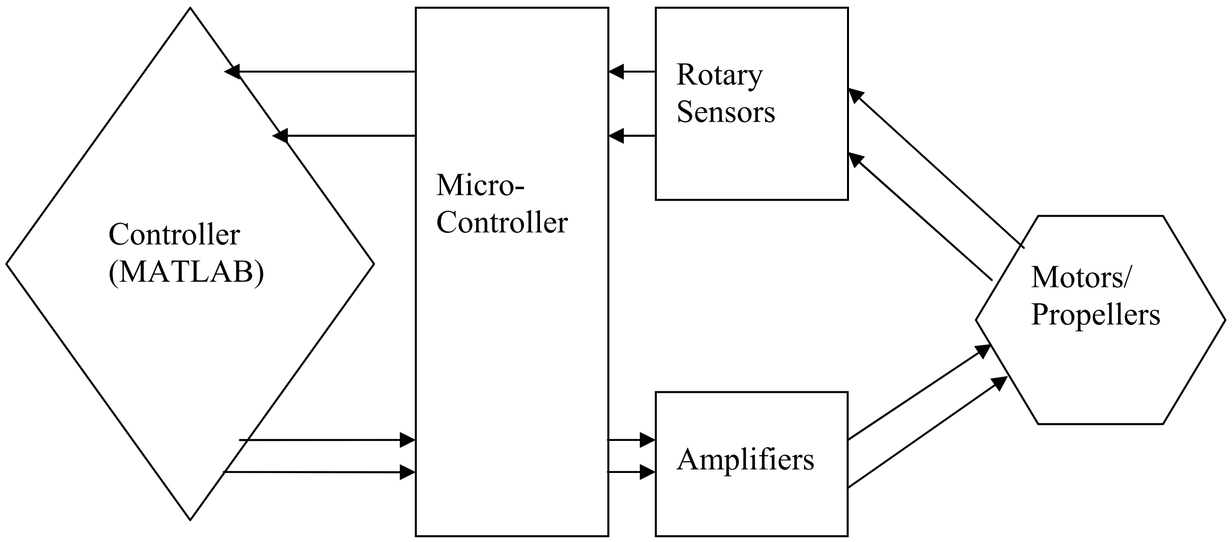


Figure 2. Flow of information within the system. Computer components are diamond-shaped, electrical components are rectangular, mechanical components are octagonal.

Mechanical

The primary components of my project that fall under the mechanical category are the motors/propellers, and the physical apparatus which consists of the swinging arm and its support structure. The latter allows for the physical freedoms and restrictions of the system, while the former drives its motion.

Motors and Propellers

Motors were chosen such that they would be easily driven and lightweight. I selected a pair of identical ±12V motors as this input range is very common. In addition, I consciously selected motors with the ability to spin in either direction as I felt it would leave fewer restrictions later in the design process.

After ordering several different propellers, I adapted the motor shafts to be able to hold and swap out these different models. In the end, a three-rotor 12" diameter by 6" pitch plastic propeller was selected for the *phi*-direction, and a dual-rotor 6" diameter by 3" pitch was fixed permanently to its motor shaft for the *theta*-direction. These choices were made through a process of measuring the ranges of force that could be easily produced by the various motor/propellers combinations without overtaxing the motor.

Once both motors and propellers were acquired, they were fitted to each other in the shop. The tail rotor was fixed permanently to its motor because I did not anticipate the need for a great deal of force in the *theta* direction. However, I was less certain about the output force requirements of the motor/propeller 'fan' which would produce lift (motion in the *phi* direction). As a result, I designed the motor shaft such that different propellers would be interchangeable.

The electrical input and mechanical output characteristics of the variously configured 'fans' were measured by holding the fans in a block which was then taped to a digital scale. The following figures illustrate the various Force vs. Input Current functions for different motor/propeller combinations:

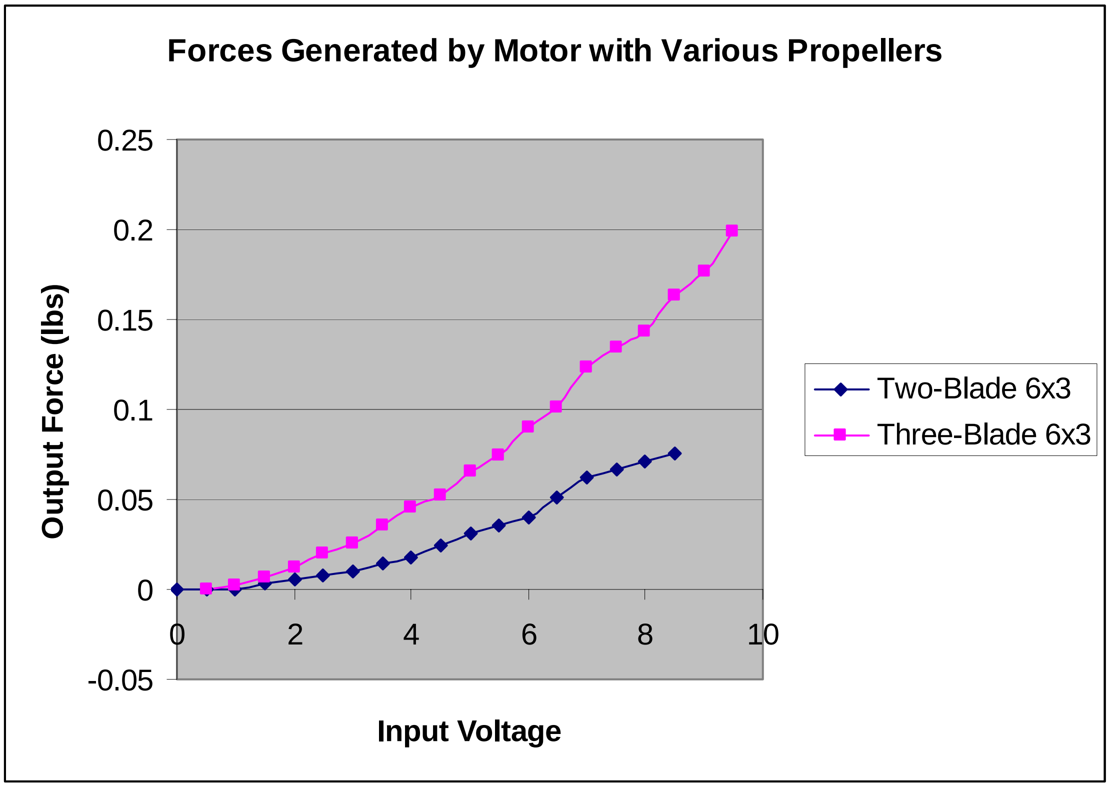


Figure 3.1. Force vs. Voltage curves for DC motor with two different propellers.

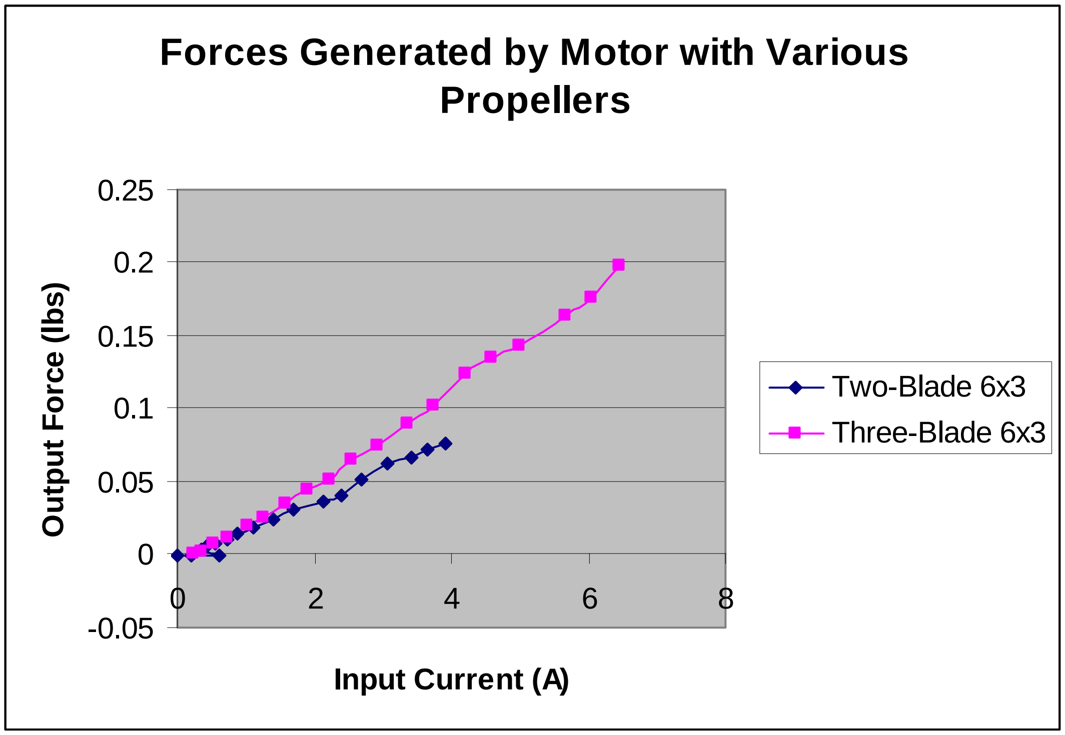


Figure 3.2. Force vs. Current curves for DC motor with two different propellers.

I performed a similar test on the combination of the motor with the 12x6 rotor. However, the results were much noisier towards the upper end of the voltage range; this was likely due to the higher air flow over the digital scale. What was significant was that the combination of the motor with the 12x6 rotor was able to produce a force of ~0.25 lbs at 8V. This option was chosen as the higher force at lower voltages allowed for a wider range of potential forces.

Swinging Arm and Support Structure

The swinging arm and its support structure constitute the body of the system. I decided to build a metal support system because it was my opinion that a PVC base would be more prone to damage and appear less attractive. The aluminum components of the support structure were shaped to fit the *theta* direction encoder (which would simultaneously allow for free rotation horizontally and measure its angle) and also the swinging arm. The set screw that holds the upper aluminum support piece to the encoder shaft is prevented from rotating past one revolution by a perpendicular machine screw. This is to prevent the wires from becoming entangled.

The swinging lever arm is constructed from 2"x3/4" cross-section pine. This wood was selected because of its low weight and ease of construction. My initial design specified an arm 35 inches long, and 2 inches wide with the pivot at 19 inches distant to the end where the *phi* direction motor was held. This would accommodate the range of forces that the *phi* direction motor could output. Wood was glued to the two ends to produce two-cubic inches of pine which were then drilled out hold the motors (one pointing upwards, the other pointing horizontally).

The transfer function of the *phi* direction motion of the arm was determined by sending step inputs to the system and recording the steady state values. The result was then fitted to an exponential function. In turn, this function was solved for the required input voltage given a desired output angle for application in the PID controller.

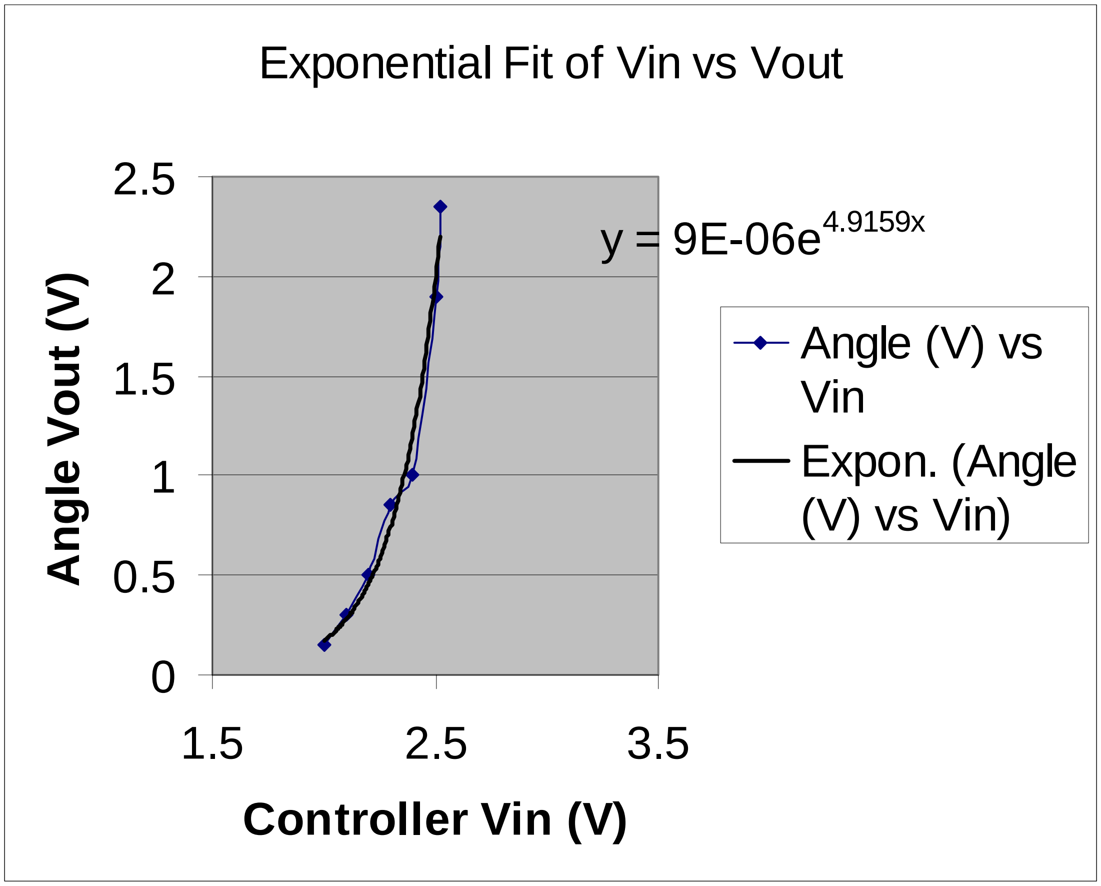


Figure 4. Numerical solution and exponential of transfer function in *phi* direction given step inputs.

The arm was eventually redesigned to new physical specifications (for reasons that will become apparent later: see **System Development**: Hardware). The final arm was 14.5" long with a ¾" x ¾" square cross-section. The motors are fixed to the ends with adjustable metal straps. This arm also has the ability to slide along the board which rotates in the support structure. This makes varying the arm's balancing much more convenient.

Electrical

The electrical components of this system consist of the rotary sensors, microcontroller, and the amplifier (with power supply). These are responsible for facilitating communication between the computer and the physical apparatus.

Sensors: Optical Rotary Encoders

There are numerous ways to measure rotation. One common method is using a dial-based potentiometer. I chose to use digital rotary encoders in order to minimize noise and maximize accuracy and precision. The two encoders I chose were *BEI Duncan*'s EX-11 and MX-15 (male and female versions). These encoders output 1024 pulses per revolution in two channels (90 degrees offset to discern direction of rotation); see figure below:

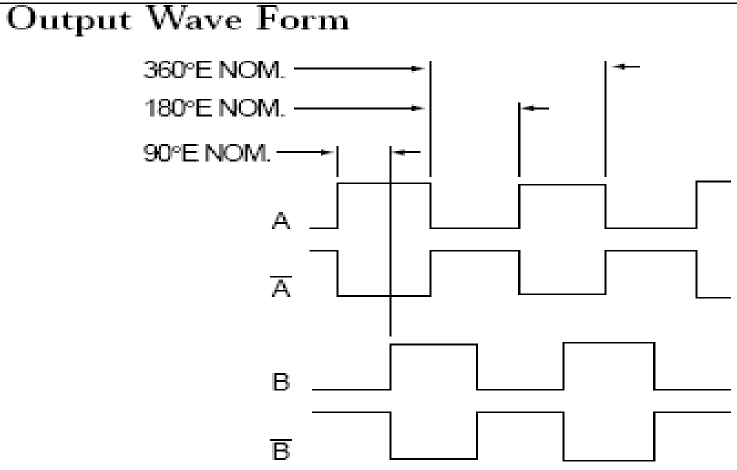
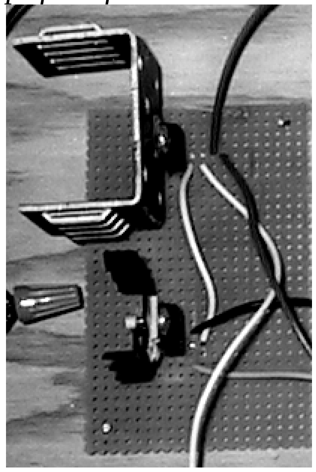
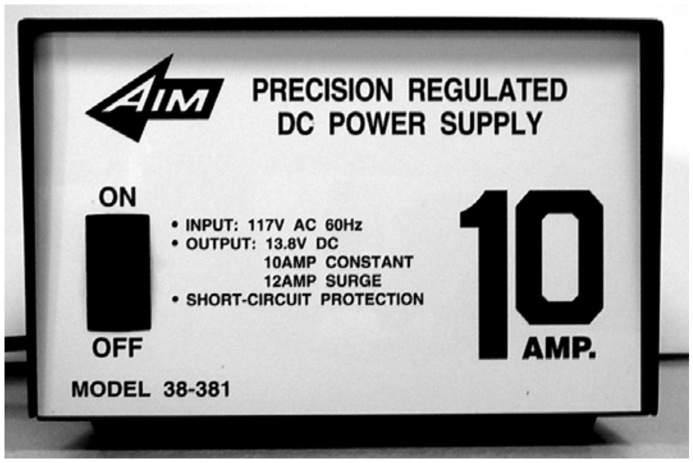


Figure 5. Output from rotary encoders.

I opted for this level of precision in order to acquire better data, and I felt confident that the microcontroller decoding its output would be fast enough to catch these cycles without missing any pulses. For complete technical specifications, please refer to **Appendix A**: Datasheets, Rotary Optical Encoders.

Amplifiers for Motors





In order to power the motors, an amplification system is necessary. The requirements of this component were set by the motors. I anticipated (given prior measurements) that the absolute maximum continuous current that would be drawn by both motors was close to 7 or 8 amps. In addition, the motors are designed for 12V continuous (max), so it made sense to use a power supply that would be able to match these characteristics. The chosen power supply outputs a continuous 13.8V, with a max continuous current rating of 10A. In addition to the fuse internal to the power supply, I added a single 8A fuse between the power supply and the motors in order to prevent potential damage to the apparatus.

In order to maximize efficiency and simplicity, a pulse-width modulation (PWM) mode of operation was chosen to power the motors. This method minimizes the energy wasted in heat in the amplifier circuit. In my implementation of PWM, a MOSFET (Metal Oxide Semiconductor Field-Effect Transistor) turns on and off quickly which determine how much current flows through the motor. A 'flyback' diode is placed in parallel with the motor to reduce the effect of the motor's internal inductance (see **figure 6** below).

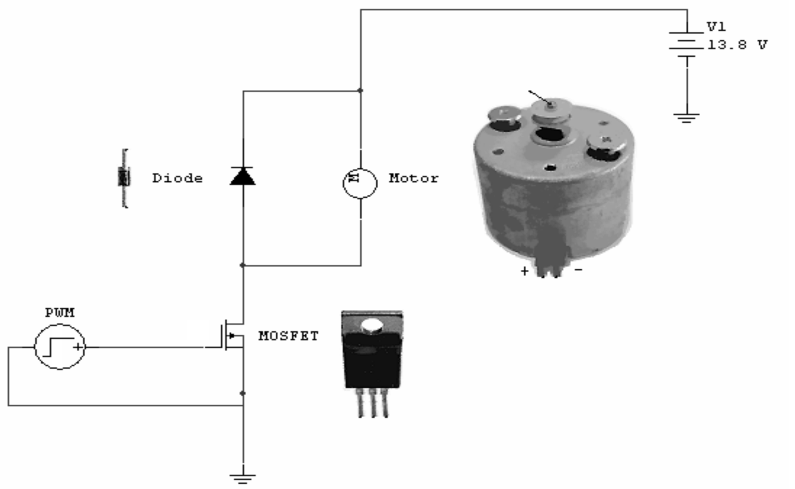
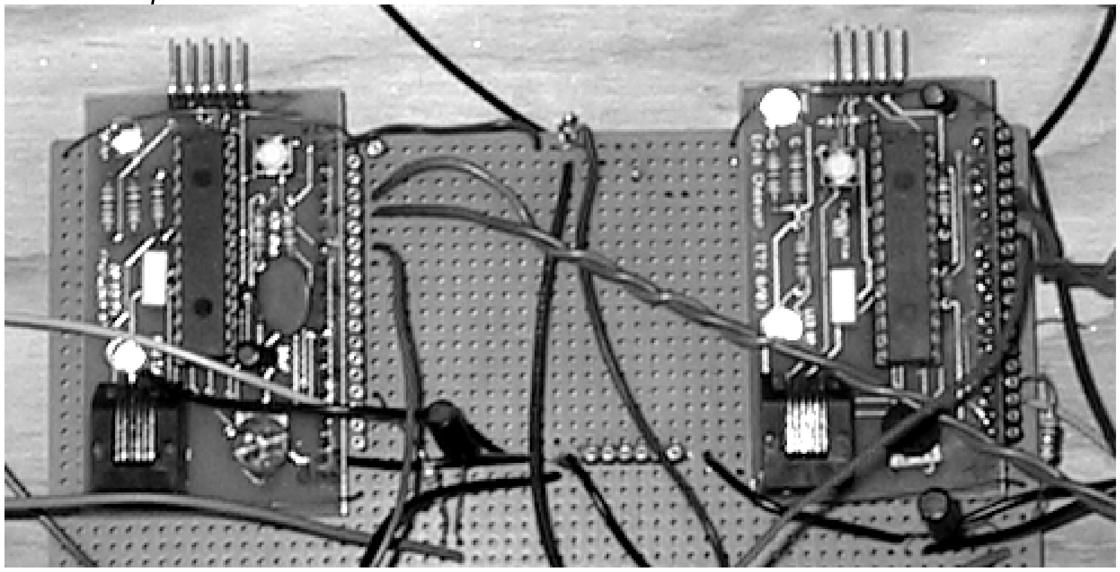


Figure 6. Amplifier circuit. V1 is the power supply.

The components that I eventually chose as permanent additions were the *10CTQ-150* Schottky diode and the *RFD16N05L* logic-level MOSFET. For complete technical details, please see **Appendix A**: Datasheets. The design processes that led up to the selection of these particular are discussed in **System Development**: **Hardware**.

Microcontroller for D/A and A/D Conversions



The microcontroller is responsible for performing the critical function of converting the signals between the computer and the system into readable data. There are two microcontrollers: one for the *phi* direction and the other responsible for the *theta* direction. The microcontroller is a necessary component because the computer's data acquisition board (accessed through MATLAB) is only capable of sending and receiving 0-5V data, and the amplifier must use PWM.

Given that the rotary encoders each output two channels of pulses, the microcontrollers must take this information, track the change in angle, and convert this data to a readable signal from between 0-5V. At the same time, the MOSFET in the amplifier only accepts TTL (transistor-transistor logic, a signal of either 0V or 5V); the microcontroller must convert a controller voltage from the computer to a PWM signal operating between 0V and 5V. For details on how this is accomplished, see **System Components**: **Computer**, *C-Code for Microcontroller*.

The microcontroller I chose for this purpose is the PIC16F873A. I chose it in part due to its availability and convenience; more importantly, it met the I/O requirements for my system. In particular, there are three inputs (two channels from the rotary encoders, the third is the computer's controller voltage) and two outputs (one to the amplifier, the other to the computer). The I/O functions of the PIC microcontroller is summarized below.

I/O Function	**I/O Pin Type**	**Micro-Controller Pin Address**	**Pin Number on PCB** (Pin 1 = 5V)
Input Encoder Channel A	Digital Input	RB0	4
Input Encoder Channel B	Digital Input	RB1	5
Input Motor Control Voltage	10-bit Analog-to-Digital	RA0	7
Output Angle Voltage	PWM	RC2/P1	16
Output Motor Control	PWM	RC1/P2	17

Figure 7. Table of microcontroller I/O functions and pin locations.

One important matter to keep in mind is that the microcontroller does not have analog outputs. Subsequently it cannot strictly output an analog voltage. However, the PWM channels can be low-pass filtered to provide a varying analog signal. This is done with a simple R-C circuit.

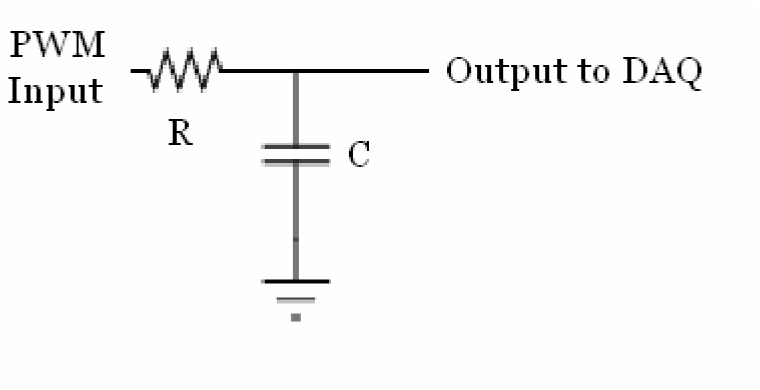


Figure 8. R-C low-pass Filter to convert PWM to a DC signal.

Through an iterative process, it was determined that optimal values for the resistor and capacitor would be R = 10kΩ and C = 33μF, given a 51.2 μs period.

A schematic of the circuit is shown here:

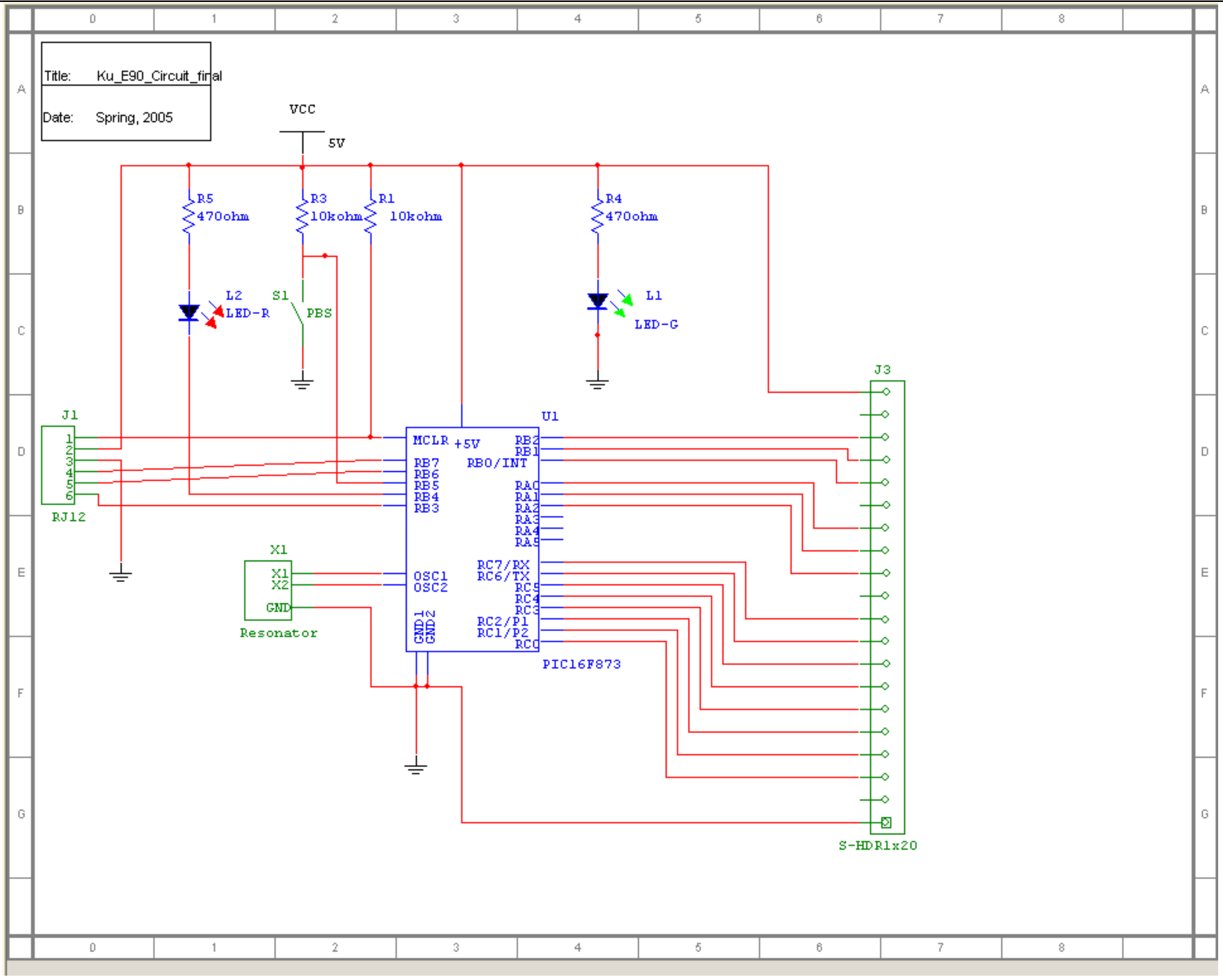


Figure 9. Microcontroller schematic.

The power supply circuit for the microcontrollers is not shown in Figure 9 (it is understood as internal to VCC); it is a very simple circuit:

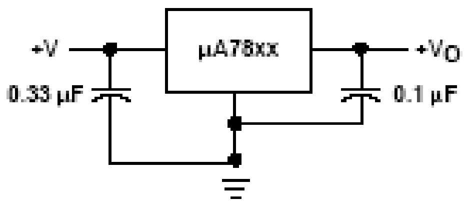


Figure 10. Power supply for microcontrollers. +V is 6VDC, V_O is 5VDC.

Computer

The computer and software components of the system are what determine how the system reacts. The microcontroller code determines how the chip parses the incoming signals and how to output appropriate information. The exact code can be found in **Appendix B**: Computer Code. The following section describes the processes that the code executes.

C-Code for Microcontroller

As discussed above, the microcontroller must provide important I/O functions for the system (see **Figure 7** for details). I will discuss the functions of the code in the order that they appear, as divided by commented section headers:

<u>//Things we need.</u>

The #include and #use lines simply tell the compiler what components are necessary for this PIC chip. Note that the library 16F873.h was modified to use 10-bit analog-to-digital conversion.

<u>//Define quadrature-based states as grey-code.</u>

Here the 4 possible states of the two channels from the encoder are defined. They are: 1) both A and B low, 2) A high, B low, 3) A and B high, 4) A low, B high. A grey code (only one bit flip at a time) is used to reduce potential errors.

<u>//Inputs</u>

This section defines which pins are used as inputs by default. The fixed_io setting saves processing time later.

<u>//Outputs</u>

This section is commented out because the PWM channels are set as such by default.

<u>//Keep track of the angular position (10-bit)</u>

The angular position variable 'angle' is stored as a 10-bit integer, as is the 'control' variable. This is setup in this manner so that the output to the PWM channel can be easily determined (it also ensures that there is enough resolution to accommodate the output format).

<u>main{}</u>

This is where the 'program' operates. More I/O pins are setup at the top, and the PWM pins initialized. The timer responsible for PWM outputs is set to a 20 kHz cycle and the angle variable is initialized to 0. Finally, an infinite loop is started which compares the current state with the last to determine which way the encoder is turning, redefines the old state for the next cycle, reads the input from the motor control channel (the 10-bit A/D conversion takes places there) and sends outputs according to which way the encoder is turning and what input the microcontroller is receiving.

In order to maximize the resolution available for each direction, the amount added to angle with each pulse and how much it is divided by in determining the PWM output is carefully chosen. For example, the *phi* direction has a range of rotation of approximately 70 degrees, whereas the *theta* direction is closer to 330 degrees. As a result, the incremental step-size for the *theta* direction is ¼ the size of the incremental step-size for the *phi* direction. This maximizes the resolution available given a 10-bit variable.

MATLAB Script to Implement Controller Algorithm

The MATLAB script is critical for both administering data I/O and also implementing the controller. As with the above C-code, I will describe the functionality of the script which is delimited by the comment titles. For specific code and detailed comments, refer to **Appendix B**: Computer Code, *MATLAB Script*.

<u>%%Setup the input channel</u>

Here the input channel is initialized. The number of seconds of the run is set at the top, in addition to the Sample Rate. While there is no hard upper limit, there is no great benefit of running the test at a very high sample rate. Anything above 20Hz will probably yield good results as that should be fast enough to simulate a continuously sampled system. In fact, setting the Sample Rate too high can quickly fill the input buffer and cause trouble. However, the greatest irritation will probably be the increase processor usage at the end of the data run when it has to plot a great many data points.

<u>%%Setup the output channel</u>

The National Instruments Data Acquisition ('nidaq') communication protocol still remains standard with the boards Swarthmore College's Department of Engineering currently uses. To avoid timing problems, the output sample rate is equal to the input sample rate.

<u>%%Initialize I/O</u>

Here we add channels and set acceptable ranges. The input range should be set to ±5V; other settings seem to upset MATLAB or the NIDAQ board.

<u>%%Pre-Set Output</u>

An initial output is sent to the motors in order to break past the static friction of the motors. This also slightly decreases the time delay at the start of the data acquisition run.

<u>%%Predefine variables & Initialize Arrays</u>

Arrays which do not have a defined size must be initialized as empty before they can be dynamically filled. Some variables have to be zeroed.

<u>%%Goal Angles (and voltages)</u>

Here the goal angles are entered. This angle is then converted to a goal voltage to be compared with the angle output from the microcontroller.

<u>%%Set Controller Parameters</u>

In my PID controller, I have two different proportional gain constants. The implementation is as follows:

```
        (Error)*[abs(Kp_a*sin(phi_desired)) + Kp_b*cos(phi_absolute)]
```

This helps account for the tendency of the system to get stuck at the top (see **System Development** for details on this problem). Kp_a diminishes with the sine of the angular difference between the current position and the goal angle. The Kp_b term varies with the cosine of the angle of the current position as measured from horizontal (zero degrees).

%%Get ready...

Trigger the data acquisition object.

%%Loop until time t

This is the main loop that governs data acquisition, controller calculations and sending output signals. Inputs and outputs are stored in pre-defined arrays, in addition to the components of the PID controller. Outputs are checked for sanity; if they exceed safe values, they are clipped and set to the maximum or minimum allowable value.

%%Post Data-Acquisition Outputs

This is an attempt at preventing the system from doing undue damage to itself, and also an attempt at resetting it to a 'zero' position.

%%Free up any memory that we used.

MATLAB's data acquisition code (when used with NIDAQ) seems problematic in that if you try to generate a new analog input or output object before the old one has been cleared, the I/O hangs altogether and requires that you restart your shell. Nonetheless, a short script at the beginning of my code looks for and tries to deal with the problem ahead of time.

%%Plot the data

This set of plots I have found very useful to debugging the controller's output. In this figure, each component of the controller is plotted against time along with the total error and the total output.

%%Optional plots

In this figure, angular positions and goals versus time are displayed, in addition to the total output of the controller. This set of commands is generally not executed to save processor time.

System Integration

This section describes in detail the working relationships between the components of each domain necessary to the functioning of the apparatus. This system is in some ways analogous to a biological organism: the mechanical components represent the muscles and bones that are critical for motion, the electrical circuits can be thought of as the nervous and circulatory systems which convey information, while the computer acts like a brain controlling these bodily functions. The following diagram shows how communication takes places within the system and by what means:

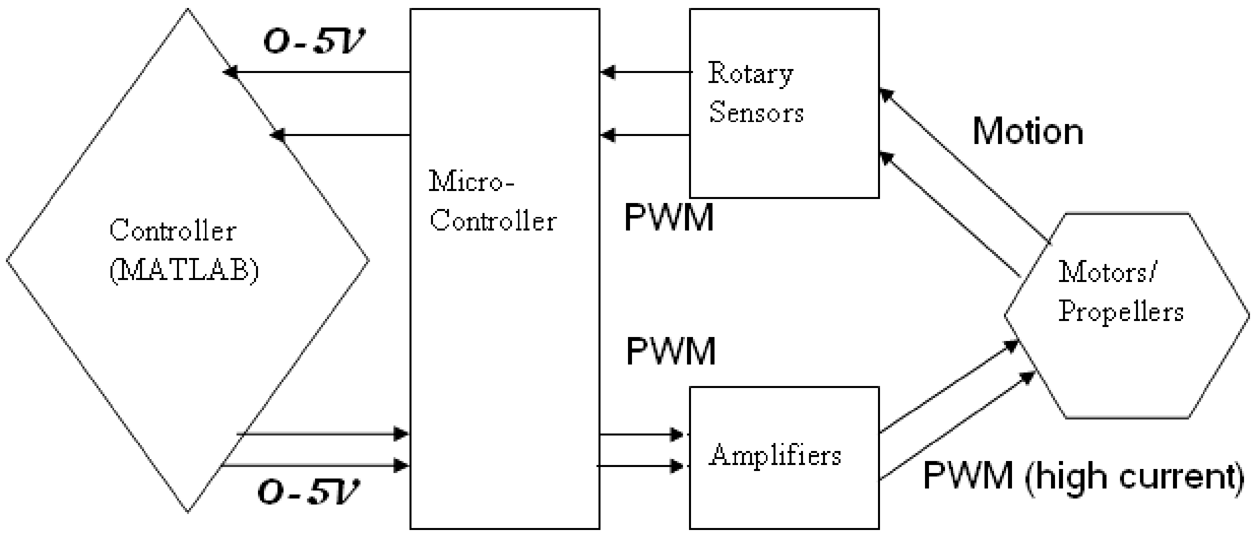


Figure 11. System flow chart in detail.

Mechanical

The skeletal and muscular structure of this system can summarized as the following components:

1. base platform
2. threaded steel pipe and connector
3. machined aluminum holder for *theta* direction encoder
4. *theta* direction encoder
5. machined aluminum ‘tuning fork’ to couple *theta* direction encoder shaft with arm
6. balanced rotating arm
7. motors and propellers
8. *phi* direction encoder

The physical relationships of these parts are illustrated on the next page.

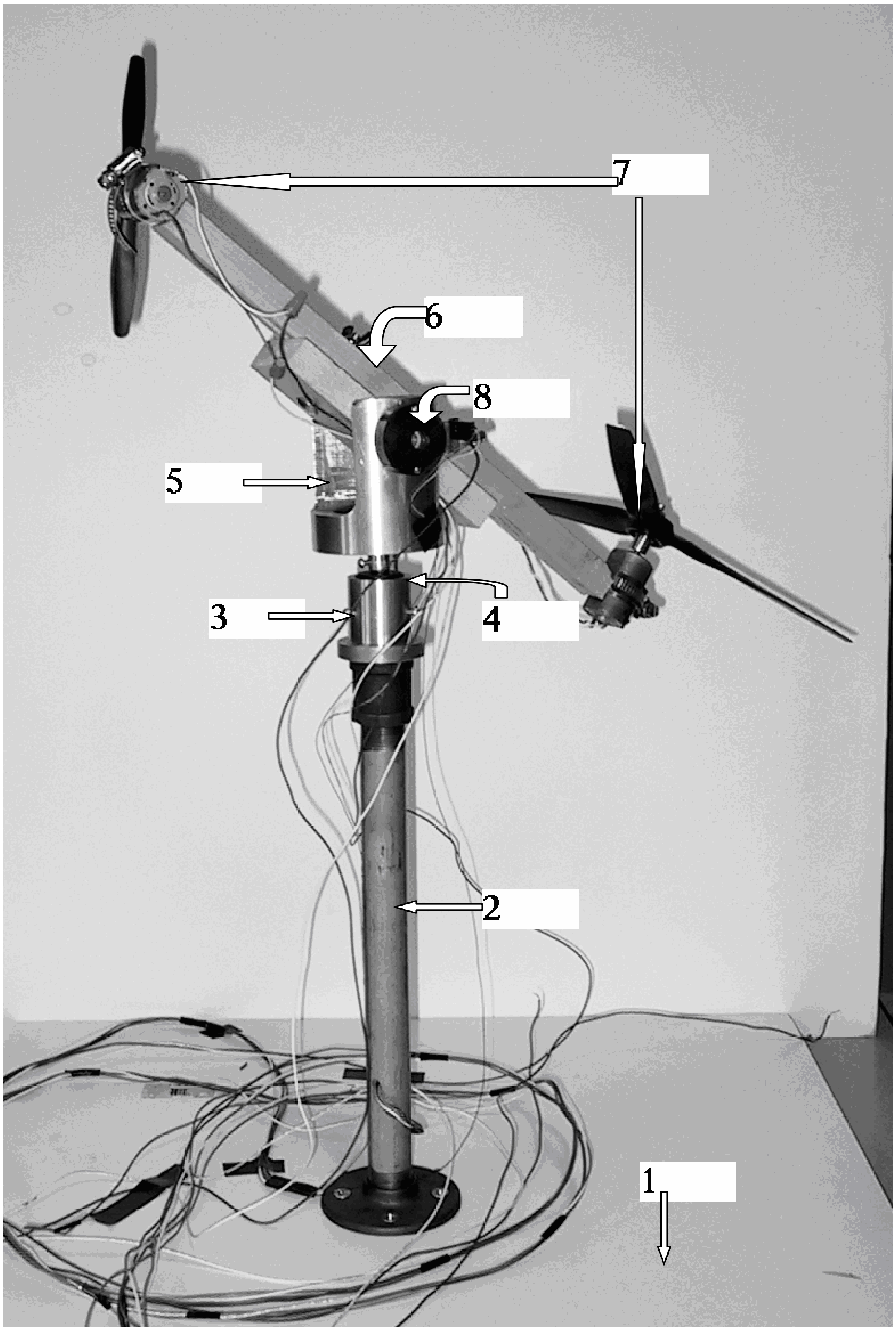


Figure 12. Mechanical Apparatus

Electrical

The electrical components of this system are shown in the figures below.

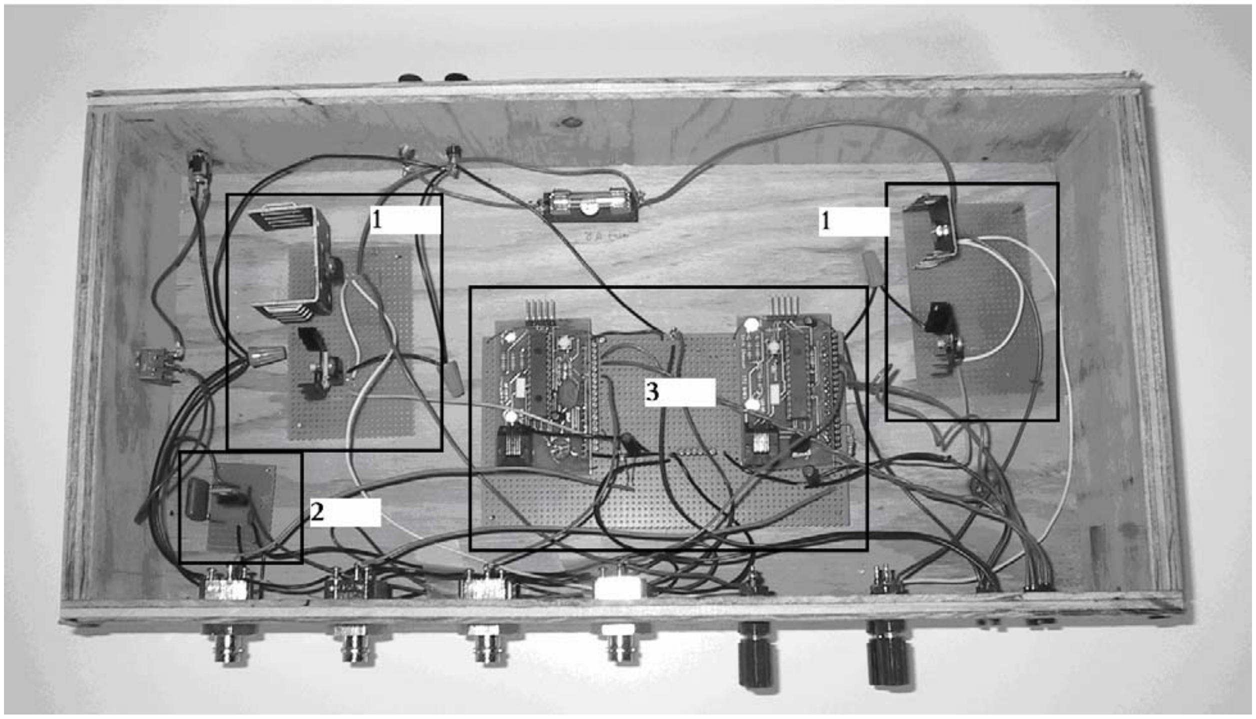


Figure 13. Physical realization of electrical components: 1) Amplifiers, 2) Voltage regulator for power supply to microcontrollers, 3) Microcontrollers; p*hi* microcontroller on left, *theta* microcontroller on right.

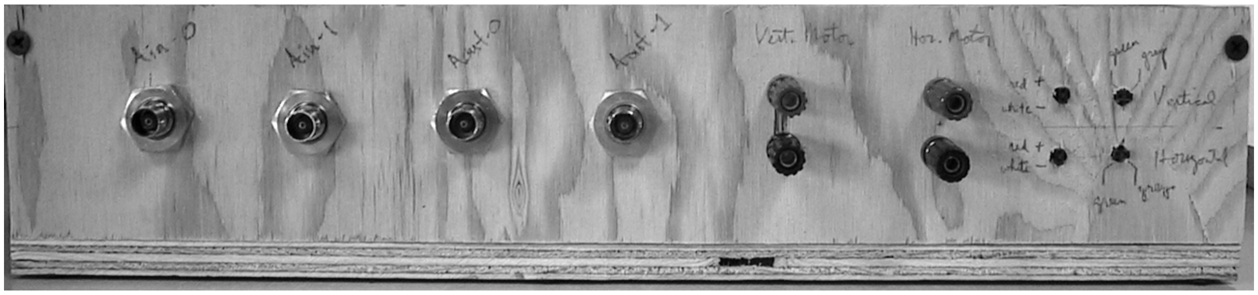


Figure 14. Faceplate to electrical components. From left to right: analog input BNC terminals (2), analog output BNC terminals (2), *phi* and *theta* motor terminals (2 sets), *phi* and *theta* encoder terminals (4 each).

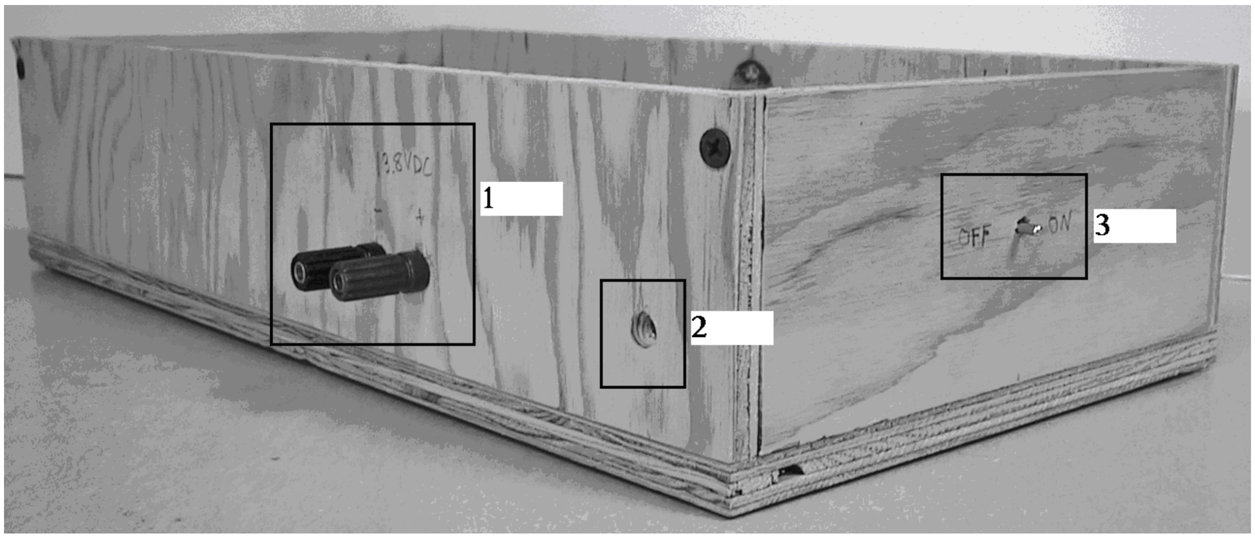


Figure 15. Additional electrical connectors: 1) high power supply connectors (13.8VDC, 10A), 2) low power supply jack (6VDC), 3) on/off switch for microcontrollers.

Computer

The following diagram outlines the PID controllers that I implemented in my system:

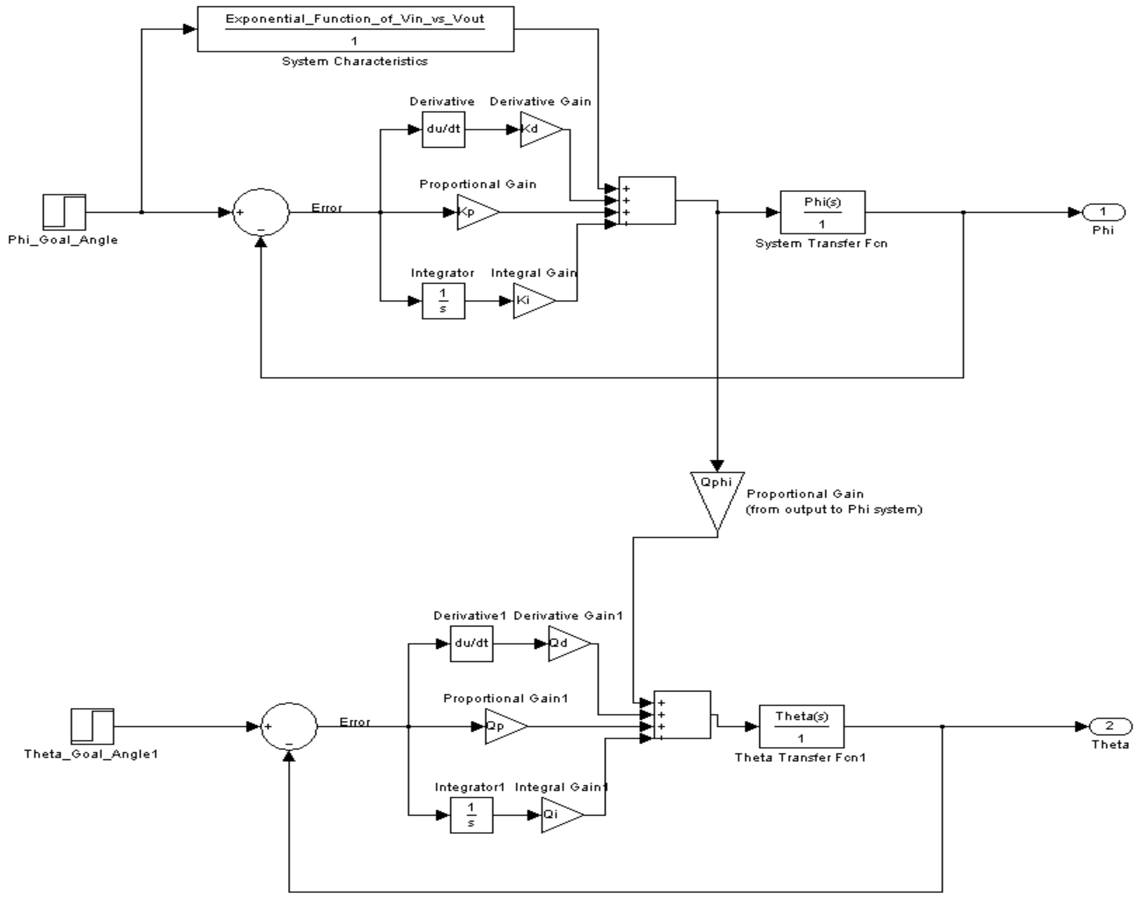


Figure 16. Basic PID controller flow chart.

Note that **Figure 16** does not detail the augmentations that were applied to the *Phi* direction controller. For a detailed discussion of controller implementation and the motivations behind design choices, see **System Development**: **Software**, *MATLAB Controller* for details.

System Development

Hardware

I encountered a number of hardware problems during the course of this project. To simplify the explanation of the design revisions that took place, I will discuss the difficulties and subsequent changes to the system that I made in chronological order.

MATLAB Data Acquisition

A great deal of time was spent trying to get the data acquisition code to work properly beyond the 2000 point buffer. Eventually this was easily overcome by leaving all settings alone but adding the line '`Ain.TriggerRepeat=inf.`' However, it was rare that I would ever need to take more than 2000 data points as most test runs were short and designed to diagnose or improve the controller.

Insufficient Vertical Forces Generated by Lift-propeller

Early on in the building process, my only power source available was only capable of outputting 6V/5A max. This was quite a constraint on the amount of lift I could generate, so I modified the system by adding weights to the shorter end of the arm. This reduced the amount of power needed to lift the arm, though the weights were eventually removed when my new power supply arrived.

Power Supply

The new power supply with its greater power output capabilities (max 13.8V/10A continuous) immediately solved the torque issues I was having, but it also created a few serious problems. My MOSFETs and diodes had been perfectly functional until I installed the new power supply, but after a couple trial runs, the MOSFETs went up in smoke (literally) and my diodes melted.

MOSFETs

I had been using some lower-power logic-level MOSFETs before the new power supply arrived. Logic-level MOSFETs were required because the other MOSFETs available in the lab were not turning on and off quickly enough, given the 20 kHz cycle which my PWM was based upon. However, the greater loads induced by the new power supply was more than my old MOSFETs could handle. In addition, the variety that I had obtained were surface mount chips, thus precluding the addition of heat sinks which might have prolonged their active lives.

The new MOSFETs I chose were also logic-level, with a low Gate to Threshold Voltage level (1-2V typical). This simply means that the transistor will turn on more readily even given a lower or slower PWM signal. In addition, a max drain current of 10A (per channel) was more than sufficient to ensure that they would function properly even under extreme conditions in my system. Lastly, heat sinks were attached to these chips to further reduce the danger of damage.

Diodes

Even more surprising to me than the failure of the MOSFETs was the melting of several Schottky diodes I had been using. Schottky diodes are well-known for their low forward voltage drops (~0.3V) which means they are wasting less energy and generating less heat. The fact that they were being destroyed suggests to me that quite a bit of current flows through them during the off-cycle of the MOSFETs. The replacements I ordered were heat-sinkable dual-channel 10A Schottky diodes. Even with (comparatively) large heat sinks attached, these diodes still get quite warm, but they have not failed even with the highest of currents.

Encoders

Perhaps with the increased current flow to and from the motors, a greater EMF was being generated in the vicinity of the encoders. It was not uncommon to observe a rise (or decrease) in the angular position voltage while the motors were on, even though there was no actual movement of the arm. However, this never occurred when the motors were off, so I attributed the problem to EMF disturbance. This problem was largely solved by twisting the pairs of cables carrying the encoder outputs. This physical arrangement of cables has the tendency to cause a reduction in sensitivity to nearby fields as they cancel their inductive effects in the twisted pair when the first cable loops around again. If values continue to drift, the angle values can be reset to zero by turning the microcontrollers off and the on again.

Time Variation of Motor Response due to Heat

This particular failing of the first design of my system was the most severe. For a long time, I attributed the day-to-day inconsistencies of my system to misbehavior of the electrical components, most notably the diodes. However, with the new diodes installed, I had no choice but to consider alternative sources of the system's variability. I eventually discovered that the motors were producing so much heat (while dissipating so little of it) that the blocks of wood surrounding them were getting hot to the touch. It seemed that the motors were producing so much heat that their electrical characteristics were being significantly altered (the electrical resistance of a conductor rises with temperature due to the physics of electron transport). The solution required a re-design of the arm mechanism, thus invalidating a significant portion of time spent perfecting the *phi* direction controller.

Swinging Arm and Support Structure

The time variability of the motors due to heat was one of two significant reasons that led to the redesigning of the arm. The second was the fact that the *phi* direction propeller was so far out from the arm's central axis of rotation that it was producing very little torque in the *theta* direction. This was problematic because the *theta* direction motor would only be unable to move the arm in one direction. Thus the redesigned arm served two primary purposes: to allow the motor to 'breathe' more and dissipate heat faster, and to bring the *phi* direction fan closer in to the center thus creating a restoring torque. All of these goals were met by the new design.

Software

The two pieces of software that this project relies upon are the C code that runs on the PIC Microcontrollers and the MATLAB script which is responsible for implementing the controller algorithm. The PIC-C code worked without much difficulty, though some trial and error was involved in maximizing the resolution for each encoder.

MATLAB Controller

Although successfully coded and applied, the PID controllers I implemented never attained full success in both *theta* and *phi* directions simultaneously. However, there were a number of ways that I worked to improve the reliability of the PID controller. The two most significant modes of modification to the classical PID controller were 1) adding in an asymmetry to the controller to minimize any overshoot, and 2) modifying the weights of the contributions of the controller components based on various functions of the angle.

The asymmetry in my PID controller directly addresses one mode of (open-loop) instability in the system. The arm is balanced about a central pivot; this can be idealized as a single rod free to rotate in a vertical plane, fixed at one end. The limit of the amount of torque required to lift this rod an incremental angle eventually goes to zero as the rod approaches a perpendicular posture. Though my system's swinging arm can never reach a vertical position (at most close to 40 degrees above horizontal), the aforementioned relation still remains true. Thus any overshoot past the desired angle can be dangerous as it will quickly lead to an almost irrecoverable state where the *phi* direction fan is barely on, but the arm is stuck at its maximum vertical angle and does not fall back down (or does so very quickly).

The asymmetrical solutions that I used in my code apply primarily to an increase in the rate of integration and the proportionality constants once the system overshoots past the goal angle. This faster integration and stronger proportionality constant cause a fairly quick reaction that usually counteracts the natural upwards tendency of the system. Balancing these constants with the rest of the system so that the motor does not lose too much speed if it overshoots is crucial.

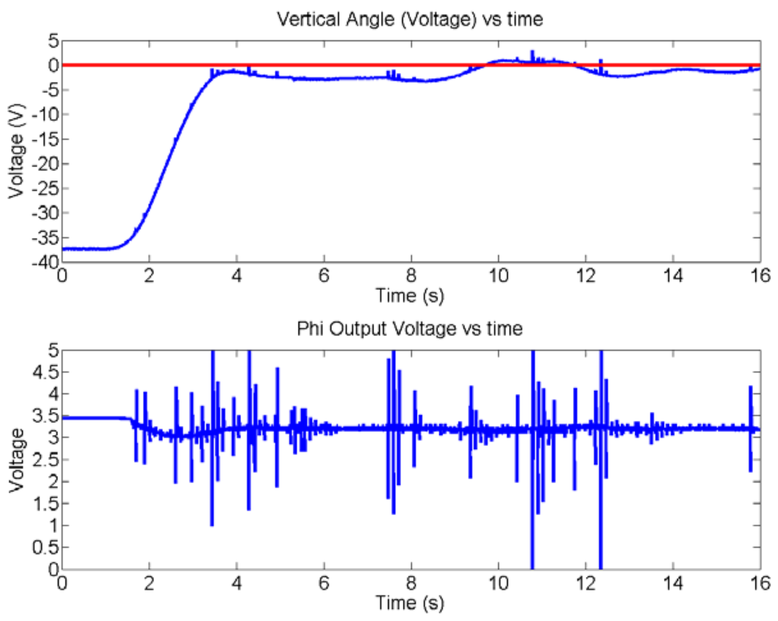
The second mode of adapting a classical PID controller to my system involved weighting controller components by sine and cosine relations based upon the angle to the goal. For example, proportionality constants are applied both to the sine of the angle to the goal (thus more quickly decreasing this controller component when the system approaches its goal) and to the cosine of the absolute angle measured against horizontal (thus retaining some level of a 'normal' proportional controller). Perhaps most significant is the weighting of the integral component of the controller by the square (or higher) of the cosine of the angle to the goal. This restricts the sensitivity of the (already very small) integral component of the controller to the region near the target angle. This serves the purpose of retaining the integral controller's ability to drive the error to zero without first slowing the system way down before it gets near its goal, while avoiding potential over- and under-shoot problems.

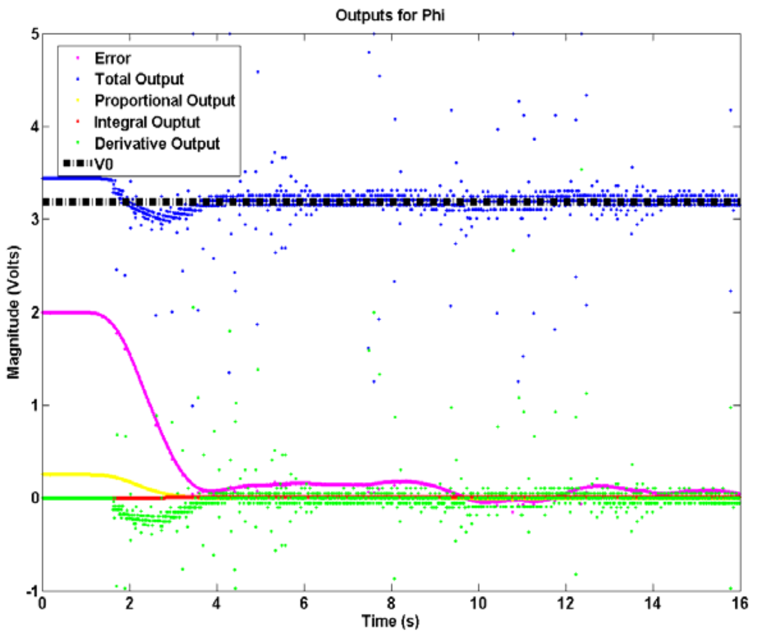
In general, I tweaked the PID constants first by aiming for some amount of overshoot and then slowly increasing the derivative constant in order to increase system speed and reduce dangerous overshoot. My iterative cycle seemed consistent with most published methods of tuning PID controllers.

Results

Before Redesigning the Arm

Before redesigning of the arm, I attained a fairly high degree of success in controlling the *phi* direction position for a variety of goal angles. The major difficulty at this point was related to the time variability of the motors. However, if I settled into a fairly steady rhythm of trails and system cooling, the motors would respond fairly consistently and I would get good results. The most readily attainable angles were those very near to horizontal, and slightly below. **Figures 17.1** and **17.2**, and **18** illustrate two such favorable tests.





Figures 17.1 and **17.2**. *Phi* goal angle of 0 degrees (horizontal) trials. **17.1** shows the rise in position and the total output of the controller. **17.2** shows the various contributions to the total output of the controller from proportional, derivative and integral components.

Of particular note here is the slight system overshoot at 10s, which is quickly corrected by the integral controller (which can be very slightly discerned as the red line dipping below 0 in **figure 17.2**).

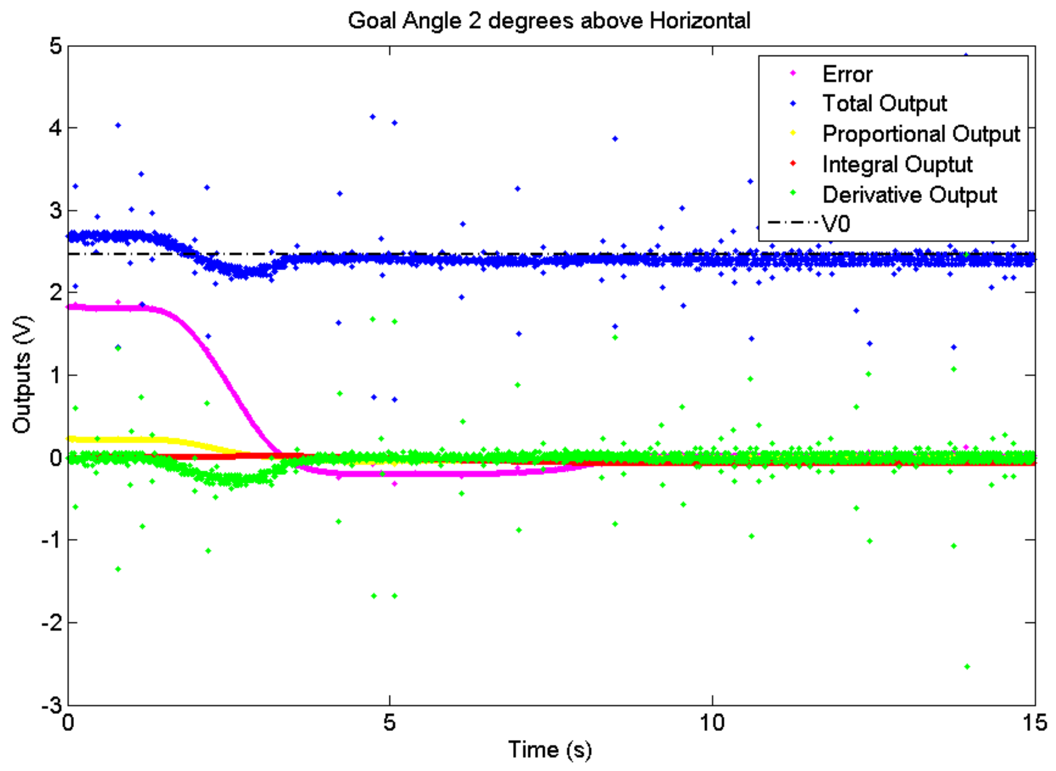
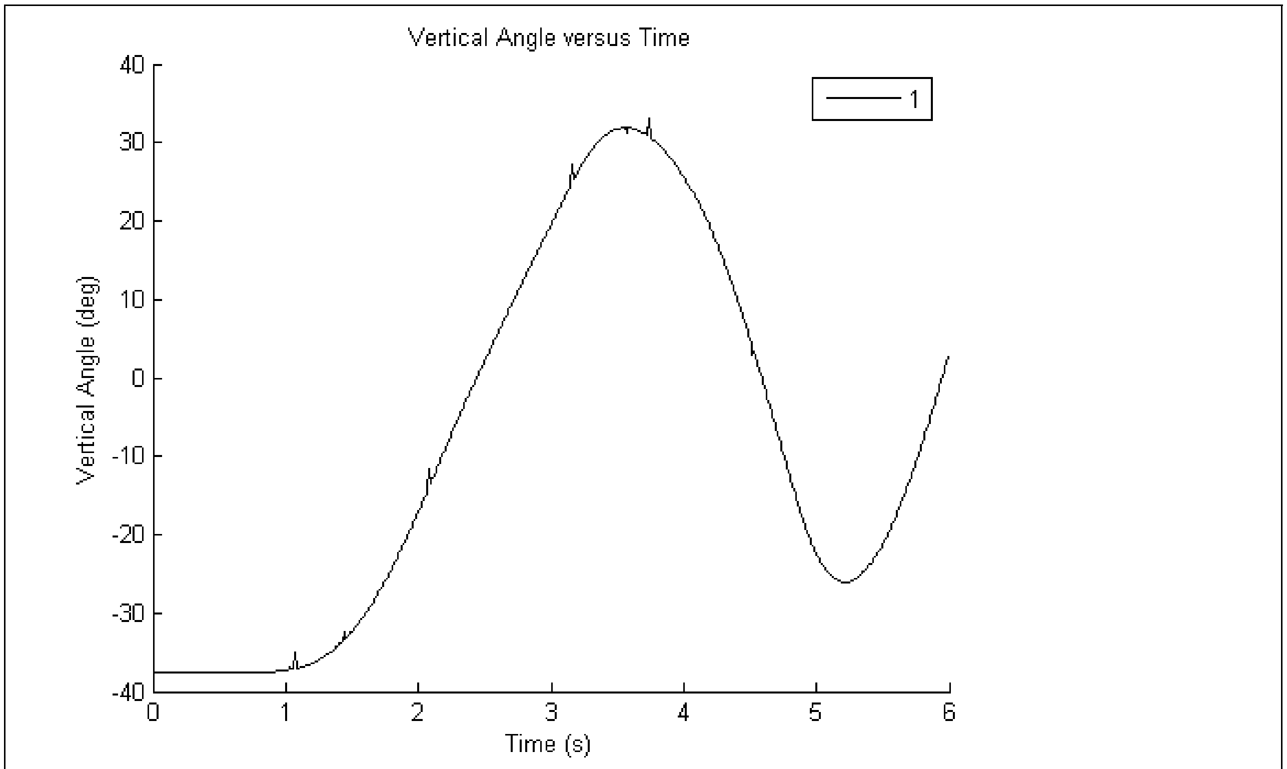


Figure 18. Output for goal angle of 2 degrees above horizontal.

While the controller tended to work fairly well for a variety of angles in those ranges, the fact that it would be impossible to design a functional *theta* direction controller coupled with the time variability of the motors necessitated that I redesign the arm. The severity of the inconsistency of the motors is illustrated by **Figures 19.1** and **19.2**.



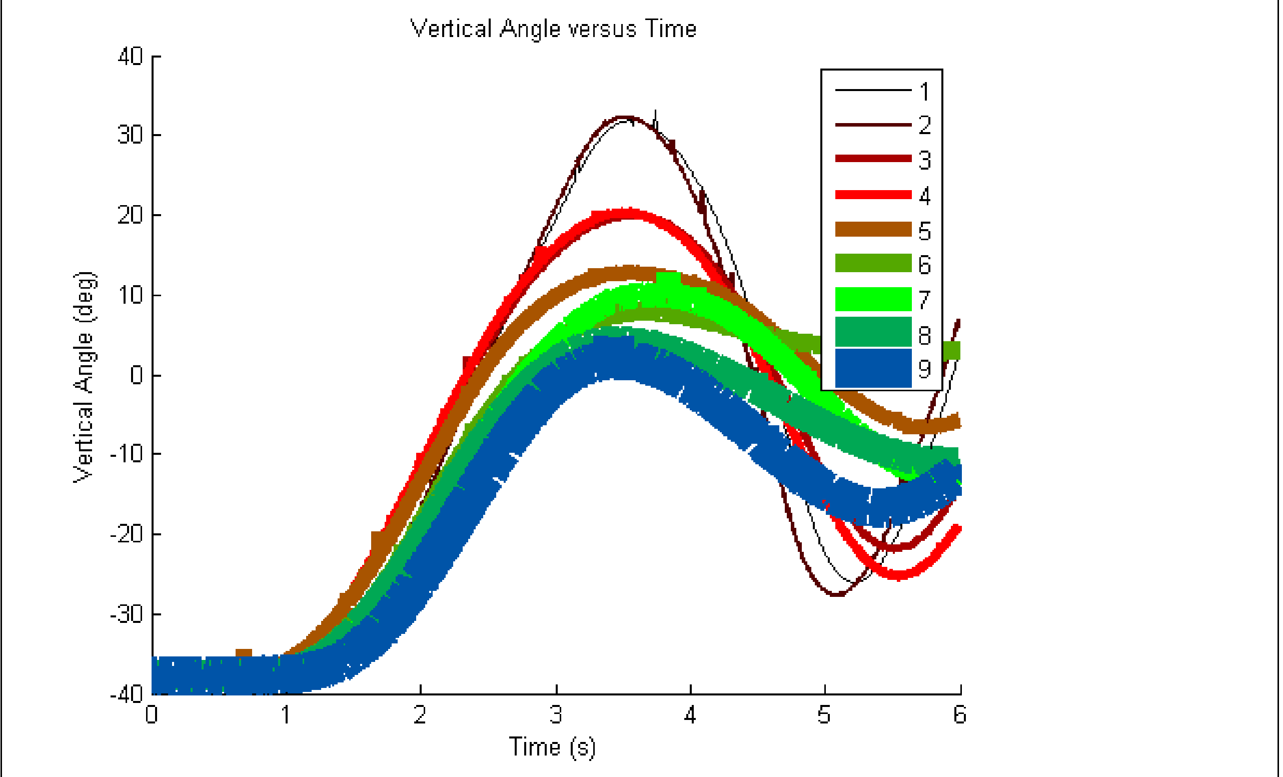


Figure 19.1 and 19.2. The first graphic illustrates the response of a system whose goal angle is nearly horizontal. The second figure shows with increasingly thick lines the progression of the system response with time (no changes to controller or other hardware were made during this time). By the ninth trial, the response peaks at an angle nearly 30 degrees less than the first run.

After Redesigning the Arm

Though redesigning the arm solved the dilemmas of creating a restoring torque in the *theta* direction and increasing the rate of heat dissipation, it also made the system more unstable. By moving motors closer to the center, the overall moment of inertia went down significantly (the moment of inertia for a bar rotating through its midpoint is given by $I = 1/12*ML^2$). This means that the system became more prone to high angular accelerations and greater angular velocities. In a system that has an inherent time delay, this modification works to make a stablizing controller more difficult to implement.

Nonetheless, a Simulink model of the system was generated and a potentially stable controller implemented. The model was generated by doing a very basic fit of a step response to a first order system with a time delay.

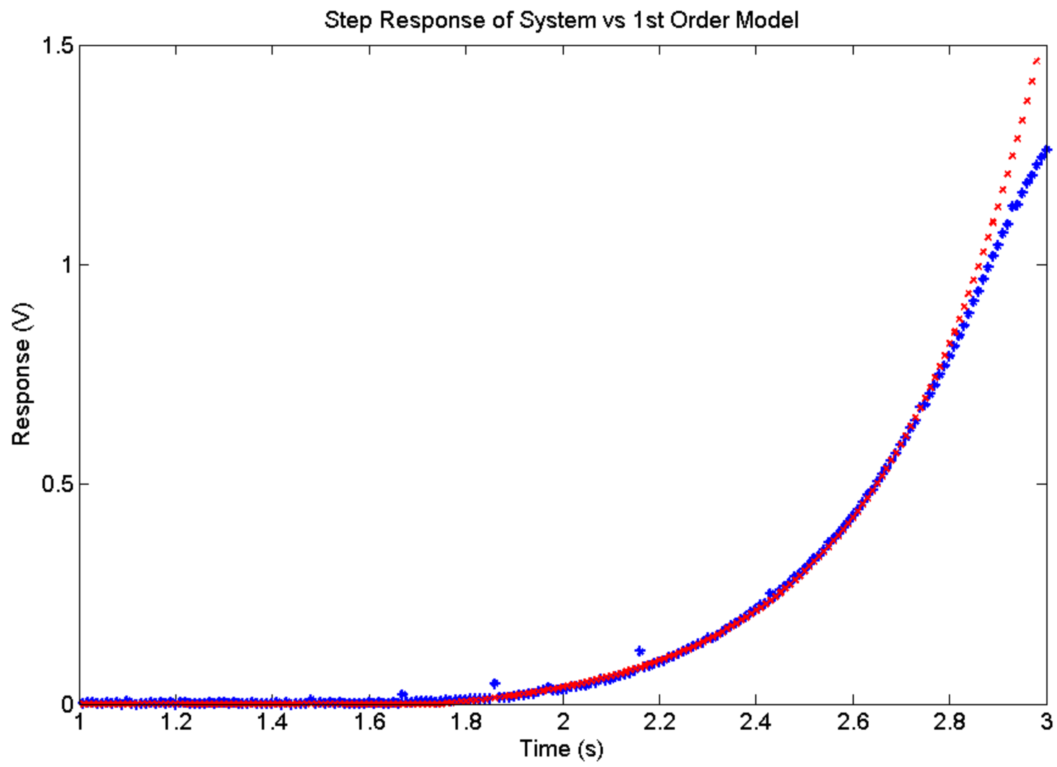


Figure 20. Here a step response of a first-order model with a time delay (red x's) was fit to the system step response in the *phi* direction (blue *s).

While it is evident that my system is of a higher order, and the two responses diverge fairly significantly after 3 seconds, it useful nonetheless to examine a simplified theoretical model with a time delay to consider potential controllers.

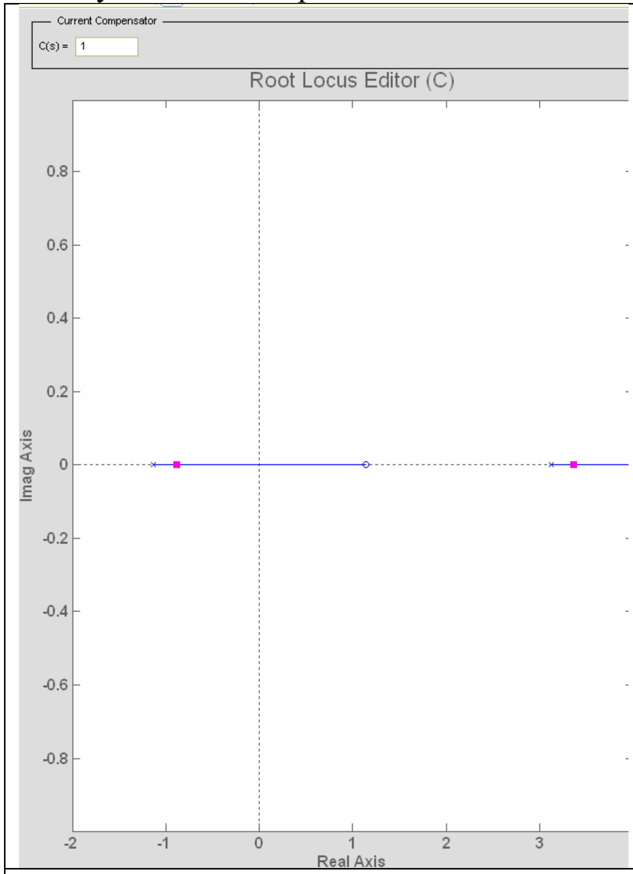
In order to model the time-delayed system in Simulink, it is necessary to perform a Pade approximation.

```
>> pade(sys, 1) %perform a first-order Pade approximation.

Transfer function:
  -0.5 s + 0.5714
---------------------
s^2 - 1.982 s - 3.571
```

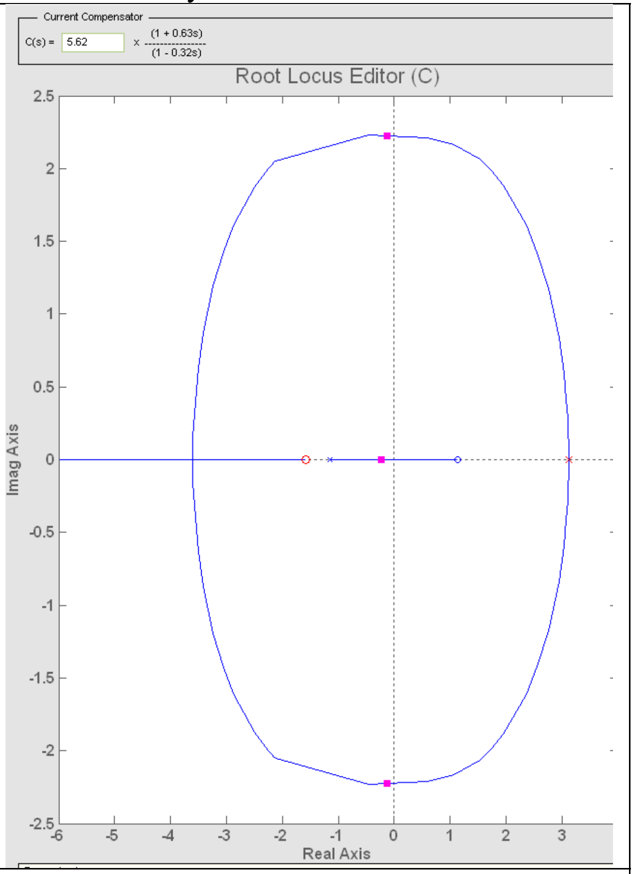
Figure 21. First order Pade approximation of theoretical system with time delay.

This system is then input into Simulink and analyzed for stability.



Current Compensator

C(s) = 1



Current Compensator

C(s) = 5.62 x (1 + 0.63s) / (1 - 0.32s)

Figure 22.1 and **22.2**. Before controller is applied, and after application of a lead controller. Note that the system is now stable.

The lead controller implemented above takes the form:

$$5.62 * \frac{1 + .63s}{1 - .32s}$$

The new system response is shown below:

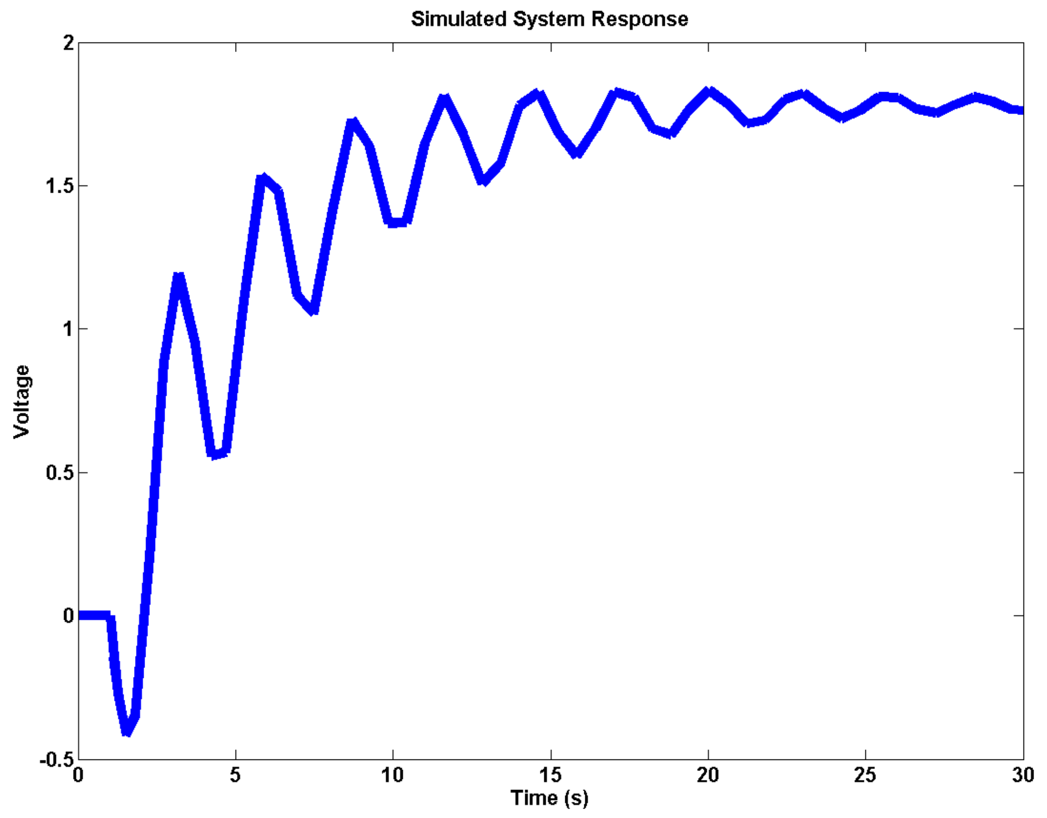


Figure 23. Simulated system response with a lead controller. Though oscillatory and relatively slow, this system does eventually stabilize.

In summary, though the controller is not 100% operational, the system is fully constructed and the controller code is in place. If the next person decides to try their hand at a modified PID controller such as mine, they simply have to load the code that has already been written and tweak the values.

Discussion

Perhaps the greatest impediment to implementing a successful controller in this system was the significant time delay inherent to the physical apparatus. The motors took a significant amount of time to spin up to speed, and then there was a slight delay before the rotors would catch the air to produce lift. I feel there are several possible directions to go from here: 1) accept the time lag as is and work on a complex controller (or accept a slow response) possibly by first developing a rigorous system model or 2) work on a mechanical solution to decrease the time lag.

If I were to redesign the system with the goal of reducing the time lag in mind, I would certainly try new motors or propellers. I don't believe these motors were designed to produce the levels of torque that they are generating at high currents. Part of the reason for the slow spin-up time is that these motors simply are not very powerful. Another alternative would be to try smaller propellers. This would lighten the load on the motors and increase system responsiveness, though the apparatus would have to be more finely balanced due to the decrease in the amount of lift generated (thus making it even more unstable!).

The primary goal of my first PID controller was to get as close as possible to the target based upon the prior knowledge of the system's transfer function (based upon the exponential fit of step inputs versus steady state solutions) and then to move as slowly as possible to make any corrections necessary. The slow speed of correction was required by the instabilities of the system and the time lag.

Designing a finely crafted controller designed to compensate for the time-lag would be quite a feat. Even a basic physical model of the system may shed light on potential solutions. The method of PID control I implemented for the first arm simply does not work very well for the redesigned system. This is largely due to the fact that the moment of inertia is so significantly decreased; the range of voltages that will keep the system at a more or less stable angle has become frighteningly thin. As a result, it is extremely difficult to produce a transfer function of the system in the *phi* direction using step inputs.

While the first-order model with a time delay fails to match the physical system's response after a period of time, the Simulink lead-controller does give us hope that a controller can be designed to combat large time delays and system instabilities. I would strongly recommend against using this system as a general lab experiment for control theory students, due to its current unstable condition. I do however have great faith in the technical abilities of Swarthmore Engineering students; my apparatus may be more appropriate for a final project or E90 that would involve research and design of experimental or exotic control techniques.

Future Work

In its current state, the system is prone to damage from the repetitive 'falls.' This is a serious condition as it is possible that the arm will in time incur permanent damage to the *theta* direction encoder shaft. This could be fairly easily remedied by constructing a downwards pointing conic support attached at the first machined aluminum piece which would 'catch' the arm.

Another important safety concern is that of the propellers. Before being used in a laboratory setting, I would recommend constructing fan cages around the plastic blades as to prevent curious fingers from getting cut.

Beyond these basic changes, there is only the simple matter of implementing a controller. I look forward to hearing about the various methods that are implemented on my apparatus when they meet with success!

Appendices

Appendix A: Datasheets

Rotary Optical Encoders

Modular Incremental Rotary Optical Encoder

MX15 Series

By the time you have read this first sentence, you could have installed BEI's model MX15 INSTA-MOUNT™ modular optical encoder. In addition to its quick and easy installation, the MX15 is designed to operate with jitter-free output signals without tight controls on shaft endplay, runout or perpendicularity. The new INSTA-MOUNT™ encoder is capable of operating within a temperature range of -10° to +70°C, requiring less than 30 milliamps of L.E.D. current, without degradation of output signals and is short circuit protected. The MX15 is perfectly suited for motor manufacturers and other high volume OEMs.

BEI's INSTA-MOUNT™ Series encoder offers 5V TTL compatible quadrature outputs with index and complements as options. Axial shaft movements during operation, of ±0.010", will not adversely affect the output signals. Shaft runouts of 0.005" TIR can also be absorbed by this device without affecting output signal performance.

Standard Features

- Resolutions to 1024 PPR
- Quick and easy installation
- Tolerant of axial shaft movement often associated with less expensive motors
- Jitter-free outputs
- Index options
- Increased MTBF (lower component count)
- 26LS31 line driver output from MX156
- High Frequency response
- 2-year warranty

Figure 1

(MX152/MX153)

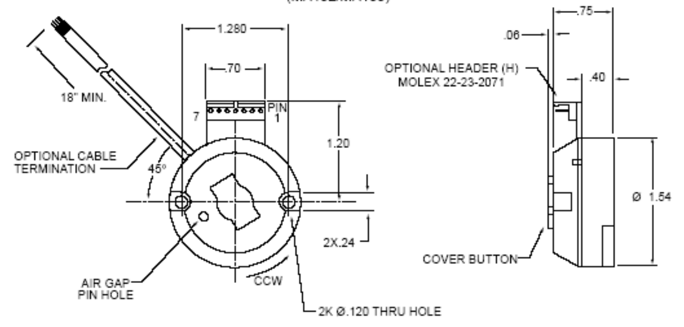
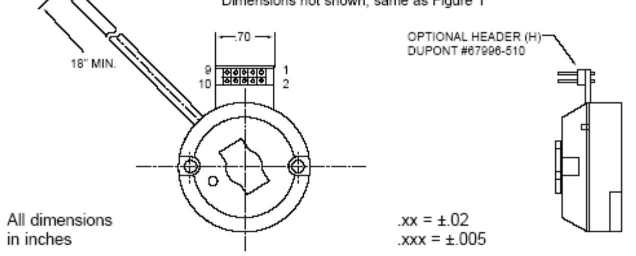


Figure 2

(MX156)

Dimensions not shown, same as Figure 1



BEI DUNCAN ELECTRONICS DIVISION
BEI TECHNOLOGIES, INC.

15771 Red Hill Avenue
Tustin, CA 92780-7303
(714) 258-7500 Fax: (714) 258-4629
email: sales@beiduncan.com
www.beiduncan.com

ISO9001 Certified/QS9000 Compliant

MX15 Series
Modular Incremental Rotary Optical Encoder

Performance Specifications

Mechanical

Dimensions	see Figure 1
Weight	2.0 oz.
Moment of Inertia	2.6 x 10^{-5}oz in sec^2
Bore Size	see "Ordering Information"

Motor Interface

Mount Holes	#2-56 threads @ 180° on 1.280 dia. B.C.
Mount Hardware	#2-56 x 3/4 in. long (provided)
Perpendicularity Shaft to Mount	±0.002" TIR
Shaft Runout	0.005" max (each 0.0001 degrades accuracy by 0.5 arc minutes)
Shaft Endplay Dynamic or Static	±0.005"
Shaft Finish	16 microinches or better End must be chamfered or rounded
Shaft Tolerance	0.0002"/-0.0007" (e.g. Ø.2493/.2498)
Shaft Length	0.45" minimum (remove cover button for motor through-shafts)

Electrical

Code	incremental
Pulses per Revolution	see "Ordering Information"
Index Pulse Options (no index on MX152)	ungated index (U) gated index (G)
Supply Voltage	5 volts ±5% @ 80mA max.
Output Format (MX152 & MX153)	dual channel quadrature and index (no index on MX152)
Output Format (MX156)	dual channel quadrature and index with complements
Output Type (MX152 & MX153)	square wave TTL. 16mA sink 500µA source. Short circuit protected
Output Type (MX156)	TTL differential line driver (26LS31 or equiv.) should be terminated into a line receiver (26LS32, or equivalent circuit)
Frequency Response	see graph: Fig. 3
Rise Time	1.0µsec. max.

Environmental

Temperature	operating: -10°C to +70°C storage: -40°C to +125°C
Enclosure	unsealed housing unit must be protected from harsh environments

Termination

Terminal Board (Header)
(MX152 & MX153)

Pinout

Pin #	Signal	Pin #	Signal
1	N/C	5	data B
2	index (MX153)	6	data A
3	N/C	7	ground
4	+5 volt		

(MX156)

Pinout

Pin #	Signal	Pin #	Signal
1	+5 volt	6	data $\overline{A}$
2	+5 volt	7	ground
3	$\overline{\text{index}}$	8	ground
4	index	9	data B
5	data A	10	data $\overline{B}$

Round Shielded Cable
(MX152 & MX153)

Color Code

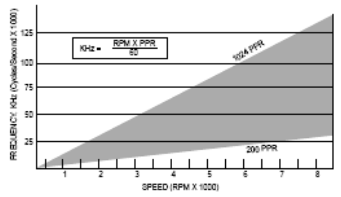
Color	Function	Color	Function
Red	+5 volt	Green	data B
Black	ground	Orange	index (MX153)
White	data A		

(MX156)

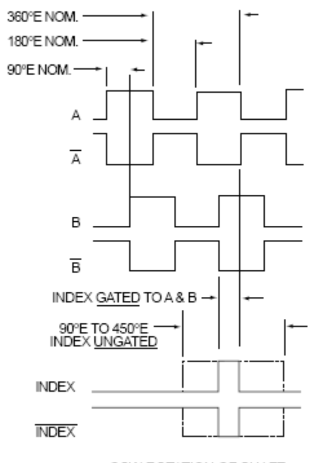
Color Code

Color	Function	Color	Function
Red	+5 volt	Green	data B
Black	ground	Wht/Blk	date $\overline{B}$
White	data A	Orange	index
Blue	data $\overline{A}$	Red/Blk	$\overline{\text{index}}$

Figure 3



Output Wave Form



CCW ROTATION OF SHAFT

■ MX152 OUTPUTS A & B ONLY
■ MX153 OUTPUTS A, B & INDEX ONLY
■ MX156 OUTPUTS AS SHOWN

Ordering Information

MX15 X - XX - XXXX - X - X

Basic Model No.

Output Format
- 2 = Quadrature
- 3 = Quadrature w/index
- 6 = Quadrature w/index & complements

Bore Size
- 25-.25", 38-.375"
- 6M-6mm, 8M-8mm

Pulses Per Revolution (PPR)
- 500, 512, 1000, 1024

Index Option
- G = gated to data A & B
- U = ungated

Electrical Termination
- T = terminal board
- H = terminal board w/header
- P = round shielded cable

EXAMPLE: MX153-25-500-U-P

BEI DUNCAN ELECTRONICS DIVISION
BEI TECHNOLOGIES, INC.

15771 Red Hill Avenue
Tustin, CA 92780-7303
(714) 258-7500 Fax: (714) 258-4629
email: sales@beiduncan.com
www.beiduncan.com

Miniature Incremental Rotary Optical Encoder

EX11 Series

The EX11 Series was developed to provide a high precision, low cost enclosed shaft encoder for light duty applications. The EX11 offers benefits of the Opto-ASIC design with 1024 line counts in a 1.1 inch diameter size.

Packaged in a glass filled polycarbonate housing with a 1/8" stainless steel shaft and precision bearings, the EX11 provides superior performance at a lower cost.

The EX11 Series is capable of operating over a temperature range of -10°C to +70°C without degradation of signals.

Proven design and Duncan Electronics' experience makes the EX11 perfectly suited for high volume OEM applications, including: robotics, process control and instrumentation.

Standard Features

- Line counts up to 1024 PPR
- RS422 compatible 26LS31 line driver
- 1.1 inch diameter package in servo or flange mount configurations
- 2-year warranty

Figure 1

Servo Mount

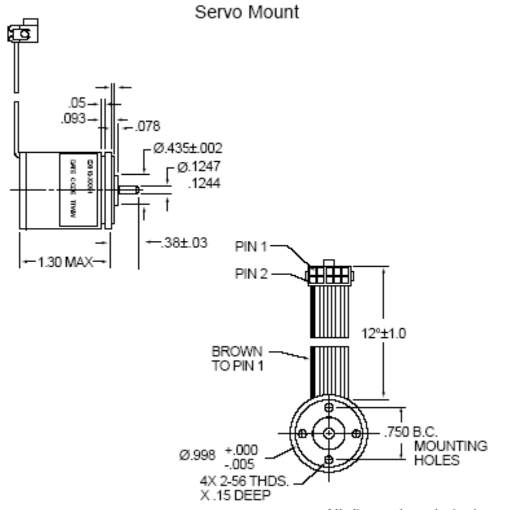
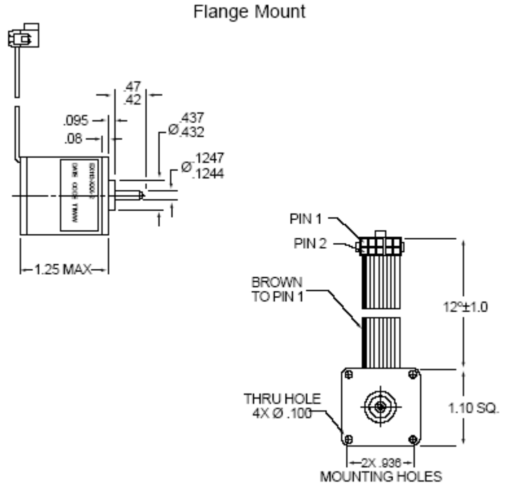


Figure 2

Flange Mount



All dimensions in inches

.xx = ±.02
.xxx = ±.005

BEI DUNCAN ELECTRONICS DIVISION
BEI TECHNOLOGIES, INC.
15771 Red Hill Avenue
Tustin, CA 92780-7303
(714) 258-7500 Fax: (714) 258-4629
email: sales@beiduncan.com
www.beiduncan.com

ISO9001 Certified/QS9000 Compliant

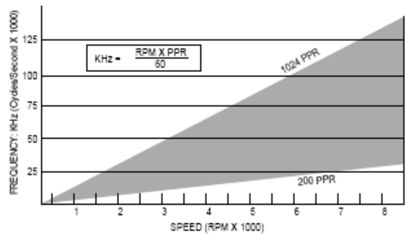
EX11 Series
Miniature Incremental Rotary Optical Encoder

Performance Specifications

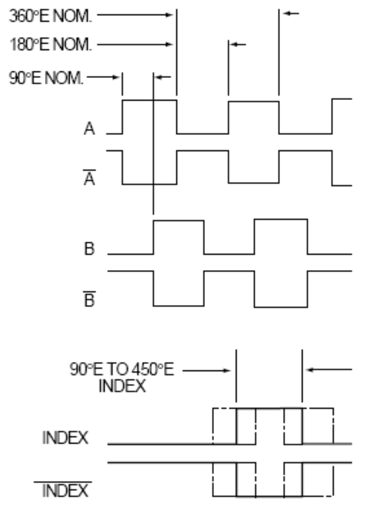
Mechanical	
Dimensions	see Figure 1
Weight	2.0 oz. (Approx.)
Shaft Diameter	0.1247 +.0000/-.0003
Shaft Load	axial 2 lbs., Radial 1 lb.
Torque, starting	less than 0.4 oz. in.
running	less than 0.2 oz. in.
Inertia	3.0 x 10^{-5} oz. in./sec.
Motor Interface	
Servo Mounting Holes	4 places #2-56 @ 90° on 0.75" B.C.
Servo Mount	designed to accommodate motor mount cleat "PIC type" L2-2
Flange Mounting Holes	4 places .100 dia. thru holes
Shaft Coupling	must be flexible (do not hard mount)
Electrical	
Code	incremental
Pulses per Revolution	see "Ordering Information"
Supply Voltage	+5 volts ±5% @ 80mA max.
Output Format	dual channel quadrature and index with complements (no index on EX112)
Output Type	TTL differential line driver (26LS31 or equiv.) should be terminated into a line receiver (26LS32, or equivalent circuit)
Rise Time	1.0µsec. max.
Frequency Response	see graph: Fig. 3
Environmental	
Temperature	operating: -10°C to +70°C storage: -40°C to +125°C
Termination	
Type	28 AWG flat ribbon cable with 10 position connector Berg P/N 65863-165 or equiv. Mates with Berg P/N65863-165 or equiv. (mating connector not provided)

Pin No.	Color	Signal
1	Brown	N/C
2	Red	+5V
3	Orange	$\overline{B}$
4	Yellow	B
5	Green	$\overline{\text{index}}$
6	Blue	index
7	Violet	$\overline{A}$
8	Gray	A
9	White	N/C
10	Black	Ground

Figure 3



Output Wave Form



CW ROTATION OF SHAFT

- ■ EX112 OUTPUTS A & B ONLY
- ■ EX113 OUTPUTS A, B & INDEX ONLY
- ■ EX116 OUTPUTS AS SHOWN

Ordering Information

EX11 X - XXXX - X

Basic Model No.

Output Format
- 2 = Quadrature
- 3 = Quadrature w/index
- 6 = Quadrature w/index & complements

Pulses Per Revolution (PPR)
- -200, -256, -500,
- -512, -1000, -1024

Mounting
- 1= Servo mount
- 2 = Flange Mount

EXAMPLE: EX113-500-2

BEI DUNCAN ELECTRONICS DIVISION
BEI TECHNOLOGIES, INC.
15771 Red Hill Avenue
Tustin, CA 92780-7303
(714) 258-7500 Fax: (714) 258-4629
email: sales@beiduncan.com
www.beiduncan.com

FAIRCHILD SEMICONDUCTOR®

RFD16N05L, RFD16N05LSM

Data Sheet **January 2002**

16A, 50V, 0.047 Ohm, Logic Level, N-Channel Power MOSFETs

These are N-Channel logic level power MOSFETs manufactured using the MegaFET process. This process, which uses feature sizes approaching those of LSI integrated circuits gives optimum utilization of silicon, resulting in outstanding performance. They were designed for use with logic level (5V) driving sources in applications such as programmable controllers, automotive switching, switching regulators, switching converters, motor relay drivers and emitter switches for bipolar transistors. This performance is accomplished through a special gate oxide design which provides full rated conductance at gate biases in the 3V to 5V range, thereby facilitating true on-off power control directly from logic circuit supply voltages.

Formerly developmental type TA09871.

Ordering Information

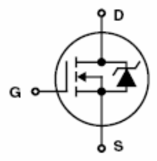
PART NUMBER	PACKAGE	BRAND
RFD16N05L	TO-251AA	RFD16N05L
RFD16N05LSM	TO-252AA	RFD16N05LSM

NOTE: When ordering, include the entire part number. Add the suffix 9A to obtain the TO-252AA variant in tape and reel, i.e. RFD16N05LSM9A

Features

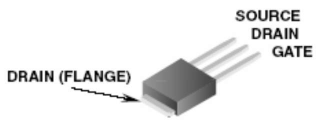
- 16A, 50V
- $r_{DS(ON)} = 0.047\Omega$
- UIS SOA Rating Curve (Single Pulse)
- Design Optimized for 5V Gate Drives
- Can be Driven Directly from CMOS, NMOS, TTL Circuits
- Compatible with Automotive Drive Requirements
- SOA is Power Dissipation Limited
- Nanosecond Switching Speeds
- Linear Transfer Characteristics
- High Input Impedance
- Majority Carrier Device
- Related Literature
  - TB334 "Guidelines for Soldering Surface Mount Components to PC Boards"

Symbol

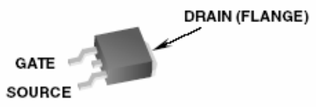


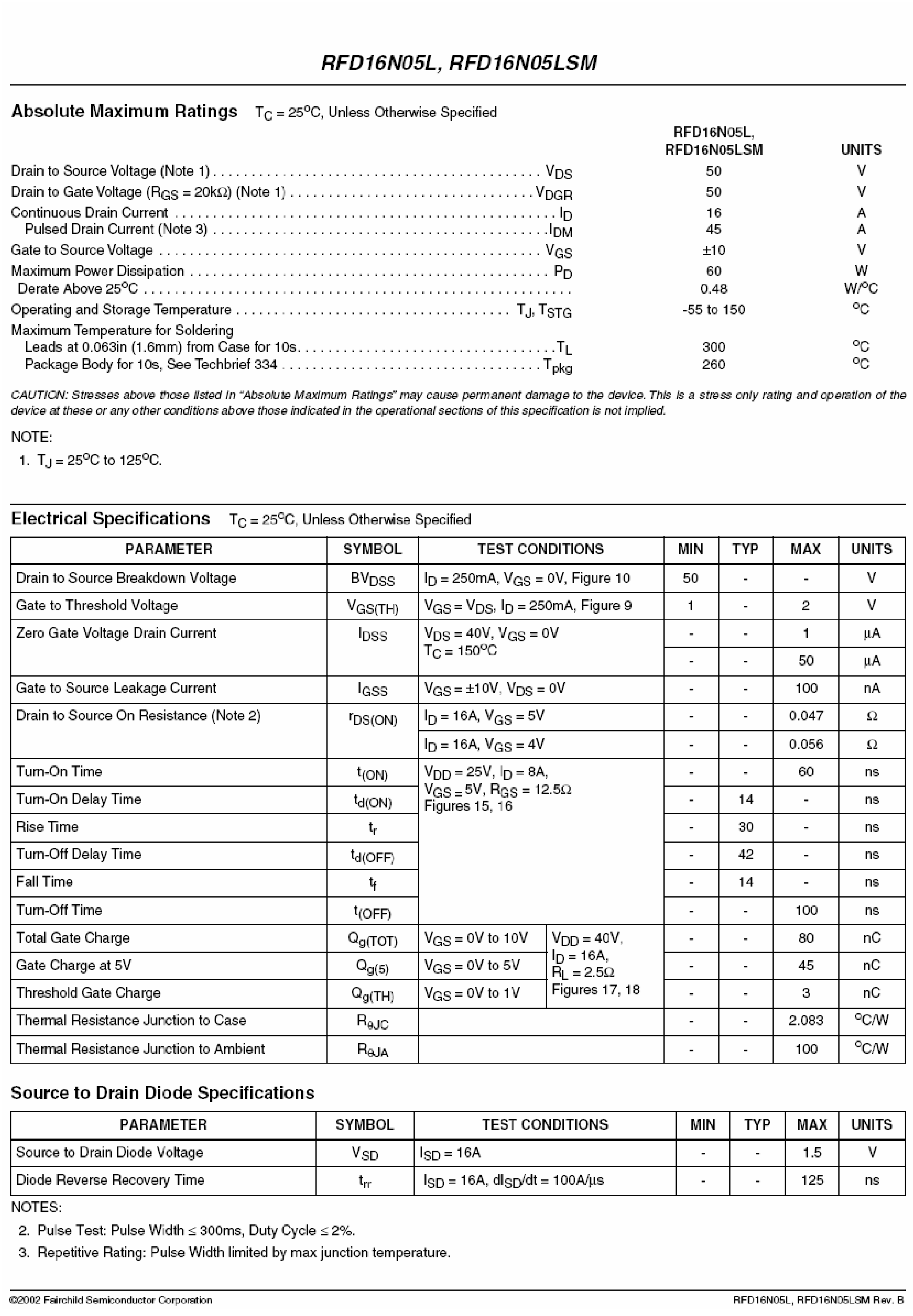
Packaging

JEDEC TO-251AA



JEDEC TO-252AA





Absolute Maximum Ratings $T_C = 25^oC$, Unless Otherwise Specified

		RFD16N05L, RFD16N05LSM	UNITS
Drain to Source Voltage (Note 1)	V_{DS}	50	V
Drain to Gate Voltage ($R_{GS} = 20k\Omega$) (Note 1)	V_{DGR}	50	V
Continuous Drain Current	I_D	16	A
Pulsed Drain Current (Note 3)	I_{DM}	45	A
Gate to Source Voltage	V_{GS}	±10	V
Maximum Power Dissipation	P_D	60	W
Derate Above 25°C		0.48	W/°C
Operating and Storage Temperature	T_J, T_{STG}	-55 to 150	°C
Maximum Temperature for Soldering			
Leads at 0.063in (1.6mm) from Case for 10s	T_L	300	°C
Package Body for 10s, See Techbrief 334	T_{pkg}	260	°C

CAUTION: Stresses above those listed in "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress only rating and operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

NOTE:

1. $T_J = 25^oC$ to 125^oC.

Electrical Specifications $T_C = 25^oC$, Unless Otherwise Specified

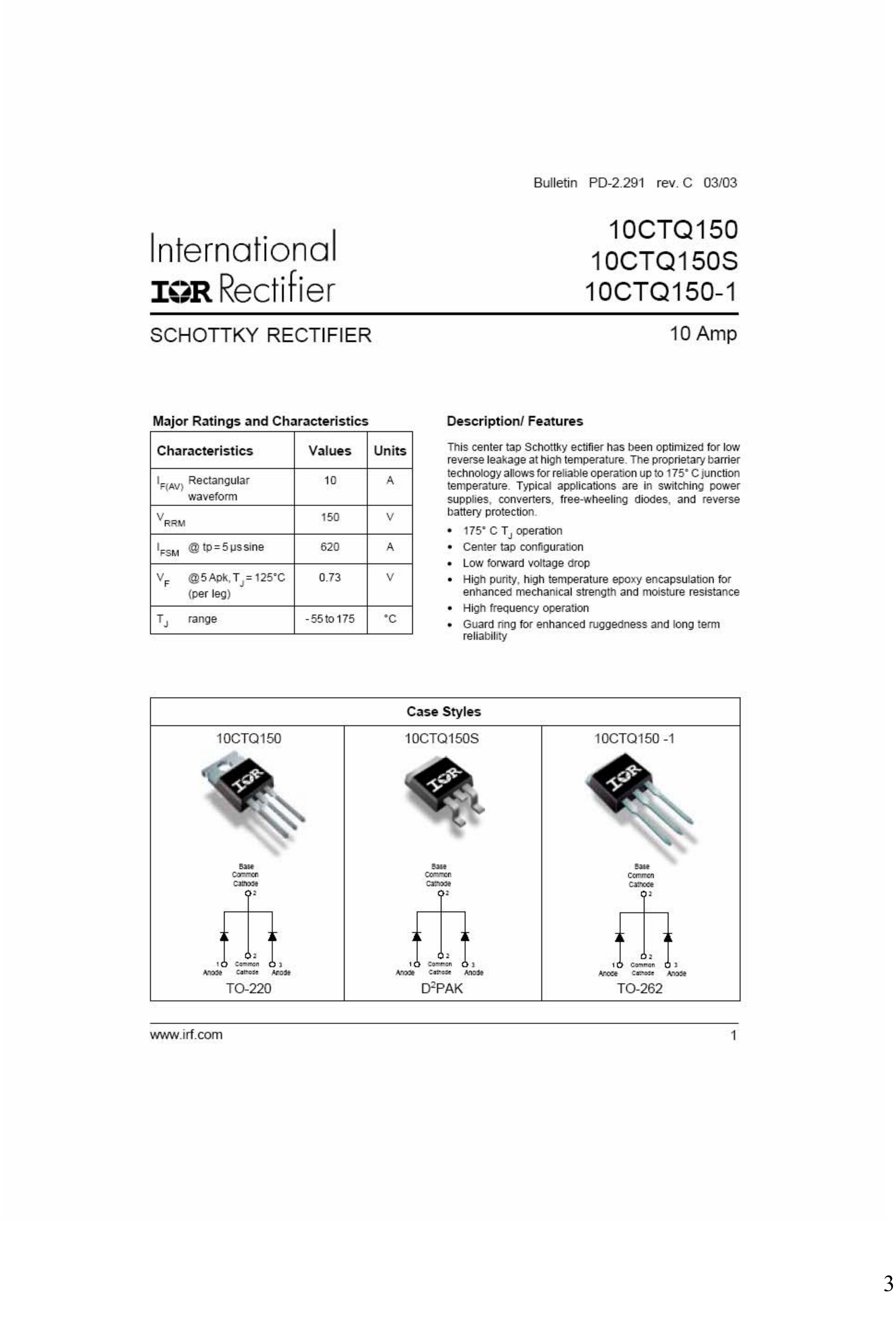
PARAMETER	SYMBOL	TEST CONDITIONS		MIN	TYP	MAX	UNITS
Drain to Source Breakdown Voltage	BV_{DSS}	$I_D = 250mA$, $V_{GS} = 0V$, Figure 10		50	-	-	V
Gate to Threshold Voltage	$V_{GS(TH)}$	$V_{GS} = V_{DS}$, $I_D = 250mA$, Figure 9		1	-	2	V
Zero Gate Voltage Drain Current	I_{DSS}	$V_{DS} = 40V$, $V_{GS} = 0V$		-	-	1	µA
		$T_C = 150^oC$		-	-	50	µA
Gate to Source Leakage Current	I_{GSS}	$V_{GS} = \pm10V$, $V_{DS} = 0V$		-	-	100	nA
Drain to Source On Resistance (Note 2)	$r_{DS(ON)}$	$I_D = 16A$, $V_{GS} = 5V$		-	-	0.047	Ω
		$I_D = 16A$, $V_{GS} = 4V$		-	-	0.056	Ω
Turn-On Time	$t_{(ON)}$	$V_{DD} = 25V$, $I_D = 8A$, $V_{GS} = 5V$, $R_{GS} = 12.5\Omega$ Figures 15, 16		-	-	60	ns
Turn-On Delay Time	$t_{d(ON)}$			-	14	-	ns
Rise Time	t_r			-	30	-	ns
Turn-Off Delay Time	$t_{d(OFF)}$			-	42	-	ns
Fall Time	t_f			-	14	-	ns
Turn-Off Time	$t_{(OFF)}$			-	-	100	ns
Total Gate Charge	$Q_{g(TOT)}$	$V_{GS} = 0V$ to $10V$	$V_{DD} = 40V$, $I_D = 16A$, $R_L = 2.5\Omega$ Figures 17, 18	-	-	80	nC
Gate Charge at 5V	$Q_{g(5)}$	$V_{GS} = 0V$ to $5V$		-	-	45	nC
Threshold Gate Charge	$Q_{g(TH)}$	$V_{GS} = 0V$ to $1V$		-	-	3	nC
Thermal Resistance Junction to Case	$R_{\theta JC}$			-	-	2.083	°C/W
Thermal Resistance Junction to Ambient	$R_{\theta JA}$			-	-	100	°C/W

Source to Drain Diode Specifications

PARAMETER	SYMBOL	TEST CONDITIONS	MIN	TYP	MAX	UNITS
Source to Drain Diode Voltage	V_{SD}	$I_{SD} = 16A$	-	-	1.5	V
Diode Reverse Recovery Time	t_{rr}	$I_{SD} = 16A$, $dI_{SD}/dt = 100A/\mu s$	-	-	125	ns

NOTES:

2. Pulse Test: Pulse Width ≤ 300ms, Duty Cycle ≤ 2%.
3. Repetitive Rating: Pulse Width limited by max junction temperature.



Bulletin PD-2.291 rev. C 03/03

International
IOR Rectifier

10CTQ150
10CTQ150S
10CTQ150-1

SCHOTTKY RECTIFIER

10 Amp

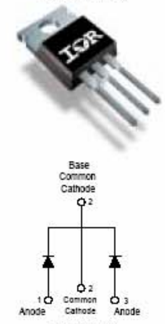
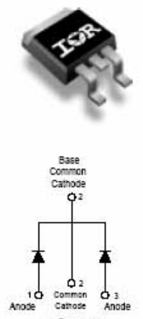
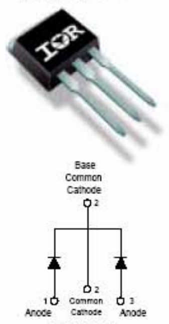
Major Ratings and Characteristics

Characteristics	Values	Units
$I_{F(AV)}$ Rectangular waveform	10	A
V_{RRM}	150	V
I_{FSM} @ tp = 5 µs sine	620	A
V_F @ 5 Apk, T_J = 125°C (per leg)	0.73	V
T_J range	- 55 to 175	°C

Description/ Features

This center tap Schottky ectifier has been optimized for low reverse leakage at high temperature. The proprietary barrier technology allows for reliable operation up to 175° C junction temperature. Typical applications are in switching power supplies, converters, free-wheeling diodes, and reverse battery protection.

- 175° C T_J operation
- Center tap configuration
- Low forward voltage drop
- High purity, high temperature epoxy encapsulation for enhanced mechanical strength and moisture resistance
- High frequency operation
- Guard ring for enhanced ruggedness and long term reliability

Case Styles		
10CTQ150	10CTQ150S	10CTQ150 -1
 Base Common Cathode 2 1 Anode 2 Common Cathode 3 Anode	 Base Common Cathode 2 1 Anode 2 Common Cathode 3 Anode	 Base Common Cathode 2 1 Anode 2 Common Cathode 3 Anode
TO-220	D^2PAK	TO-262

www.irf.com 1

Voltage Ratings

Parameters		10CTQ150 10CTQ150S 10CTQ150-1
V_R	Max. DC Reverse Voltage (V)	150
V_{RWM}	Max. Working Peak Reverse Voltage (V)	

Absolute Maximum Ratings

Parameters		Values	Units	Conditions	
$I_{F(AV)}$	Max. Average Forward (Per Leg)	5	A	50% duty cycle @ T_C = 155°C, rectangular wave form	
	Current * See Fig. 5 (Per Device)	10			
I_{FSM}	Max. Peak One Cycle Non-Repetitive	620	A	5µs Sine or 3µs Rect. pulse	Following any rated load condition and with rated V_{RRM} applied
	Surge Current (Per Leg) * See Fig. 7	115		10ms Sine or 6ms Rect. pulse	
E_{AS}	Non-Repetitive Avalanche Energy (Per Leg)	6.75	mJ	T_J = 25 °C, I_{AS} = 0.30 Amps, L = 150 mH	
I_{AR}	Repetitive Avalanche Current (Per Leg)	0.30	A	Current decaying linearly to zero in 1 µsec Frequency limited by T_J max. V_A = 1.5 x V_R typical	

Electrical Specifications

Parameters		Values	Units	Conditions	
V_{FM}	Max. Forward Voltage Drop	0.93	V	@ 5A	T_J = 25 °C
	(Per Leg) * See Fig. 1 (1)	1.10	V	@ 10A	
		0.73	V	@ 5A	T_J = 125 °C
		0.86	V	@ 10A	
I_{RM}	Max. Reverse Leakage Current	0.05	mA	T_J = 25 °C	V_R = rated V_R
	(Per Leg) * See Fig. 2 (1)	7	mA	T_J = 125 °C	
$V_{F(TO)}$	Threshold Voltage	0.468	V	T_J = T_J max.	
r_t	Forward Slope Resistance	28	mΩ		
C_T	Max. Junction Capacitance(Per Leg)	200	pF	V_R = 5V_{DC}, (test signal range 100Khz to 1Mhz) 25°C	
L_S	Typical Series Inductance (Per Leg)	8.0	nH	Measured lead to lead 5mm from package body	
dv/dt	Max. Voltage Rate of Change (Rated V_R)	10000	V/ µs		

(1) Pulse Width < 300µs, Duty Cycle <2%

Thermal-Mechanical Specifications

Parameters			Values	Units	Conditions
T_J	Max. Junction Temperature Range		-55 to 175	°C	
T_{stg}	Max. Storage Temperature Range		-55 to 175	°C	
R_{thJC}	Max. Thermal Resistance Junction to Case (Per Leg)		3.50	°C/W	DC operation
R_{thJC}	Max. Thermal Resistance Junction to Case (Per Package)		1.75	°C/W	DC operation
R_{thCS}	Typical Thermal Resistance, Case to Heatsink (only for TO-220)		0.50	°C/W	Mounting surface, smooth and greased
wt	Approximate Weight		2 (0.07)	g (oz.)	
T	Mounting Torque	Min.	6 (5)	Kg-cm (lbf-in)	
		Max.	12 (10)		

Appendix B: Computer Code

PIC-C Code: Phi

```
/*
  Emery Ku, E90

  This file details the I/O for a PIC microcontroller (16F873A) which is
  connected to an angular encoder (1024 pulses/revolution) and outputs a pulse-
  width modulated signal to convert angular position to a voltage.
*/

//Things we need.
#include <16F873.h>  //ADC set to 10 (open 16F873.h)
#include <STDLIB.H>  //Required by read_adc()
#fuses HS,NOWDT,NOPROTECT,NOLVP
#use delay(clock=10000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7, BRGH1OK) // Jumpers: 8 to 11, 7 to 12

//Define quadrature-based states as grey-code.
#define  state_0 0   //Both encoder outputs A and B are low
#define  state_1 1   //A is high, B is low
#define  state_2 3   //Both encoder outputs A and B are high
#define  state_3 2   //A is low, B is high

//Inputs
#use fast_io(A)      //this requires the set_tris_X command
//#use fixed_io(a_inputs=PIN_A0)              //A0 is the motor control voltage
//#use fixed_io(b_inputs=PIN_B0, PIN_B1)       //B0 and B1 are encoder inputs
                                        //B0 is A from encoder
                                        //B1 is B from encoder

//Outputs  << I'm not sure we need this any more? CCP is separate from RCX >>
//#use fixed_io(c_outputs=PIN_C1,PIN_C2)       //Pin C1 is PWM output (DAQ)
                                      //Pin C2 is PWM output (motors)


long int angle;  //Keep track of the angular position (10-bit)
//int angle;  //Keep track of the angular position (10-bit)
long int control;
int cur_state;
int old_state;

void main()
{
  SET_TRIS_A( 0x0F );
  // A7,A6,A5,A4 are outputs
  // A3,A2,A1,A0 are inputs
  SET_TRIS_B( 0x0F );
  // B7,B6,B5,B4 are outputs
  // B3,B2,B1,B0 are inputs

  setup_ccp1(CCP_PWM);   // Configure CCP1 and CCP2 as a PWM
  setup_ccp2(CCP_PWM);
      //   The cycle time will be (1/clock)*4*t2div*(period+1)
             //   In this program clock=10000000 and period=127 (below)
      //   For the three possible selections the cycle time is:
                    //     (1/10000000)*4*1*128 =  51.2 us or 19.5 khz
  setup_timer_2(T2_DIV_BY_1, 127, 1); //  20 KHz signal

  setup_port_a(ALL_ANALOG);
  setup_adc(adc_clock_internal);
  set_adc_channel( 0 );   //A0 is the motor control voltage pin

  angle = 0;

  //Loop to collect/output data
  do {
    control=read_adc();  //Control should be a 10-bit long int, set in #device

    cur_state = input(PIN_B0) + 2*input(PIN_B1);

    if (cur_state != old_state)
    {
      if (cur_state == state_1 && old_state == state_0 && angle > 200) //65400 for theta
      {
        //angle+=64; //Encoder shaft is turning clockwise (theta)
        angle-=256; //Use this for the phi-direction encoder
      }
      if (cur_state == state_3 && old_state == state_0 && angle < 65200)
      {
        //angle-=64; //Encoder shaft is turning CCW (theta)
        angle+=256;  //The max output comes to 1.04V if left @ +/-64 (phi)
      }
    }

    old_state = cur_state;  //For the next cycle...

////////////////////////////////////////////////////////////////////////////////
///////////////////////////////  Outputs  ////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

    //Need to convert angle to a voltage to output to DAQ
    set_pwm1_duty(angle/128);
```

```
        //This should be okay... angle is a long int

    //Need to output our motor control voltage as PWM:
    set_pwm2_duty(control/2);
        //control reads in from 0 to 1024 :-)
        //value*(1/clock)*t2div
        //value may be an 8 or 16 bit constant or variable
        //want duty cycle to be some fraction of 51.2 us
        //What is the range of control??  I think I set it to 10 bits
  } while(1);
}
```

PIC-C Code: Theta

```
/*
  Emery Ku, E90

  This file details the I/O for a PIC microcontroller (16F873A) which is
  connected to an angular encoder (1024 pulses/revolution) and outputs a pulse-
  width modulated signal to convert angular position to a voltage.
*/

//Things we need.
#include <16F873.h>  //ADC set to 10 (open 16F873.h)
#include <STDLIB.H>  //Required by read_adc()
#fuses HS,NOWDT,NOPROTECT,NOLVP
#use delay(clock=10000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7, BRGH1OK) // Jumpers: 8 to 11, 7 to 12

//Define quadrature-based states as grey-code.
#define  state_0 0   //Both encoder outputs A and B are low
#define  state_1 1   //A is high, B is low
#define  state_2 3   //Both encoder outputs A and B are high
#define  state_3 2   //A is low, B is high

//Inputs
#use fast_io(A)      //this requires the set_tris_X command
//#use fixed_io(a_inputs=PIN_A0)              //A0 is the motor control voltage
//#use fixed_io(b_inputs=PIN_B0, PIN_B1)       //B0 and B1 are encoder inputs
                                  //B0 is A from encoder
                                  //B1 is B from encoder

//Outputs  << I'm not sure we need this any more? CCP is separate from RCX >>
//#use fixed_io(c_outputs=PIN_C1,PIN_C2)       //Pin C1 is PWM output (DAQ)
                                //Pin C2 is PWM output (motors)
```

```
long int angle;  //Keep track of the angular position (10-bit)
//int angle;  //Keep track of the angular position (10-bit)
long int control;
int cur_state;
int old_state;

void main()
{
  SET_TRIS_A( 0x0F );
  // A7,A6,A5,A4 are outputs
  // A3,A2,A1,A0 are inputs
  SET_TRIS_B( 0x0F );
  // B7,B6,B5,B4 are outputs
  // B3,B2,B1,B0 are inputs

  setup_ccp1(CCP_PWM);   // Configure CCP1 and CCP2 as a PWM
  setup_ccp2(CCP_PWM);
       //   The cycle time will be (1/clock)*4*t2div*(period+1)
              //   In this program clock=10000000 and period=127 (below)
       //   For the three possible selections the cycle time is:
                     //     (1/10000000)*4*1*128 =  51.2 us or 19.5 khz
  setup_timer_2(T2_DIV_BY_1, 127, 1); //  20 KHz signal

  setup_port_a(ALL_ANALOG);
  setup_adc(adc_clock_internal);
  set_adc_channel( 0 );   //A0 is the motor control voltage pin

  angle = 0;

  //Loop to collect/output data
  do {
    control=read_adc();  //Control should be a 10-bit long int, set in #device

    cur_state = input(PIN_B0) + 2*input(PIN_B1);

    if (cur_state != old_state)
    {
      if (cur_state == state_1 && old_state == state_0 && angle < 65400) //65400 for theta
      {
        angle+=64; //Encoder shaft is turning clockwise (theta)
        //angle-=64; //Use this for the phi-direction encoder
      }
      if (cur_state == state_3 && old_state == state_0 && angle != 0)
      {
        angle-=64; //Encoder shaft is turning CCW (theta)
        //angle+=64;  //The max output comes to 1.04V if left @ +/-64 (phi)
      }
    }

    old_state = cur_state;  //For the next cycle...

/////////////////////////////////////////////////////////////////////////////
///////////////////////////////  Outputs  //////////////////////////////////
/////////////////////////////////////////////////////////////////////////////

    //Need to convert angle to a voltage to output to DAQ
    set_pwm1_duty(angle/128);
        //This should be okay... angle is a long int

    //Need to output our motor control voltage as PWM:
    set_pwm2_duty(control/2);
        //control reads in from 0 to 1024 :-)
        //value*(1/clock)*t2div
        //value may be an 8 or 16 bit constant or variable
        //want duty cycle to be some fraction of 51.2 us
        //What is the range of control??  Set it to 10 bits
  } while(1);
}
```

MATLAB Script

Ku_E90_MIMO.m

```
Written by Emery Ku
E90 Data Acquisition and Control Algorithm
```

```
%==========================================================================%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%% Emery's Script %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%% Two-Input/Two-Output Control %%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%==========================================================================%
clear;                                          %Clean up from last time
clc;
close all;
%==========================================================================%
InChans=[0 1];                                  %DAQ Board Input channels
OutChans=[0 1];                                 %DAQ Board Output channels
%==========================================================================%
```

Setup the input channel

```
NumSecs = 6;
Ain = analoginput('nidaq');
Ain.SampleRate=100;
NumPts = Ain.SampleRate*NumSecs;
Ain.SamplesPerTrigger=NumPts;
Ain.InputType='SingleEnded';
Ain.BufferingConfig=[1  2000];
Ain.TransferMode='Interrupts';
Ain.TriggerRepeat=inf;
DT = 1/Ain.SampleRate;
%==========================================================================%
```

Setup the output channel

```
Aout = analogoutput('nidaq');
Aout.SampleRate=Ain.SampleRate;
%==========================================================================%
```

Initialize I/O

```
Inputs = addchannel(Ain,InChans);
Outputs = addchannel(Aout,OutChans);

% For whatever reason, NIDAQ insists on setting its input range to +/-5V.
% Other settings (even those allowed by hardware) give anomalous results.
set(Inputs, {'Units', 'UnitsRange'}, {'Volts', [-5 5]})
%==========================================================================%
```

Pre-Set Output

```
Phi_Pre = 1;            Theta_Pre = 0;

putsample(Aout,[Phi_Pre Theta_Pre]);            %for now, no change in theta
pause(1.5);                                     %Spin up
%==========================================================================%
```

Predefine variables & Initialize Arrays

```
Phi=[];                 Theta=[];               %Acquired Data (degrees)
Phi_Vs=[];              Theta_Vs=[];            %Acquired Data (V)

Time=[];                counter=0;              %Time vector & counting

Phi_Error = [];         Theta_Error = [];       %Save calculated errors
Phi_Error_Int = [];     Theta_Error_Int = [];
Phi_Error_Der = [];     Theta_Error_Der = [];

Out0=[];                Out1=[];                %Save outputs

Phi_Out_p = [];         Theta_Out_p = [];       %Save outputs of each
Phi_Out_i = [];         Theta_Out_i = [];       %part of the PID controller
Phi_Out_d = [];         Theta_Out_d = [];

Phi_IntError=0;         Theta_IntError=0;       %Initialize Integral
Phi_IntError1=0;        Theta_IntError1=0;      %Controller
%====================================================================%
```

Goal Angles (and voltages)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%% Phi Voltage-Angle Conversion %%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%% ~75-degree Range of motion   %%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%% ~2V = 0 degrees             %%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Phi_Goal_Angle = -15;                           %Range from [-37 ~20] deg
Phi_Goal = 2.2*(Phi_Goal_Angle)/37.5 + 2.2;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%% Theta Voltage-Angle Conversion %%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%% ~75-degree Range of motion   %%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%% ~2V = 0 degrees             %%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Theta_Goal_Angle = 180;                         %Range from [0 to 345] deg
Theta_Goal = (Theta_Goal_Angle/345)*4.65;
%====================================================================%
```

Set Controller Parameters

```
%/\/\/\/\/\/\/\/\/\/\ Phi Controller Parameters /\/\/\/\/\/\/\/\/\/\/\/\%
Kp_a=0.3;                                      %For proportional control:
Kp_b=0.2;   %(Error)*[abs(Kp_a*sin(phi_desired)) + Kp_b*cos(phi_absolute)]
%--------------------------------------------------------------------%

%Different Ranges implies Diff. Control Factors.  Want a more aggressive
%integral control factor at smaller angles: less chance of dangerous
%overshoot.
if Phi_Goal_Angle < -2
    Ki = 0.00;                                 %Int(Error) Control Factor
    Kd = .3;                                    %Der(Error) Control Factor
else
    Ki = 0.005;
    Kd = .23;
end
%--------------------------------------------------------------------%
%"Modelling the System"
%%% V0      Vchan1  %%%
%   2.0     0.15
%   2.1     0.3
%   2.2     0.5
%   2.3     0.85
%   2.4     1
%   2.5     1.9
%   2.52    2.35
%   Excel says Phi in V = 9e-6*exp(4.9159*V_to_PIC)
%   This suggests: V0_phi =
%                         100       V
%                         --- log(------)
%                         492      9e-6
%-----------------------------------------------------------------------%

if Phi_Goal_Angle < 6
    %V0_Phi = 1/4.92*log(Phi_Goal/9e-6)-.05;     %This model isn't perfect.
    V0_Phi = 1.62;
else
    V0_Phi = 1/4.92*log(Phi_Goal/9e-6)-.08;
end
%=======================================================================%

%/\/\/\/\/\/\/\/\/\/\ Theta Controller Parameters /\/\/\/\/\/\/\/\/\/\/\/\%
Qp_a=1;                                      %For proportional control:
Qp_b=0.3;     %(Err)*[abs(Qp_a*sin(theta_desired)) + Qp_b*cos(theta_abs)]
%-----------------------------------------------------------------------%
Qi = 0.0003;                                     %Int(Error) Control Factor
%-----------------------------------------------------------------------%
Qd = 2;                                         %Der(Error) Control Factor
%=======================================================================%
```

Get ready...

```
start(Ain);
%=======================================================================%
```

Loop until time t

```
while counter < NumPts
    counter=counter+1;                      %Keep track of cycles
    [InData, InTime]=getdata(Ain,1);        %Get the Data
    Theta=[Theta InData(:,1)];              %Who knows why Chan 1 is the
    Phi=[Phi InData(:,2)];                  %first row of data
    Time=[Time InTime];                     %Time vector
%-----------------------------------------------------------------------%
    Phi_V = Phi(counter);                   %Remember raw voltages
    Theta_V = Theta(counter);

    Phi_Vs = [Phi_Vs Phi_V];                %Collect Phi(V)s
    Theta_Vs = [Theta_Vs Theta_V];          %Collect Theta(V)s
%-----------------------------------------------------------------------%
    Phi(counter) = ( (Phi(counter)-2)/2 )*37.5;      %Convert Phi to degrees
    Theta(counter) = ( (Theta(counter)/4.74)*345 ); %and the same for Theta
%-----------------------------------------------------------------------%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%<<<<<<<<<<<<<<<<<<% Begin Phi Controller Algorithm %>>>>>>>>>>>>>>>>>>>>>>%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Phi Angle error in rad
    angle_error = (Phi_Goal_Angle - Phi(counter))*pi/180;
%Phi--------------------------------------------------------------------%
    %Proportional Error Calculation
    Phi_Error_now = (Phi_Goal - Phi_V);     %Proportional Error Factor
    Phi_Error = [Phi_Error Phi_Error_now];  %Save Error

    if Phi_Error_now < 0                    %If past goal angle, increase
        Kp_aa = 2*Kp_a;                    %proportional correction factors
        Kp_bb = 2*Kp_b;
    else
        Kp_aa = Kp_a;
        Kp_bb = Kp_b;
    end
%Phi--------------------------------------------------------------------%
    %Integrated Error Calculation
    if Phi(counter) > -35       %Is the helicopter at least 2 degrees up?

        %         if -4 < (Phi(counter) - Phi_Goal_Angle) && (Phi(counter) -
Phi_Goal_Angle) < 10
        %             Phi_IE1 = (Phi_Goal - Phi_Vs(counter-1));
        %             %         elseif abs(Phi(counter) - Phi_Goal_Angle) < 4
        %             %             Phi_IE1 = 0;
        %         else
        %             Phi_IE1 = 0;
        %         end
        %         Phi_IntError = Phi_IntError + (Phi_Error_now + Phi_IE1)*(DT/2);

        if Phi_Error_now > 0
            Phi_IntError = Phi_IntError + (Phi_Error(counter))*(DT)*cos(angle_error)^3;
        else
            if Phi_Goal_Angle < 5
                Phi_IntError = Phi_IntError +
5*(Phi_Error(counter))*(DT)*cos(angle_error);
            else
                Phi_IntError = Phi_IntError + 10*(Phi_Error(counter))*(DT);
            end
        end
%Phi---------------------------------------------------------------------%
        %Derivative Error Calculation
        if counter > 1
            %if we're below our target, do normal derivative of error
            if Phi_Error_now > 0
                Phi_DE1 = (Phi_Error(counter) - Phi_Error(counter-1))/DT;

                %otherwise scale derivative of error
            else
                Phi_DE1 = .6*(Phi_Error(counter) - Phi_Error(counter-1))/DT;
            end
            %If we're on the first sample, you can't take the derivative
        else
            Phi_DE1 = 0;
        end
%Phi---------------------------------------------------------------------%
        %If the 'copter is sitting still, set integral and derivative errors to
        %zero.
    else
        Phi_IntError = 0;
        Phi_DE1 = 0;
    end
%Phi---------------------------------------------------------------------%
    Phi_Error_Int = [Phi_Error_Int Phi_IntError];   %Save Int Error
    Phi_Error_Der = [Phi_Error_Der Phi_DE1];        %Save d/dt(error)
%Phi---------------------------------------------------------------------%
    %Final Output Calculation
    Phi_Out_p = [Phi_Out_p (Phi_Error_now)*(abs(Kp_aa*sin(angle_error))...
        + Kp_bb*cos(Phi(counter)*pi/180))];
    Phi_Out_i = [Phi_Out_i Ki*Phi_Error_Int(counter)*(cos(Phi(counter)*pi/180))];
    Phi_Out_d = [Phi_Out_d Kd*Phi_DE1*(cos(angle_error))];

    %Output is the sum of above:
    OutData0= V0_Phi + Phi_Out_p(counter) + Phi_Out_i(counter) + Phi_Out_d(counter);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%<<<<<<<<<<<<<<<<<<<% End Phi Controller Algorithm %>>>>>>>>>>>>>>>>>>>>>>%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%=========================================================================%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%<<<<<<<<<<<<<<<<% Begin Theta Controller Algorithm %>>>>>>>>>>>>>>>>>>>>>%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Theta Angle error in rad
    angle_error = (Theta_Goal_Angle - Theta(counter))*pi/180;
%Theta-------------------------------------------------------------------%
    %Proportional Error Calculation
    Theta_Error_now = (Theta_Goal - Theta_V);     %Proportional Error Factor
    Theta_Error = [Theta_Error Theta_Error_now];  %Save Error
%Theta-------------------------------------------------------------------%
    %Integrated Error Calculation (perform only when close to goal)
    Theta_IntError = Theta_IntError + (Theta_Error(counter))*(DT)*cos(angle_error)^3;
%Theta-------------------------------------------------------------------%
    %Derivative Error Calculation (perform only after first sample)
    if counter > 1
        Theta_DE1 = (Theta_Error(counter) - Theta_Error(counter-1))/DT;
    else
        Theta_DE1 = 0;
    end
%Theta-------------------------------------------------------------------%
    Theta_Error_Int = [Theta_Error_Int Theta_IntError];   %Save Int Error
    Theta_Error_Der = [Theta_Error_Der Theta_DE1];        %Save d/dt(error)
%Theta-------------------------------------------------------------------%
```

```
    %Final Output Calculation
    Theta_Out_p = [Theta_Out_p (Theta_Error_now)*(abs(Qp_a*sin(angle_error))...
        + Qp_b*cos(Theta(counter)*pi/180))];
    Theta_Out_i = [Theta_Out_i Qi*Theta_Error_Int(counter)];
    Theta_Out_d = [Theta_Out_d Qd*Theta_DE1*(cos(angle_error))];

    %Output is the sum of above:
    %OutData1= Theta_Out_p(counter) + Theta_Out_i(counter) + Theta_Out_d(counter);
    OutData1 = 0; %for testing purposes
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%<<<<<<<<<<<<<<<<<<% End Theta Controller Algorithm %>>>>>>>>>>>>>>>>>>>>>>%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %Don't send outputs that you know you can't send...
    %Phi output is 5V max
    if OutData0 > 5
        OutData0 = 5;
    elseif OutData0 < 0
        OutData0 = 0;
    end
%-------------------------------------------------------------------------%
    %Theta output is 3V max (shouldn't need to go above this)
    if OutData1 > 2.5
        OutData1 = 2.5;
    elseif OutData1 < 0
        OutData1 = 0;
    end
%-------------------------------------------------------------------------%
    %Send the Outputs and Collect Output Data
    putsample(Aout,[OutData0 OutData1]);
    Out0=[Out0 OutData0];
    Out1=[Out1 OutData1];
end
%=========================================================================%
```

Post Data-Acquisition Outputs

```
%putsample(Aout,[Out0(end) 0]);

% pause(1.8);
 putsample(Aout,[0 0]);                              %Reset Output
%=========================================================================%
```

Free up any memory that we used.

```
stop(Ain)
stop(Aout)
delete(Ain)
clear Ain
delete(Aout)
clear Aout
%=========================================================================%
```

Plot the data

```
figure %for determining where stuff went wrong in the phi direction
plot(Time,Phi_Error,'m.',Time,Out0,'b.',Time,Phi_Out_p,'y.',Time,Phi_Out_i,'r.',Time,Phi_O
ut_d,'g.',...
    Time,V0_Phi*ones(max(size(Time))),'k-.')
legend('Error','Total Output','Proportional Output','Integral Ouptut','Derivative
Output','V0','Location','Northwest')
title('Output due to Errors for Phi');
%-------------------------------------------------------------------------%
% figure %for determining where stuff went wrong in the theta direction
%
plot(Time,Theta_Error,'m.',Time,Out1,'b.',Time,Theta_Out_p,'y.',Time,Theta_Out_i,'r.',Time
,Theta_Out_d,'g.')
% legend('Error','Total Output','Proportional Output','Integral Ouptut','Derivative
Output','Location','Northwest')
```

```
% title('Output due to Errors for Theta');
%=====================================================================%
```

Optional plots

```
if 0    %Don't show lest you want to be fancy.. poor computer can't take it
    %  Actual positions & outputs
    subplot(4,1,1), plot(Time,Phi)
    hold on
    plot(Time,[Phi_Goal_Angle*ones(1,max(size(Time)))],'r-');
    hold off
    xlabel('Time (s)');
    ylabel('Voltage (V)');
    title('Vertical Angle (Voltage) vs time');

    subplot(4,1,3), plot(Time,Theta)
    xlabel('Time (s)');
    ylabel('Voltage (V)');
    title('Horizontal Angle (Voltage) vs time');

    subplot(4,1,2), plot(Time,Out0)
    xlabel('Time (s)');
    ylabel('Voltage');
    title('Phi Output Voltage vs time');

    subplot(4,1,4), plot(Time,Out1)
    xlabel('Time (s)');
    ylabel('Voltage (V)');
    title('Theta Output Voltage vs time');

    figure %for Errors
    subplot(2,1,1), plot(Time,Phi_Error)
    hold on
    plot(Time,[zeros(1,max(size(Time)))],'r-');
    hold off
    xlabel('Time (s)');
    ylabel('Error (V)');
    title('Phi Angle Error (V) vs time');

    subplot(2,1,2), plot(Time,Theta_Error)
    xlabel('Time (s)');
    ylabel('Error (V)');
    title('Theta Angle Error (V) vs time');
end

%=====================================================================%

%The little birdie says we're done
load chirp.mat
wavplay(y(1:4500),Fs);
clear y Fs
disp('Data Acuisition done.');

%=====================================================================%
%=====================================================================%
%=====================================================================%
```