

## ECE 3084A Lab 3: Motor Velocity Control

Date Lab Performed: \_\_\_\_\_

Name/Section of Partner 1: \_\_\_\_\_

Name/Section of Partner 2: \_\_\_\_\_

Name/Section of Partner 3: \_\_\_\_\_

Name/Section of Partner 4: \_\_\_\_\_

### Goals:

1. Investigate P, PI, and PID speed control for a DC motor.
2. Observe tracking performance for time-varying references.
3. Determine appropriate gains to achieve given best system performance.
4. Understand trade-offs between various system performance parameters like tracking, stability, oscillation, etc.
5. Observe and understand how a real system deviates from an ideal mathematical model.

### Preparation:

To perform this lab, you must download and install software. **WARNING!!!** This process can take a long time (1 hour or more depending on your download speed).

### PC Users

1. You should already have the Elvis suite installed, which has the myDAQ drivers. If it isn't, or if it isn't the latest version, you must install it from the following link:  
<http://www.ni.com/download/ni-elvismx-14.0/4801/en/>.  
It is over 1.4 GBytes, and takes a really long time (1-2 hours depending on your connection speed). Try the rest of the installation before doing this!
2. Install the Runtime Engine (much smaller) from:  
<http://www.ni.com/download/labview-run-time-engine-2014/4887/en/>.
3. Download the zip file "MotorControl-PC.zip" from T-Square (under Lab 3 in Resources), and extract the file "Spring15.MotorControlsLab.exe".

## Mac Users

1. myDAQ driver only: [http://www.ni.com/gate/gb/GB\\_EVALDAQMXMYDAQ/US](http://www.ni.com/gate/gb/GB_EVALDAQMXMYDAQ/US)
2. Runtime Engine: <http://www.ni.com/download/labview-run-time-engine-2014/4894/en/>
3. Download the zip file "MotorControl-MAC.zip" from T-Square (under Lab 3 in Resources), and extract the application "Spring15 MotorControlsLab".

After you have downloaded all software and rebooted your computer, it is critical that you do the following before you come to class:

1. Plug your myDAQ into your computer.
2. Run the executable (application) "Spring15 MotorControlsLab".
3. Verify that you get the startup screen shown in Figure 1 with no error messages.
4. Verify that you can start it using the small white arrow on top without getting an "Invalid Device" error (use the "Stop" button so you can close the GUI).

Also, carefully read through the entire lab in advance. Bring a printout of the lab to class with your laptop.

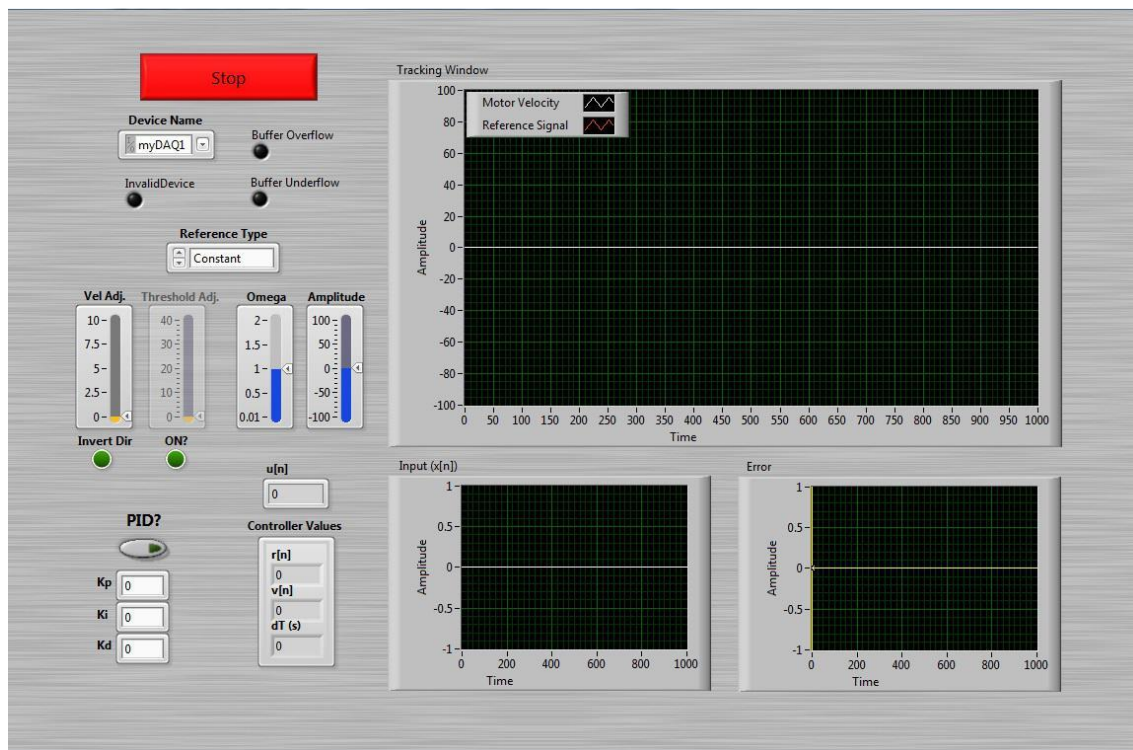


Figure 1: Screenshot of the VI after startup.

## Background:

This lab demonstrates basic controller design for velocity control of a DC motor. We are going to use the LabVIEW myDAQ equipment to control the angular velocity of the motor via a custom LabVIEW program. The motor is controlled using Pulse Width Modulation (PWM). There are two digital command signals: one is the direction (clockwise or counter clockwise) and the other is the pulse-width modulated signal that controls how much power the motor uses. The frequency of this PWM signal is constant, but the larger the duty cycle (percentage of time it is non-zero), the faster the motor turns (a 50% duty cycle means that it is "on" half of the time). We will make the approximation that the PWM duty cycle is directly proportional to the motor speed. This is one reason our open loop controller doesn't work very well (it is actually non-linear), but using feedback, we can overcome this and other modeling errors. The position of the motor shaft is measured using an incremental encoder, and the velocity is derived from the position in software.

In the time domain, we have:

1. The dynamics of the motor (the plant) that relate the input  $x(t)$  to the output  $y(t)$ , usually via a differential equation (Eq. 1).
2. The relationship between the measured output (the angle of the motor)  $y(t)$  and the value we will use for feedback (angular velocity)  $\dot{y}$  (Eq. 2).
3. An error signal  $e(t)$  { how close the actual motor velocity is to the desired velocity  $r(t)$  at each time point (Eq. 3).
4. A controller (control function) to generate the input  $x(t)$  to the motor based upon the error  $e(t)$  (Eq. 4).

$$\dot{y} = f(u; y) \quad (1)$$

$$\dot{y} = \frac{dy(t)}{dt} \quad (2)$$

$$e(t) = r(t) - \dot{y}(t) \quad (3)$$

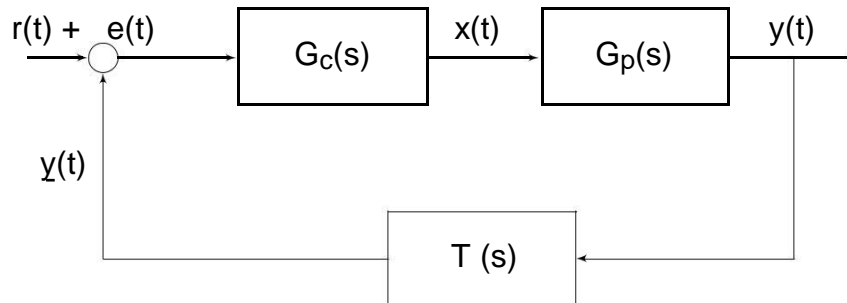
$$x(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \dot{e}(t) \quad (4)$$

The overall goal of the lab is to choose  $K_p$ ,  $K_i$ , and  $K_d$  (the gains) such that system performance specifications are successfully met.

It is useful to view Eqs. 1-4 in the Laplace domain so that we can draw a block diagram to describe the setup. Taking the Laplace transform of each equation, we can define three transfer functions corresponding to the three blocks in the diagram:

1. Between the error signal and the plant input (the controller),  $G_c(s)$ .
2. Between the input and the output (the plant),  $G_p(s)$ .

3. Between the output (shaft angular position) and the shaft angular velocity (the tachometer),  $T(s)$ .



Because the plant output  $y(t)$  (the shaft angular position) is inherently noisy, we have added a low pass filter to the differentiator to smooth the approximation of the motor's angular velocity,

$$T(s) = \frac{as}{s+a};$$

making this block a filtered differentiator. For this setup we have  $a = 15$ .

Note that

$$X(s) = G_c(s)E(s)$$

where

$$G_c(s) = K_p + \frac{K_i}{s} + K_d s;$$

In discrete time, this controller is implemented in LabView software by the myDAQ as

$$x[k] = K_p e[k] + K_i \sum_{0}^k e[k] dT + K_d \frac{e[k] - e[k-1]}{dT}$$

where  $k$  is a discrete representation for time and  $dT$  is the width of our time step. Here  $dT = 0.01$  sec.

Experiment:

Figure 2 shows the overall setup of the myDAQ, the printed circuit board (PCB) with attached motor, and the battery pack. Figure 3 shows where the terminals where the battery pack is connected. PC users can verify that your computer is seeing the myDAQ by starting the NI ELVIS Instrument launcher and opening the scope. You should see static.

1. Referring to Figures 2 and 3, plug the terminal strip type connector on the PCB into your myDAQ by aligning the outer edge pins. Then plug in the battery pack, making sure that you have the correct polarization. If you are unsure, the LED located near the bottom of the board should immediately blink once when unplugging the battery pack if it was plugged in correctly. Plugging it in the wrong way will not do immediate damage, but could damage the polarized capacitors if left plugged in for many minutes.

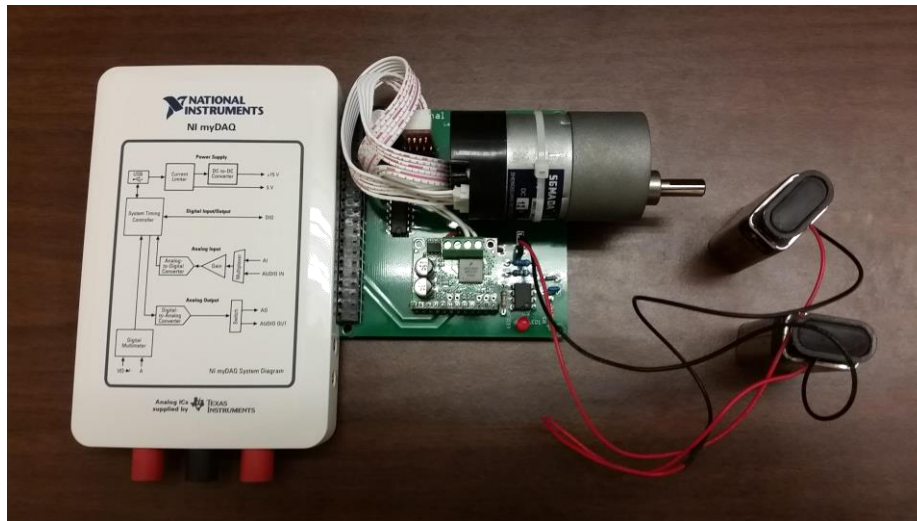


Figure 2: Overall setup of the myDAQ and motor controller.

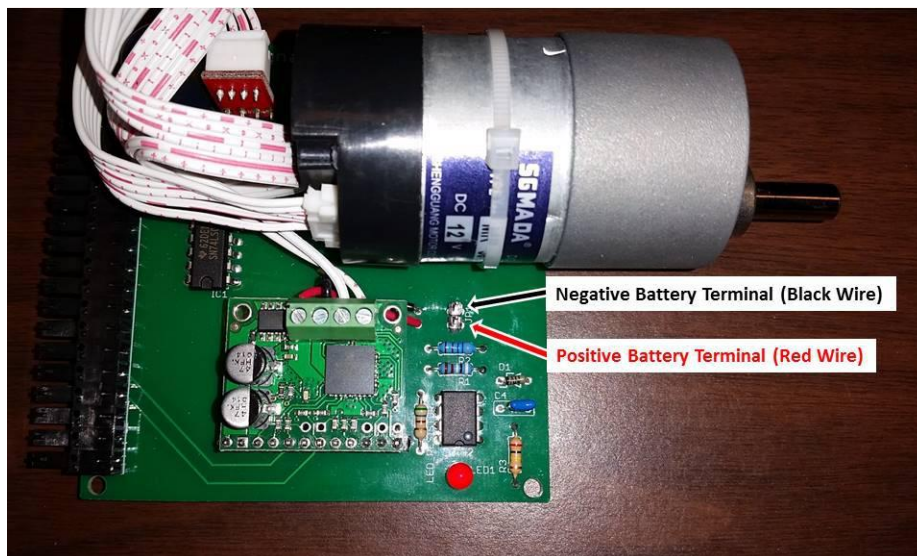


Figure 3: Closeup showing connection of the battery pack.

2. Run the NI executable (application), Spring15-MotorControlsLab, that you downloaded from T-Square. Start the VI using the little white arrow at the top of the GUI. Select a device from the pull down menu and then make sure the "Invalid Device" LED is not lit up. The screen shot of Figure 1 should be similar to what you see. Use the big red "Stop" button to stop the VI, which will let you then close it. The "Stop" button also resets the myDAQ buffers, and is a good thing to do if the GUI does not seem to be working properly.
3. In the large plot, the reference  $r[n]$  is the red signal and the measured velocity of the motor is the white one. This plot has a cursor that you may use to mark the reference and compare the behavior of different gains. The other two plots show the input signal  $x[n]$  (left) and the error  $e[n]$  (right). The error plot also has a cursor, which may be

helpful to keep near (0,0) so that you can see how well you're tracking the reference. To the left of the plots, specify the parameters of your reference signal (it can be a constant value, sine wave, or square wave). The "PID?" button allows you to toggle between open loop (light is off) and closed loop (light is on) using the PID controller with the gains you've selected. Note that for open loop control, the reference signal is directly input to the plant; i.e.,  $x(t) = r(t)$ , and there is no feedback.

4. To get started, make sure the reference type is set to "Constant" (reference is a constant amplitude) and the PID light is off. We will now calibrate the feedback signal to your specific setup. Gradually increase the amplitude slider (right one) until the motor starts to move; you are setting the reference (red) signal. You should now see the white trace (actual velocity) start to change. Note the direction of its change. If it is changing in the opposite direction of the reference (red) signal, as shown in Figure 4, turn on the "Invert Dir" button to flip the sign. This will ensure that the directions of both the desired velocity and actual velocity are the same. Now gradually increase the amplitude slider to a little less than 100%. You should see another slider in yellow labeled "Vel Adjust". Adjust this slider until both the reference signal and the estimated velocity signal are approximately the same as shown in Figure 5. This allows the measured signal to stay within the PWM range. Make sure you can get your motor to move in both directions with a reasonable velocity signal before proceeding. (Ask for help if you can't get it to work.) You have just tuned the open loop controller by making the output track the input (without feedback).

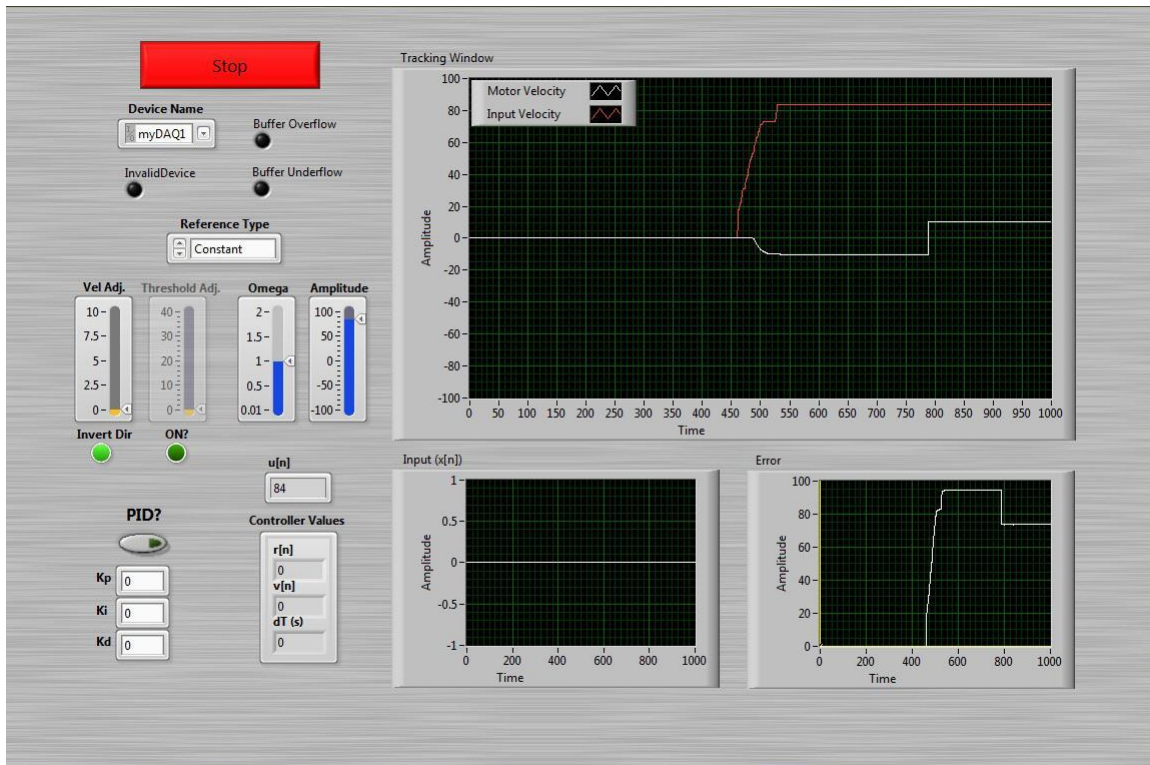


Figure 4: Screenshot showing inverted direction.



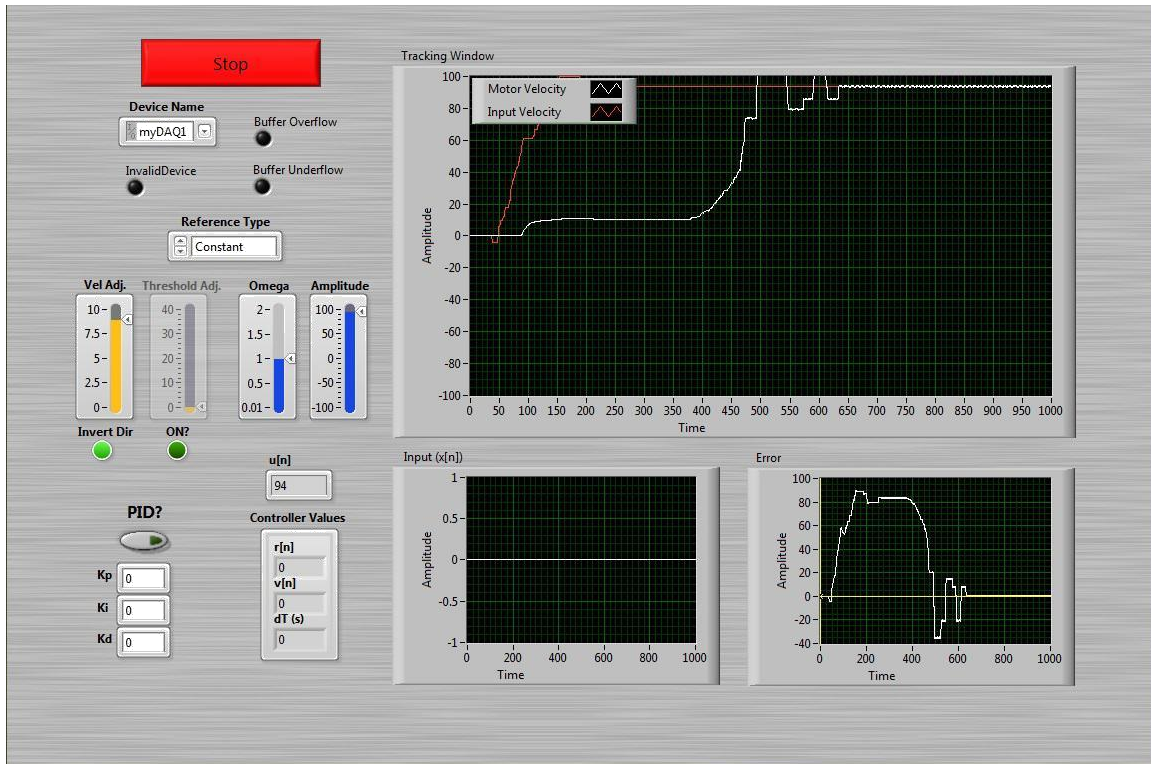


Figure 5: Screenshot showing velocity adjustment.

5. Further investigate the open loop behavior of the system for different reference inputs. For all of them, change the amplitude. For the sine and square reference types, you can also change Omega (frequency); start with Omega = 0.5. Record your observations below. How well does the velocity match the reference? Note that a square wave is essentially a step function if the period is long (small Omega).

(a) Constant

(b) Sine wave

(c) Square wave

Verification #1: Have a lab instructor verify your observations on the pre-vious page and open loop operation with a square wave reference by signing below:

Instructor Signature: \_\_\_\_\_ Time: \_\_\_\_\_

- Set the amplitude back to zero and turn on feedback with the PID button (the light should go on). Initially set  $K_p = 2$ ,  $K_i = 0$  and  $K_d = 0$ , and set the reference type to a square wave with  $\Omega = 1$ . Gradually increase the amplitude to near 100%. Your signals should look something like those in Figure 6. You may have to change  $K_p$ , but keep  $K_i = 0$  and  $K_d = 0$ . Do not proceed if your signals don't look similar to those of Figure 6 (ask for help).

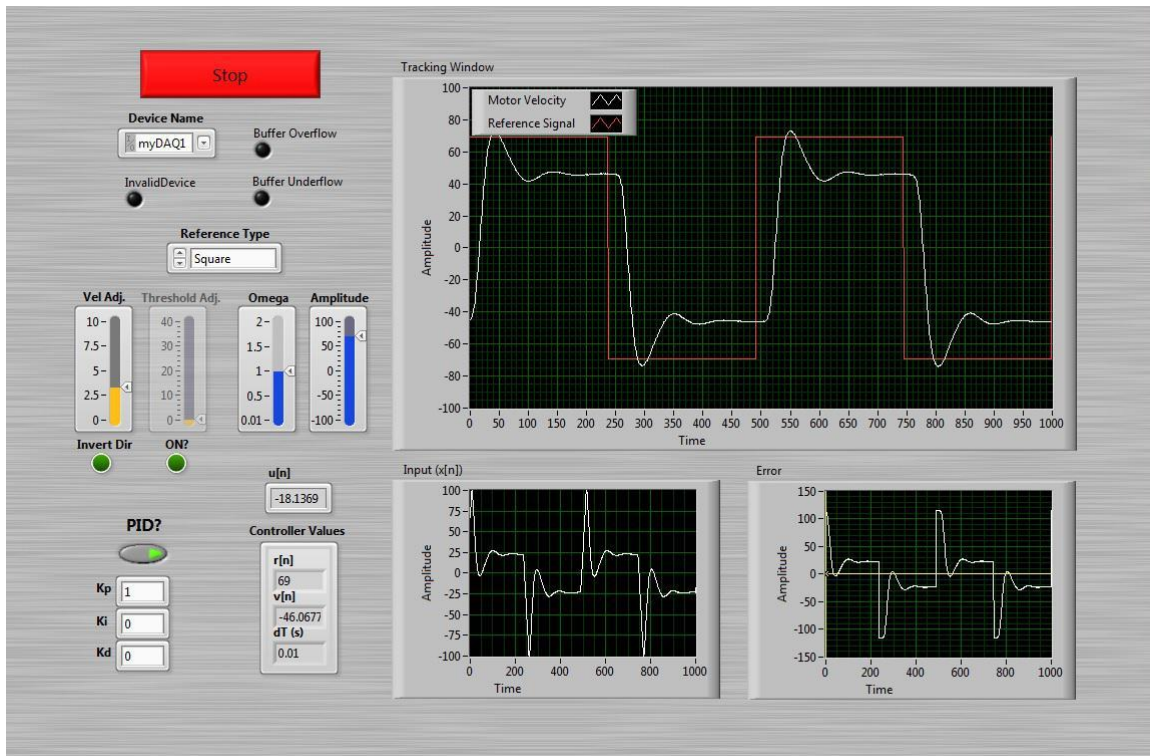


Figure 6: Screenshot of the VI with proportional control.

- Now change the reference type to a constant while keeping  $K_i = 0$  and  $K_d = 0$ . Move the amplitude up and down. Your velocity should also move up and down but the error will not be zero (remember, a 'P' controller can't provide an input to the plant with zero error).



8. With the amplitude non-zero (bigger than 50 is good), change  $K_i$  to 1. Watch the error plot in the lower right as the integral control kicks in and reduces the error. Switch back to a square wave and see how the system performs.
9. Now play around with different values for  $K_p$  and  $K_i$  (keep  $K_d = 0$  at first) to get more of a feel for what they do. It is best to initially use a square reference input at a lower frequency ( $\Omega < 1$ ) so the motor has time to reach steady state.
10. Find good<sup>1</sup> gains. Here are some guidelines:
  - (a) Start with the proportional term. You won't be able to get rid of steady-state error, but you should see the velocity somewhat mimic the basic shape of the reference without maxing out the control signal  $x[n]$  (which goes from -100 to 100). Increasing  $K_p$  should decrease the steady state error but increases oscillations when the amplitude changes.
  - (b) Move on to the integral term. This should get rid of steady state error without introducing too many oscillations (happens if  $K_i$  is too big). Try to balance  $K_p$  and  $K_i$  to get the best performance.
  - (c) Add a tiny bit of derivative term. Do you see what happens with too much? You may be able to increase  $K_p$  and  $K_i$  after you add a little  $K_d$ .  
The real world can be kind of messy! Use your judgment as you tune the gains. For a constant reference input of 80, see how low you can keep  $e[n]$ . What is  $|x[n]|$ ? This represents the magnitude of the control effort (smaller is better), and it should not be saturated (i.e., hit 100%).
  - (d) Briefly summarize your observations regarding the main effects of  $K_p$ ,  $K_i$  and  $K_d$  for this system and record your minimum  $e[n]$ .

---

<sup>1</sup>Note that there are not 3 unique values that work! Everyone will have a different answer here; do not ask if yours is "right." The point is to make observations, to see how different gains produce different closed-loop behavior, and to understand the tradeoffs associated with increasing/decreasing each of the three gains.

Verification #2: Have a lab instructor verify your observations on the pre-vious page and closed loop operation with constant, square and sine wave reference signals by signing below:

Instructor Signature: \_\_\_\_\_ Time: \_\_\_\_\_

11. For your tuned motor controller, evaluate performance as a function of frequency ( $\Omega$ ) for a sine wave reference. How well do the amplitude and phase track the reference? Summarize your observations below.
  
  
  
  
  
  
  
  
  
  
12. You have probably noticed that the motor velocity is not proportional to the control signal (i.e., it is non-linear). This problem is particularly apparent around zero in that the motor doesn't move if the control signal is too small. Enable the slider "Threshold Adj", which sets a threshold for the control signal; i.e., it forces  $|x(t)|$  to always be greater than this value. Set this adjustment to give the best performance of your motor for a sinusoidal reference. How effective is this threshold in overcoming this problem? Does it do better for some frequencies than others?