ME333: Introduction to Mechatronics
Final Project - Motor control

Milestone Demo, Part 5 – In class, Thursday 3/14
Electronic Submission – Wednesday, 3/20, 11am
Final Demo, Part 9 and Extra Credit – Sign up on google doc for either:
Monday, 3/18, 3-5pm, or Wednesday, 3/20, 9-11am

In the final project, you will create a controller for your DC motor with encoder. In Parts 1 to 5, you will control the motor based on feedback from only the encoder. In Parts 6 to 9, you will use information about the desired trajectory to control the current through the motor and make a better controller.

A demonstration of Part 5 is due in class on Thursday, 3/14. The final project is due by electronic submission on Wed 3/20 at 11am. A demo of part 9 and extra credit is due during the exam times for the two sections of the class. A google doc with sign-up times will be emailed.

There are 4 extra credit portions, listed at the end of this document.

This is a big assignment! You can take entire courses on Controls and Digital Signal Processing - we will spend a little time in class discussing these topics, but the real benefit ME333 can offer is to implement a real controller on your motor. Follow this document carefully, and use the book, google group forum and office hours if you have any questions.

This should be fun! Start early, and ask lots of questions. To encourage you to get started early, we will have a demo due of Part 5 in class on Thursday 3/14, and a **programming party on Wednesday 3/13 from 6-9pm** in L361 or M345. We will have pizza, and all of the professors and TAs will be available to answer questions.

The usual guidelines are expected – do not copy someone else's code, allow some to copy your code, or use code from the internet without attributing a source. See the Student Contract section of the ME333 wiki page.

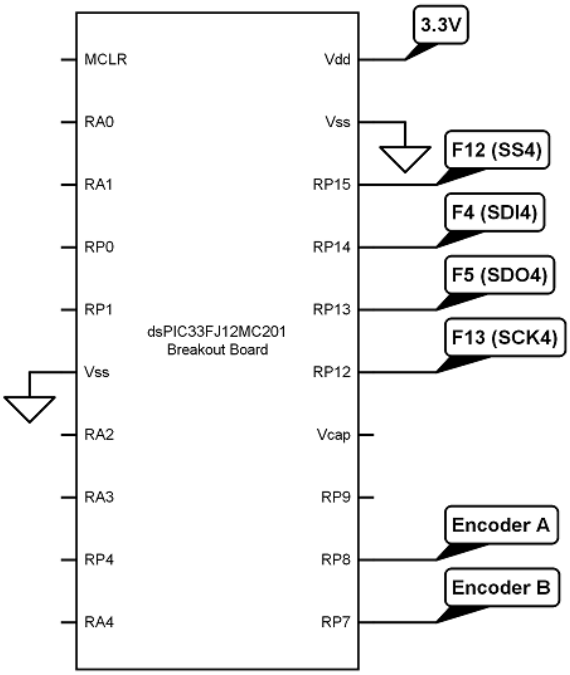
Each Part has deliverables. Code to turn in is highlighted in red, questions to answer are highlighted in blue, and images or plots to turn in are highlighted in green.

For the electronic submission, code should be named Part#_FirstnameLastname.c, images should be named Part#_FirstnameLastname.png, and all questions should be answered in FinalAnswers_FirstnameLastname.pdf, and everything should be zipped up and submitted as Final_FirstnameLastname.zip.
For example, I would submit Final_NickMarchuk.zip, filled with FinalAnswers_NickMarchuk.pdf, Part1_NickMarchuk.c, Part1_NickMarchuk.png, …

Good luck!

Part 1: Read the motor position

Wire up the following schematic for the dsPIC33FJ12MC201 and motor encoder.



Examine the dsPIC_Encoder sample code posted on the wiki at
http://hades.mech.northwestern.edu/index.php/ME333_Sample_Code#Final_Project
This code contains the functions that read the encoder value. For example,

```
init_encoderSPI(); // starts communication with the dsPIC
int encoder_counts = get_encoder(); // returns the encoder counts from the dsPIC
sprintf(out_message,“%d %d\n”, (encoder_counts – 65535/2)+255,255);
NU32_WriteUART1(out_message);
```

Note that the dsPIC initializes the encoder counts value to 65536/2 when it turns on, so move the motor shaft to the position you want it to start at before you turn on the NU32. Every time you read the encoder counts, expect to see a number in the range of 0 to 65535. Subtract 65535/2 to make the motor angle when the dsPIC turns on correspond to 0 encoder counts.

The NU32_Utility can plot numbers from 0 to 512 if the numbers are formatted with the sprintf() command above. If the integers are outside the range of 0-512, you will not see them on the screen. By adding 255, 0 motor encoder counts will appear in the middle of the plot.

Use the sample code to test your encoder, dsPIC circuit and plotting in the utility.

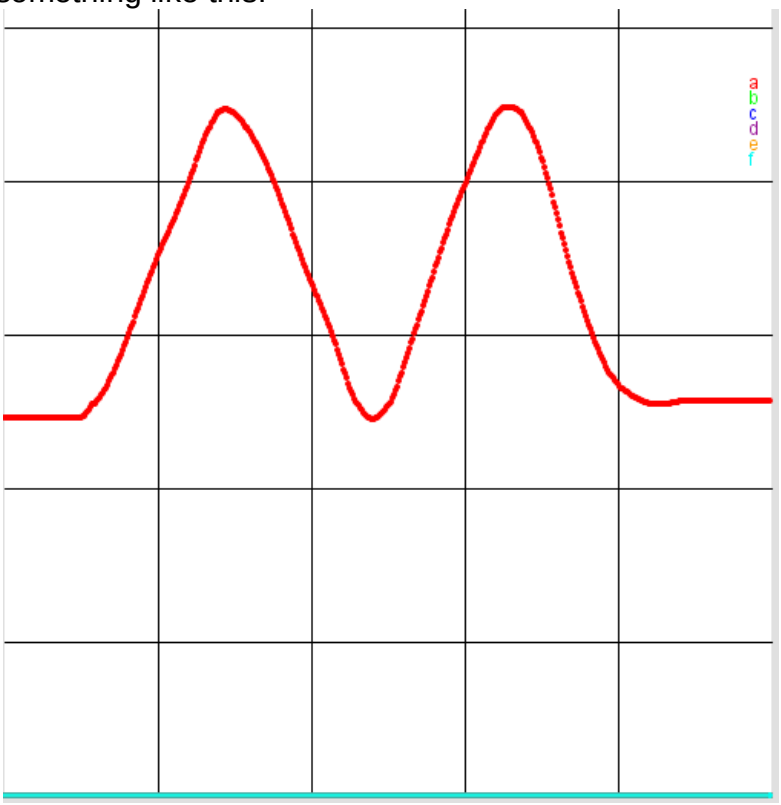
The encoder on our motors has 99 lines per revolution. The dsPIC uses x4 decoding.

Set up a 100Hz timer based interrupt. In every interrupt, read the encoder and send the angle of the motor shaft to the computer, in degrees, where the initial angle of the motor when you turn on the NU32 is 0 degrees.

1. What equation converts the encoder counts to the angle?

Use the plot capability in the NU32_utility to view the angle of the motor. Have 0 degrees of motor angle located in the middle of the screen (a value of 255).

You should see something like this:

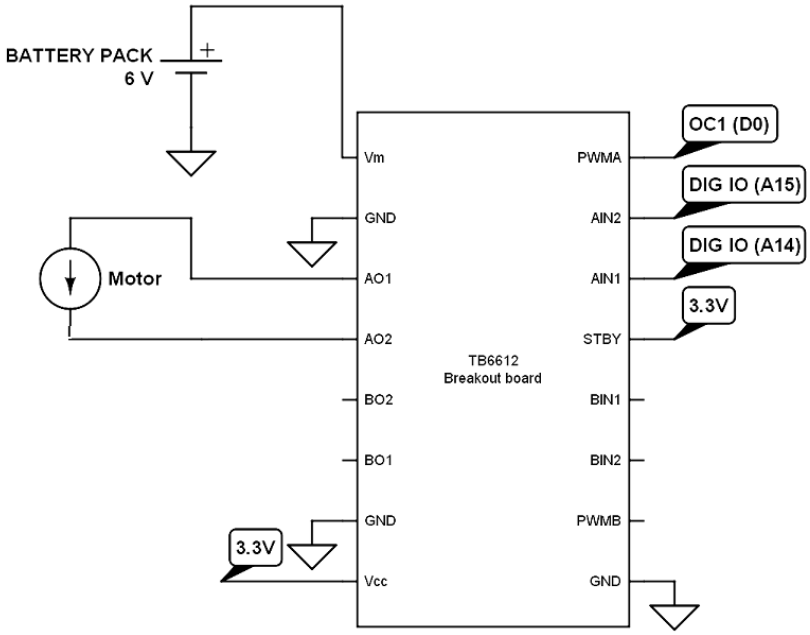


Take a screenshot of the plot as you rotate the motor shaft back and forth 180 degrees two times. Name the image Part1_FirstnameLastname.png

Save your code as Part1_FirstnameLastname.c.

Part 2: Power the motor

Use the following schematic to wire your motor and NU32 to the TB6612 H-bridge. Use your AA battery pack to power the H-bridge at Vm. Put the inertia bar on the motor without the washers.



In your code from Part 1, initialize PWM on pin D0 at 20kHz, and define pins A14 and A15 as digital outputs to control direction. Use the following logic table to control the direction and speed of the motor.

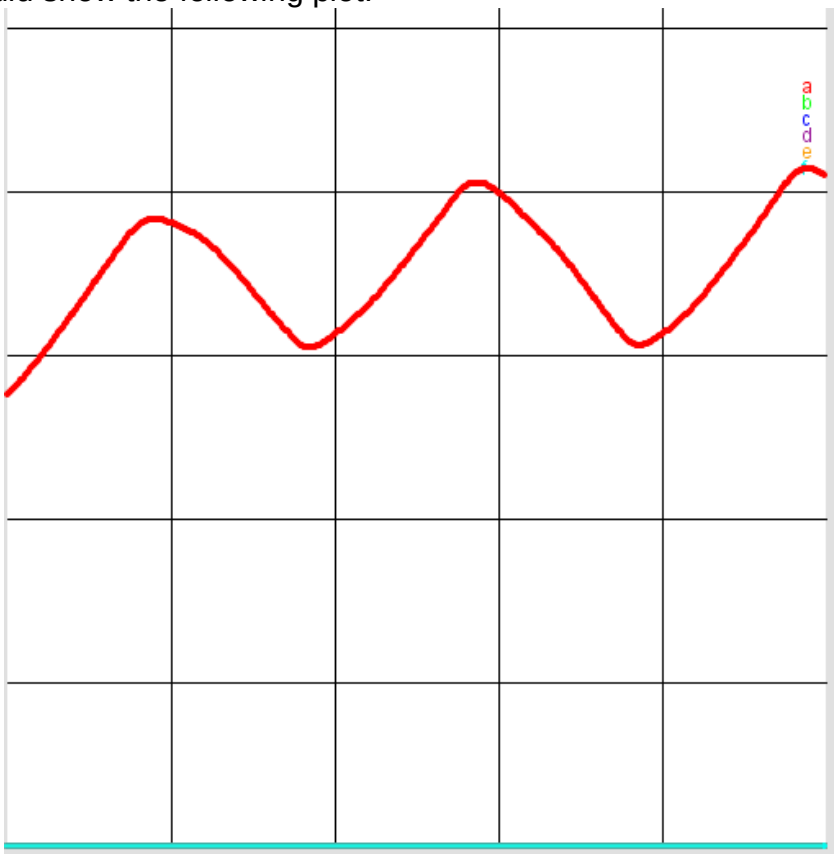
Input				Output		
IN1	IN2	PWM	STBY	OUT1	OUT2	Mode
H	H	H/L	H	L	L	Short brake
L	H	H	H	L	H	CCW
		L	H	L	L	Short brake
H	L	H	H	H	L	CW
		L	H	L	L	Short brake
L	L	H	H	OFF (High impedance)		Stop
H/L	H/L	H/L	L	OFF (High impedance)		Standby

Write a function called SetMotor(Speed,Direction) that takes Speed (0 to max duty cycle) and Direction (0 or 1) as inputs, and sets the duty cycle and direction of the motor. Don't let the duty cycle be outside the range of 0 to the maximum duty cycle. If the Direction is 0, have the motor rotate CW, and CCW if the Direction is 1.

2. What is your SetMotor() function?

In the 100Hz interrupt, set the speed of the motor to 20% of the maximum and alternate the direction every second.

The utility should show the following plot:



Make a screenshot of the angle of the motor plotted in the utility. Name the image Part2_FirstnameLastname.png

Save the code as Part2_FirstnameLastname.c.

Part 3: Simple position control

In this part, the user will enter a desired angle for the motor to move too. The motor will alternate from the positive value of the desired angle, to the negative value of that angle, for 2 seconds each. The desired angle and actual angle are plotted in the utility.

Create three global variables called UserDesiredPosition, DesiredPosition, and Kp, and initialize them to 0. Initialize the motor to Speed=0. Ask the user to type in a value for UserDesiredPosition, from 0 to 360, and Kp, from 0 to 50. Start the 100Hz interrupt. **If anything goes wrong from here, hit the reset button on the NU32, or turn off the battery pack.** In the infinite loop, ask the user to enter a new desired angle and value for Kp, so you can change the values without touching the NU32.

In the 100Hz interrupt, set the DesiredPosition to UserDesiredPosition for two seconds, and –UserDesiredPosition for the next 2 seconds, and repeat.

3. For how many interrupts is DesiredPosition positive?

In the 100Hz interrupt, read the motor angle, and calculate the Error between the motor angle and the DesiredPosition. Multiply the Error by the gain Kp, and set it to a variable called Control. Set the speed and direction of the motor using Control.

4. What is your equation for Control?

Run your code, and start with small user desired angles and small values for Kp. If your motor spins out of control, you have positive feedback – you can use any of these fixes 1. switching the motor leads at the H-bridge, 2. switching the sign of your error calculation, or 3. switch the A and B wires at the dsPIC.

Tune Kp to get a fast response time, but less than 50% overshoot when the user desired angle is 90 degrees.

You should get a response like the following image.

5. What value of Kp worked best?

Take a screenshot of the best result. Name the image Part3_FirstnameLastname.png

Save the code as Part3_FirstnameLastname.c.

Part 4: Derivative control

In this part, penalize overshoot in the motor angle by adding a derivate control term.

Let the user enter a third input, Kd, between 0 and 10000, and set the value to another global variable, Kd.

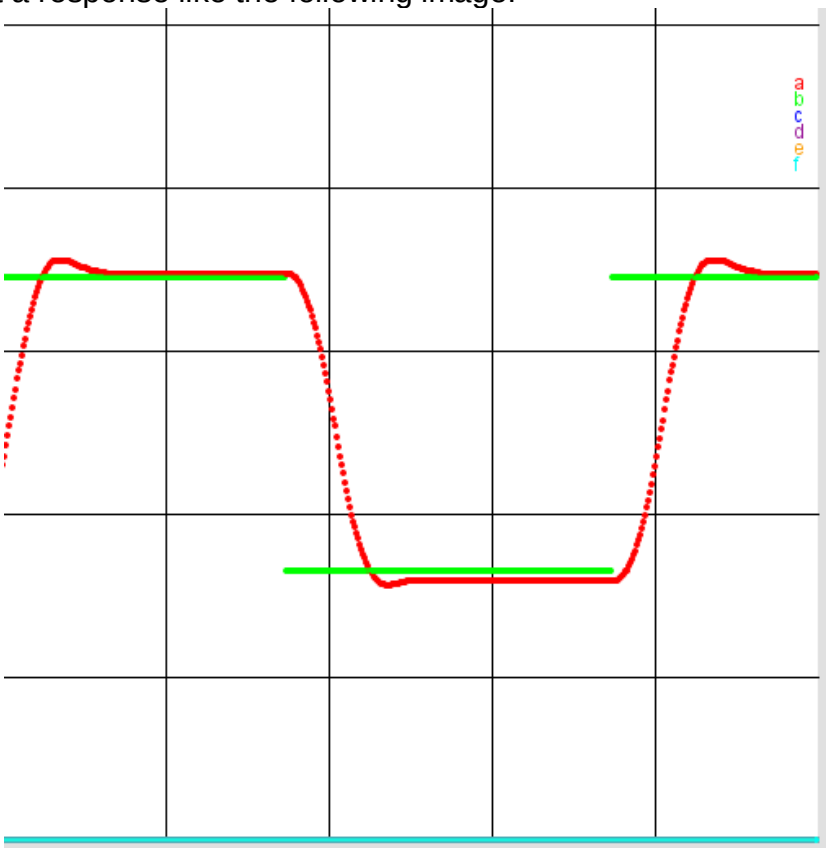
For the derivative term, you need to remember the previous Error, so make another global variable called PreviousError, initially 0, and in the 100Hz interrupt set the PreviousError to Error after calculating Error, so that you remember it for the next interrupt.

In the 100Hz interrupt, calculate the change in error, and multiply it by Kd, and add that to Control.

6. What is your equation for Control now?

Play with different user desired angles, Kp, and Kd inputs. Try to get a better rise time than you had in Part 3, with less overshoot, when the desired user angle is 90 degrees.

You should get a response like the following image.



7. What value of Kp and Kd worked best?

Take a screenshot of the best result. Name the image Part4_FirstnameLastname.png

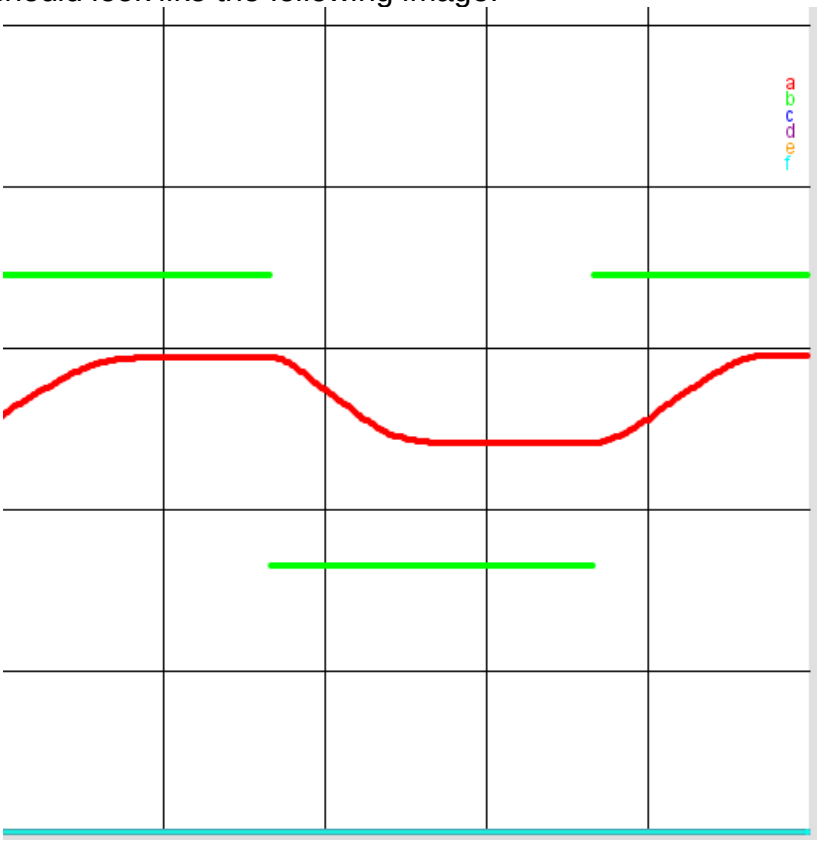
Save the code as Part4_FirstnameLastname.c.

Part 5: Integral control

Your response so far has steady state error – a gap between the actual motor angle and the desired motor angle. You can reduce the steady state error by increasing Kp, but that may make your controller unstable.

Emphasize the steady state error in your controller by setting Kp too small, and Kd too small, and observe that the motor never gets to the desired angle. We can fix that using an integral term.

The response should look like the following image.



Take a screenshot of the response of your controller with big steady state error. Name the image Part5a_FirstnameLastname.png

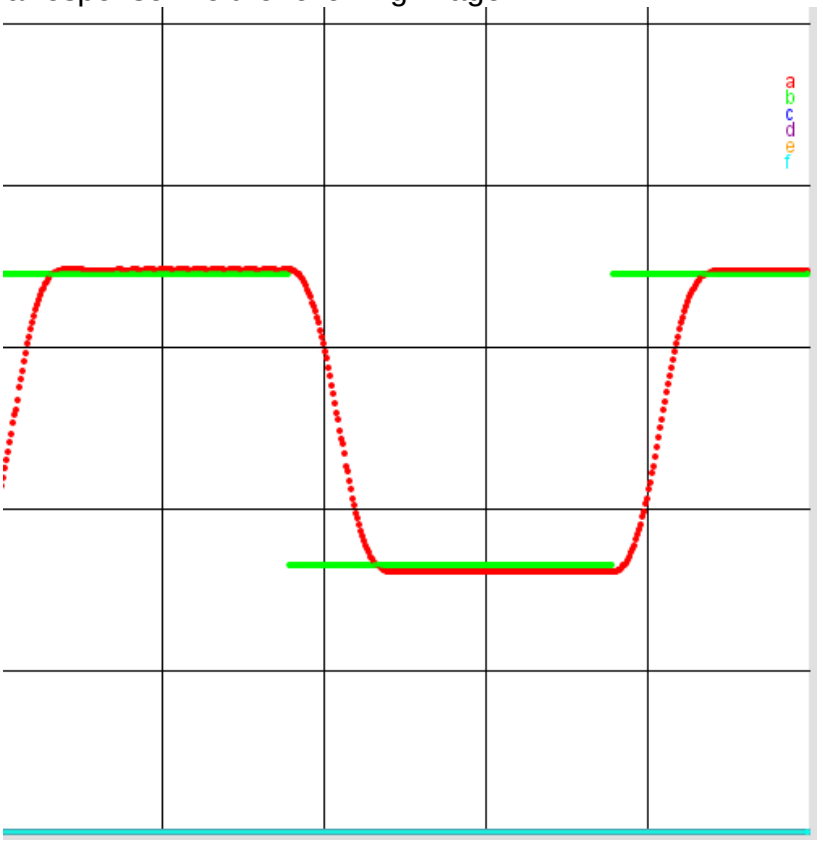
Let the user enter a fourth term, Ki, between 0 and 10000, and set it to a new global variable, Ki.

The integral term uses the integral of the Error, so you need to remember the sum off all of the Errors. Make another global variable, called SumError, initially 0. In the 100Hz interrupt, after calculating Error, add Error to SumError. Don't let SumError get bigger than 36000 or less than -36000. Multiply SumError by Ki, and divide it by 10000 so it doesn't add up to fast, and add the result to Control.

8. What is your equation for Control now?

You have now created a full PID controller! Tune Kp, Kd and Ki for a good response to a 90 degree desired user input. Ki can make your controller unstable, so be careful.

You should get a response like the following image.



Take a screenshot of the result. Name the image Part5c_FirstnameLastname.png

9. What Kp, Kd and Ki did you use?

Save the code as Part5_FirstnameLastname.c.

Add a washer to each end of the inertia bar. Retune Kp, Kd and Ki.

Take a screenshot of the result. Name the image Part5d_FirstnameLastname.png

10. What Kp, Kd and Ki did you use?

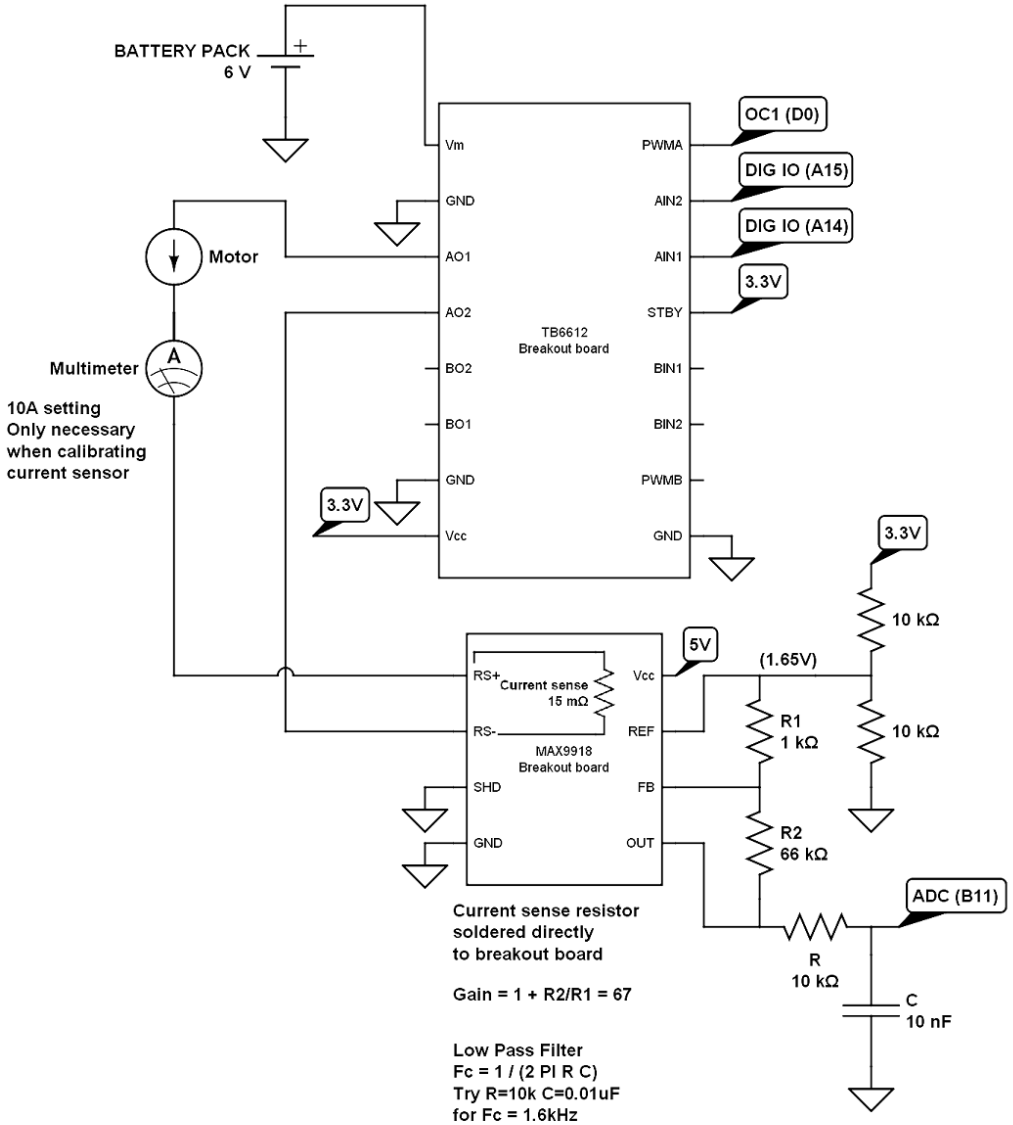
11. List 2 examples of systems where this PID controller would work well, and 2 examples of systems where it would not.

Demo the full PID controller, without the washers, in class on Thursday 3/14.

Part 6: Motor current

Our system contains a motor (with known parameters) and an inertia bar. We also know the trajectory the motor should follow. With this information, we could calculate the torque needed to create the angular acceleration necessary to follow the trajectory, without any feedback from the encoder. But we still need to be able to set the torque of the motor, and for that we need to be able to control the motor current.

Add the current sensor to the circuit using the following schematic. For now, include the multimeter in current mode.



Comment out the 100Hz interrupt and the code that turns the 100Hz interrupt on, we will come back to it later.

Write a function called ReadCurrent() that reads the analog voltage at pin B0 and returns the value as a number from 0 to 1023.

12. What is in the function ReadCurrent()?

You need to calibrate the current sensor to know what the analog voltage represents. Temporarily replace the motor with a small resistor, like 33Ω, to get a clean signal.

Current can change must faster than motor position, so create a timer based interrupt at 1000Hz. In the 1000Hz interrupt, read the current sensor, and send every 10th read to the utility.

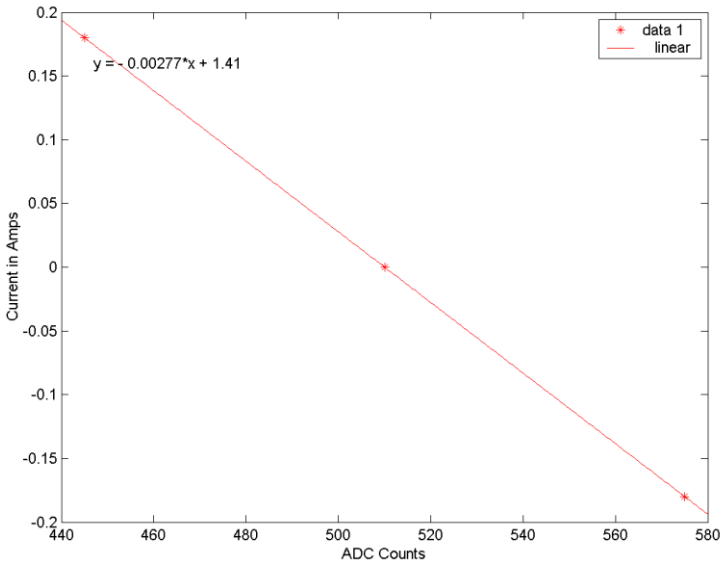
Set the Speed to maximum, and the direction to CW. Write down the average value you see in the utility. Write down the current reading for your multimeter.

Set the Speed to maximum, and the direction to CCW. Write down the average value you see in the utility. Write down the current reading for your multimeter.

Set the Speed to 0, and the direction to CW. Write down the average value you see in the utility. Write down the current reading for your multimeter.

Create a calibration plot for the current sensor in MATLAB or Excel, showing the slope with units A/count.

You should get something like the following plot.



Turn in your calibration plot. Name the image Part6_FirstnameLastname.png

13. What is the slope? What equation will turn the analog integer into current in A?

Edit ReadCurrent() so that it returns the current as a float in units of A.

14. What is the ReadCurrent() function now?

Save the code as Part6_FirstnameLastname.c.

Part 7: Current control

A professional motor controller controls current, and thus torque output, of the motor. In this part, create a PI controller for the motor current.

1000Hz is too fast to send data back, so we won't be able to see the desired and actual current it in real time. Instead, run the interrupt for 1 second, saving the current data to an array with 1000 elements. After the interrupt has run 1000 times, disable the interrupt, and send all 1000 data points back at once.

Make a global array of desired current with 1000 elements, called DesiredCurrent. Initialize the array with 100 values at 0.1A the next 100 elements at -0.1A, and repeat for all the elements in the array.

15. What is the loop that populates this array?

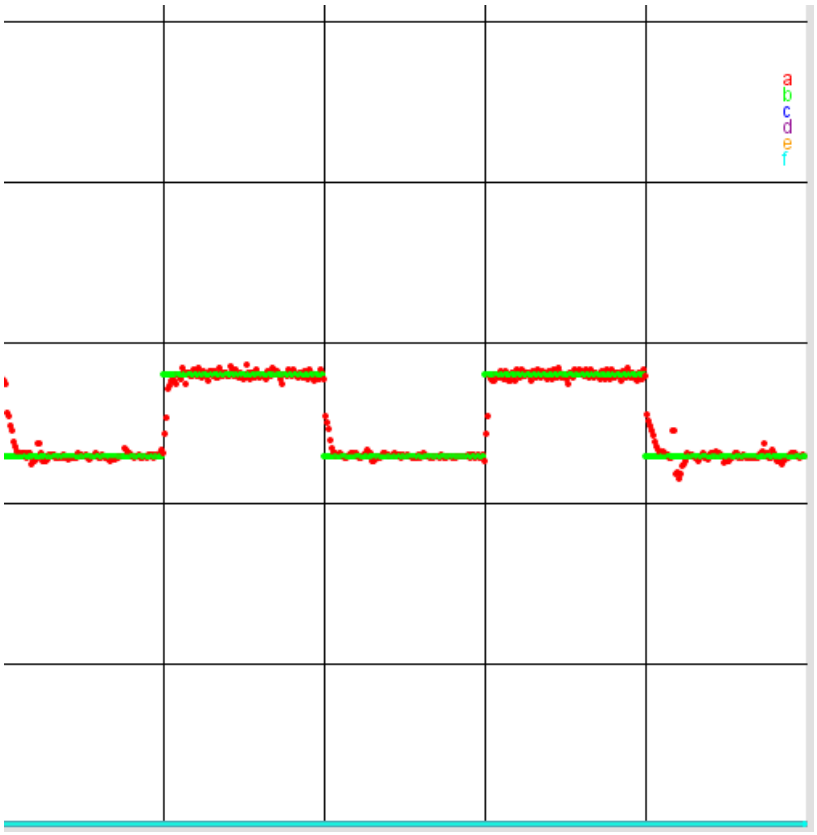
In the 1000Hz loop, read the current, calculate the CurrentError from the DesiredCurrent, calculate SumCurrentError, and create a CurrentControl using two constants, KpCurrent and KiCurrent. You can hardcode in the KpCurrent and KiCurrent values, or ask the user for them.

Remove the multimeter and 33Ω resistor, and put the motor back in place. The motor will give much noisier current readings than the resistor, but this is expected.

Run the interrupt 1000 times. On the 1000th time, disable the interrupt, set the Speed to 0, and plot the desired current and actual current data in the utility. Have 0A be in the middle of the plot, 1A at the top, and -1A at the bottom.

Tune the gains KpCurrent and KiCurrent to get a good match to the desired current.

You should get a plot like the following image.



16. What value of KpCurrent and KiCurrent worked best?

Take a screenshot of the best result. Name the image Part7_FirstnameLastname.png

Save the code as Part7_FirstnameLastname.c.

Part 8: Feed forward control

The angle trajectory you want the motor to follow is saved in DesiredTrajectory.h as float DesiredTrajectory[500]. This trajectory was generated with constant accelerations, saved as float DesiredAccelerations[500]. If you can set the torque of the motor to match those accelerations, you should be able to match the trajectory without position feedback.

Keep your 1000Hz interrupt with the best gains from Part 7. Remove the code that disables the interrupt after it runs 1000 times, remove the code that sends the current arrays to the utility, and remove the arrays for the desired current and actual current.

Add the 100Hz interrupt back in, but leave the position control commented out. Keep the code in the 100Hz interrupt that reads the position of the motor and sends it to the utility to plot it.

Convert the desired acceleration to a desired current. You could calculate the equation to do this, using the inertia of the system, but instead make a model, and find the parameters that work best.

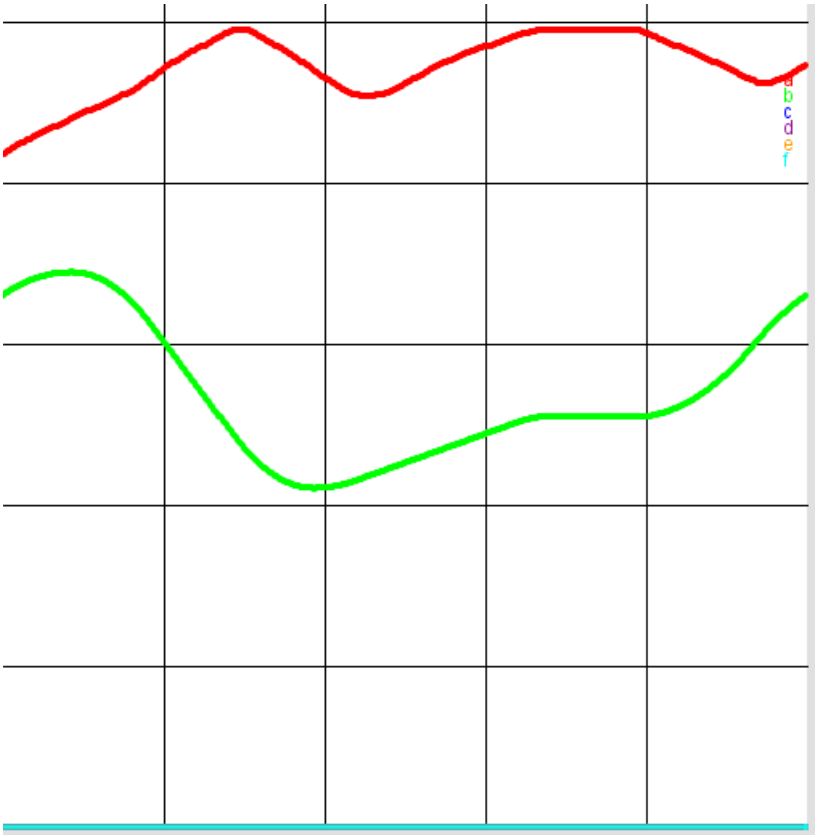
Create two global variables called Kt and Kc. Set the desired current to the desired acceleration times Kt. Add Kc if the desired acceleration is positive, or subtract Kc if the desired acceleration is negative. Kt will take acceleration and convert it to current, and Kc will cancel out the static torque necessary to overcome friction.

In the 100Hz interrupt, set the desired current, looping through the elements in the desired acceleration. Plot both the actual angle of the motor and the DesiredTrajectory in the utility.

In the 1000Hz interrupt, control the current to match the desired current, using the gains you found in Part 7.

Tune Kt and Kc until the actual angle starts to look like the DesiredTrajectory. Try to match the shape – it will be very hard to match the trajectory exactly.

You should get something like the following image.



17. What value of Kt and Kc worked best?

Take a screenshot of the best result. Name the image Part8_FirstnameLastname.png

Save the code as Part8_FirstnameLastname.c.

Part 9: Feed forward and Feedback motion control

The ultimate controller! Not really, but better than Part 5 alone.

In this part, combine the controller from Part 5 to the controller from Part 8.

In the 100Hz interrupt, read the motor angle, calculate Error, and set the desired current based on the PID Control and the feed forward controls from Part 8.

Tune the motion PID gains to track the desired trajectory.

Take a screenshot of the result. Name the image Part9_FirstnameLastname.png

18. What Kt, Kp, Kd and Ki did you use?

Save the code as Part9_FirstnameLastname.c.

Bring Part 9 to your final demo time, and be ready to demonstrate out a new DesiredTrajectory.h!

You can do any of the following for extra credit:

Extra credit 1:
Add the 16x2 LCD to your circuit. Display the motion PID gains on the LCD. Add three buttons to your circuit: 'Up' 'Down' and 'Next'.
Pressing 'Up' and 'Down' increase or decrease the gain, and pressing 'Next' allows you to change the next gain.
Demonstrate changing the gains in Part 5 or Part 9 using the buttons.
Save the code as EC1_FirstnameLastname.c

Extra credit 2:
(Code is unrelated to Part 1-9, create a new project, or integrate it into Part 1-9)
Add a potentiometer to your circuit, so that it outputs a voltage proportional to angle.
Read the voltage, and set the position of an RC servo to an angle corresponding to the potentiometer angle.
An RC servo takes a pulse every 20mS that is 0.5mS long for 0 degrees, and 2.5mS long for 180 degrees. You can do this with PWM, but this code is already posted on the Design Competition wiki page. Instead of the PWM method, make the pulses in a timer based interrupt, so that you can set the servo to 180 different angles.
Demonstrate the RC servo position matching the potentiometer angle.
Save the code as EC2_FirstnameLastname.c

Extra credit 3:
Read Chapter 15.3 on filtering, and add a 5th order MAF to the motion control derivative term in Part 5 or Part 9. Plot both the original, now unused, PreviousError and the filtered PreviousError in real time to show the effect of the filter.
Demo Part 5 or 9 with the MAF plot.
Take a screenshot of the result. Name the image EC3_FirstnameLastname.png
Save the code as EC3_FirstnameLastname.c

Extra credit 4:
(Code is unrelated to Part 1-9, create a new project, or integrate it into Part 1-9)
Get another H bridge, and wire up a stepper motor.
Allow the user to enter a desired final position for the stepper motor shaft, and how long it should take to get to that position.
What is the maximum speed of the stepper motor?
Save the code as EC4_FirstnameLastname.c