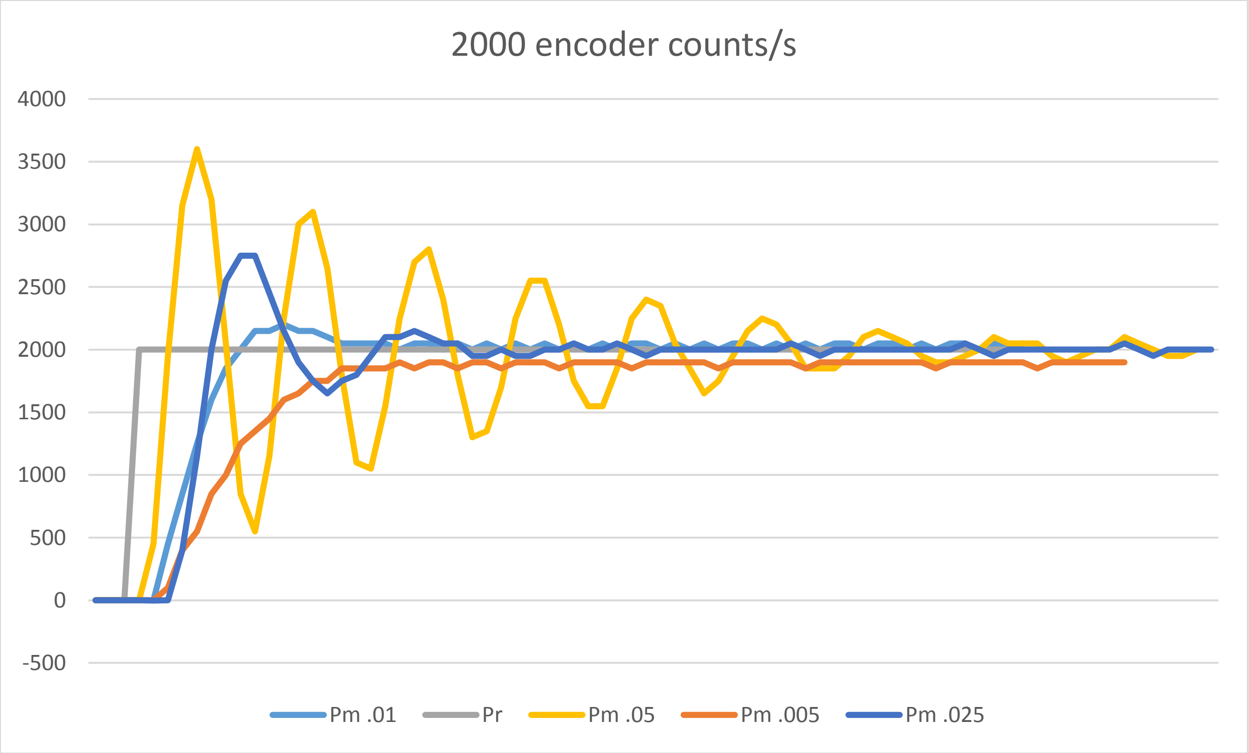
Lab 2 Motor

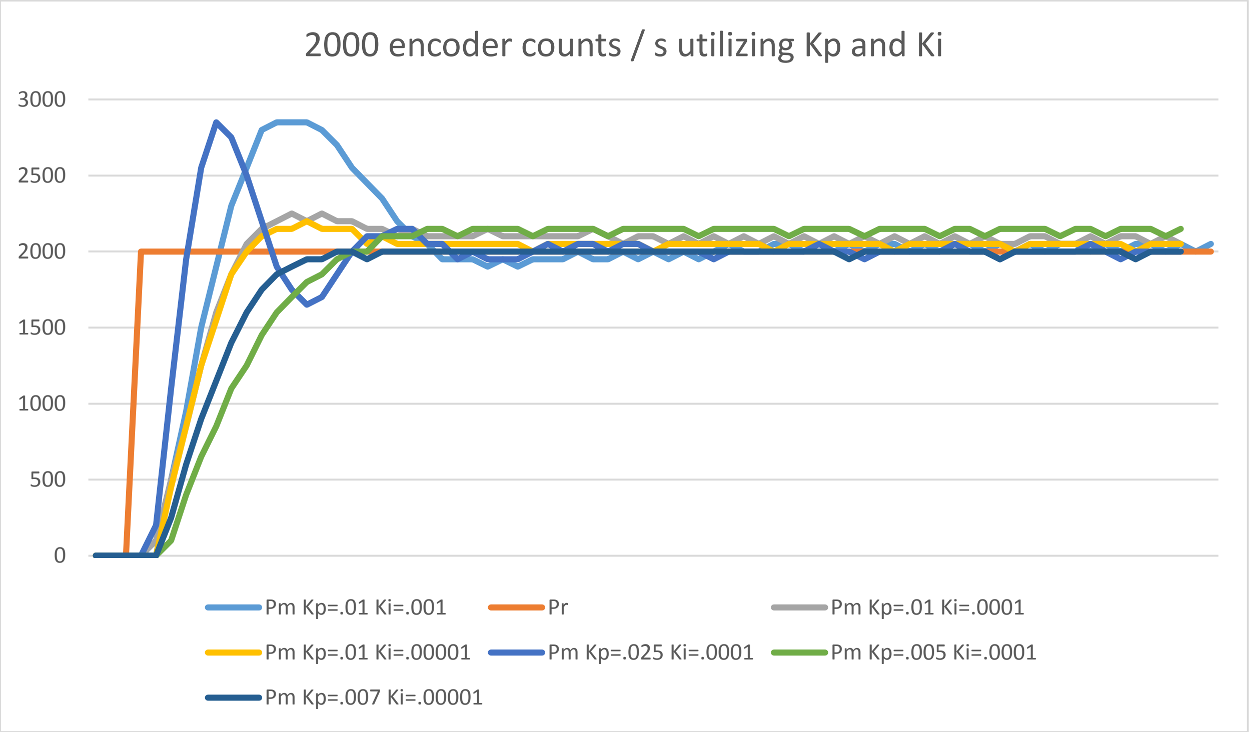
Question 1

1. Using only P it was necessary to get the TIMER0 time to a manageable amount. Also it was necessary to average the speed of the encoder reads over a set period of time. If this period of time was too large then the motor error would be too much. The calculation would use an averaged motor speed to get the current motor speed in encoder counts per second. In my code the value of the proportional gain was measured to be around .005 to .010. Somewhere in between there would get you a similar result. At slow speeds, tested around 200 encoder counts / s, this resulted in the speed being a bit wavy. Where it would show quite a bit of jitter, but not enough to result in a speed change with only P. At higher speeds like 2000 encoder counts per second this jitter was not as large and didn't result in a speed change. The chart below shows what the measured speed of the motor was generated based on the Kp values of .005, .01, .025, and .05. The lower the Kp value the harder it was for the motor to get to its position as the multiplier was not a large enough value to cause the torque of the motor to change. The higher the Kp value resulted in major shaking of the motor (technical term here) that actually caused the motor to go to a negative torque and after the error calmed down the motor tried hard to maintain 2000 counts / s but was only close and saw lots of jitter.



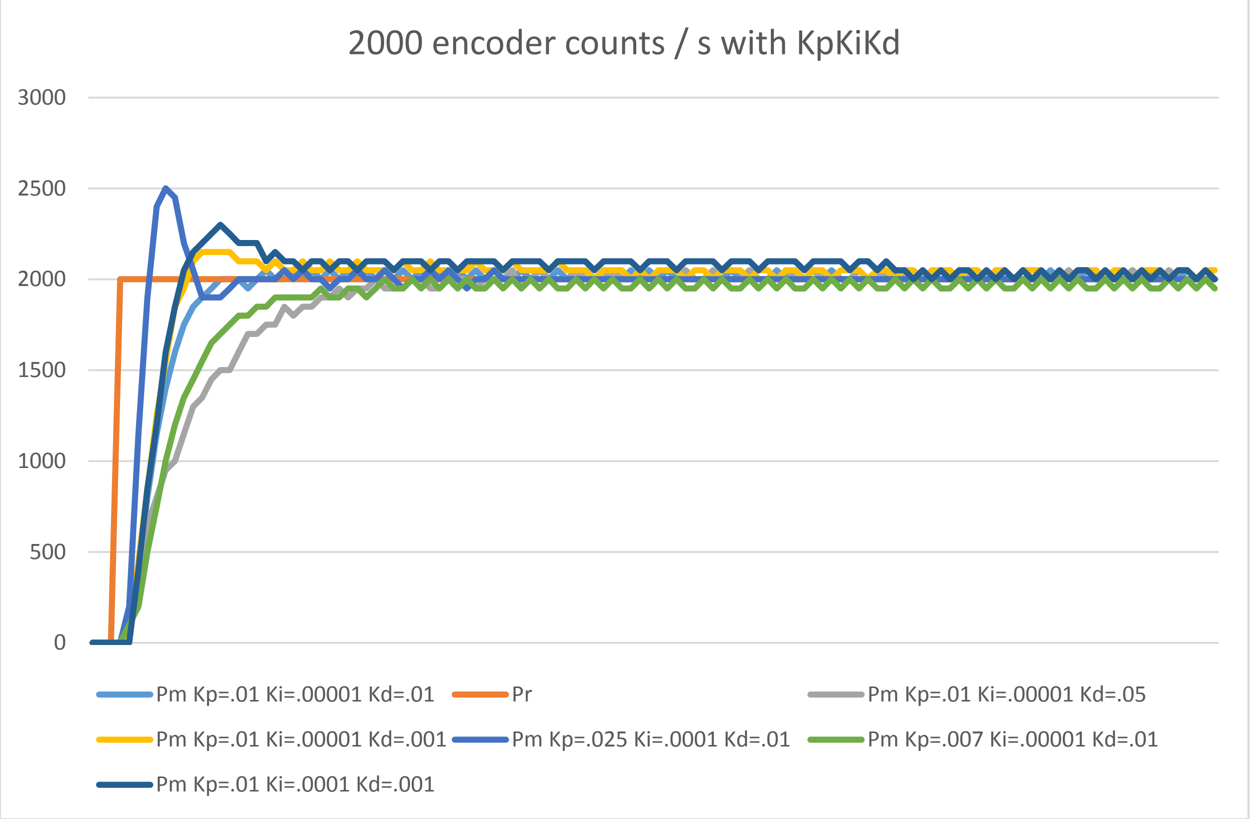
2. Incorporating I into the mix. In running these experiments… due to how I was calculating speed in encoders / s. I needed to make the Ki value really really, really small. What I found was that the smallest value of Ki being .00001 with a min/max of 100000 allowed me to get the best control. In this case a Kp value of .007 and a Ki value of .00001 gave me the best approach to the desired drive (2000) and didn't oscillate too much.. .The oscillation was observed in the physical presence of the device as it changed speed… Too much time had elapsed to capture all

of the data. The following chart shows the experiments ran and what they produced. All tests produced a curve that eventually got close to the desired speed. One thing of particular note was the Kp .005 and Ki .0001. This one took a long time to eventually get to the desired speed, but it did eventually get there (probably not obvious on the graph).



3. Add D into the mix. For this I noticed that the oscillations slowed down. My data tends to show oscillations because of how the current speed is calculated and measured. There are instances where I am getting more encoder counts within a window than other windows and when I average them out I am getting numbers that will jump and drop. The diagram below shows the

experiments that were performed. And the data it generated.



Question 2

The values that I selected for the Kp, Ki, Kd work best when the speed is calculated on a 100 Hz boundary. The PID controller was only updated every 50Hz and used the average speed from the previous two readings of the encoder counts per second. This appears to be okay… More work would need to be done to see what gains are necessary when doing things faster. When changing the PID controller to run at 100Hz and only use a single instance of the speed the target speed tended to oscillate more frequently. Changing the timer value to a 1Khz boundary caused the motor to jump around as the speed was updated very fast and caused the oscillation to be exasperated so the change in motor speed was more extreme, rather than gradual.

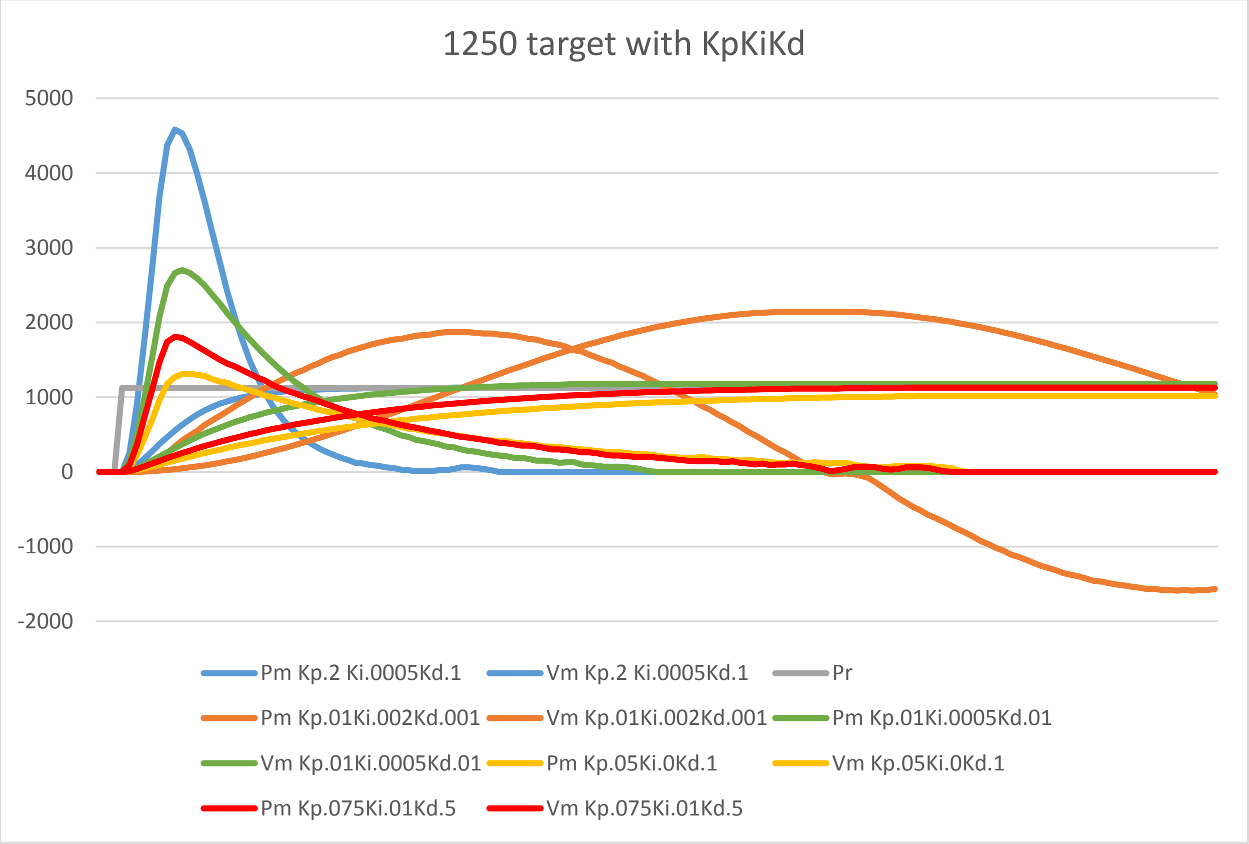
Decreasing the amount of times the PID can be ran resulted in a much smoother motor operation. The gains held up very well in this instance. The value I changed it to was every 10 Hz to attempt to update the PID if necessary instead of every 50 Hz.

Had issues on this one because when setting the motor to full speed and then telling it to turn off… the gains I had set made a HUGE difference. With my gains of p=.01 i=.0001 d=.001, I ended up resetting the micro controller when attempting to get to 0. By setting all of the values to their smallest possible values I was able to turn down the motor very slowly and avoid the sudden crash. I then set p=.001, i=.00001, d=.001 and it is behaving more smoothly. I also set the PID controller to run every 10Hz. Also calculating the last 10 speed entries and averaging them out. This ultimately slows us down but makes things sound better as it ramps up.

Question 3.

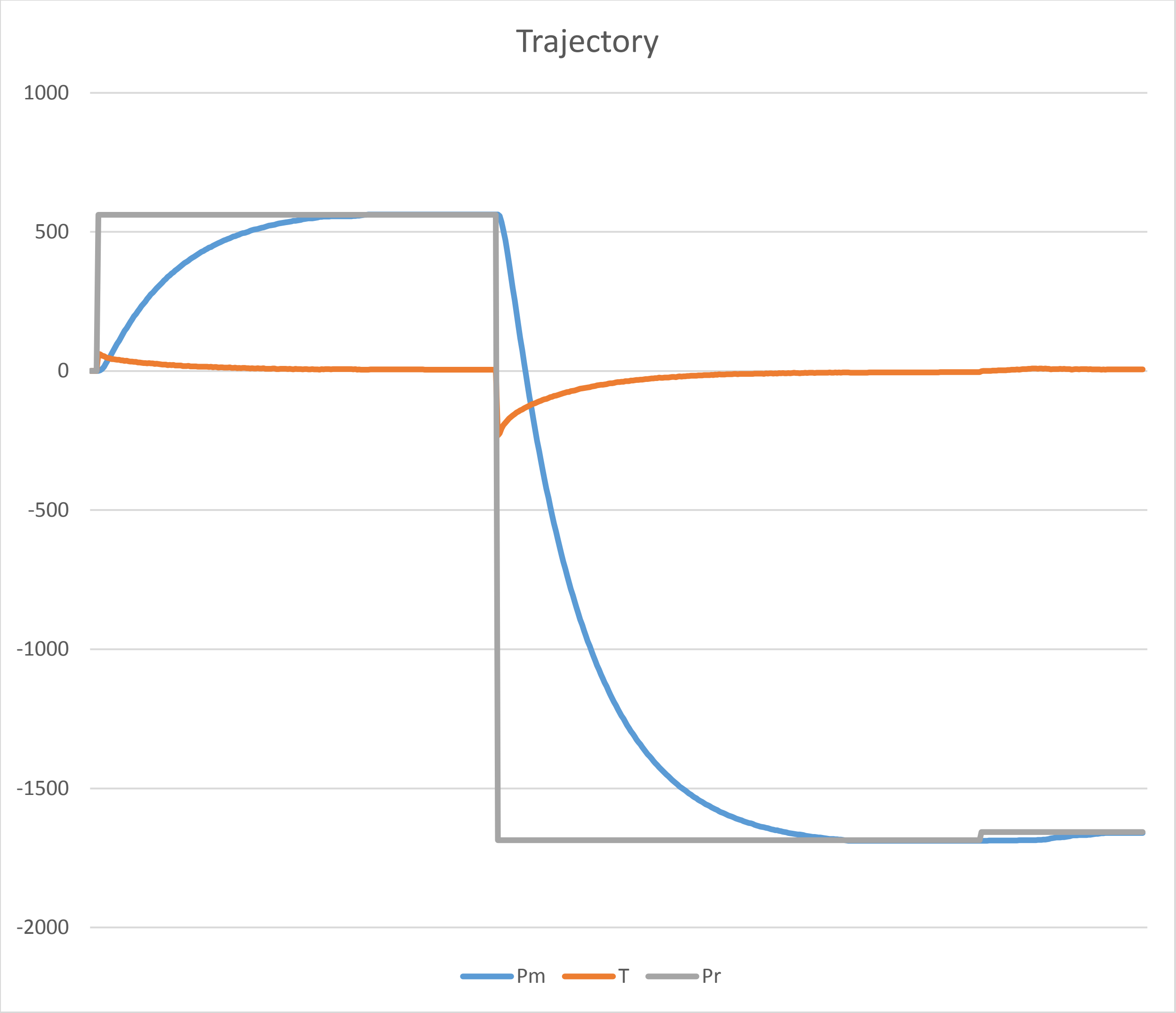
Started with very low gains, but they were higher than what was used for the speed controller. When removing I and having a small d I noticed that the PID loop was producing a value < 6-8. Any value < 6 does not move the motor and will then get the position close to the final value, but not exactly the final value. In most instances when we get close to the target we are still oscillating a bit because of the accrual of errors when counting Ki. Without Ki included we often approach the final target very slowly and due to not counting errors we never quite make it. In the example on the chart, color orange, we had low values for Kp, Ki, Kd and this resulted in a massive oscillation and overshoot.

The ideal values I arrived at resulted in the red lines below. I observed that if I speed up too fast and then slow down too fast the board will reset itself, or completely turn itself off. These values allowed the board to stay on, specifically when attempting to achieve very large targets like 720 degrees or higher.

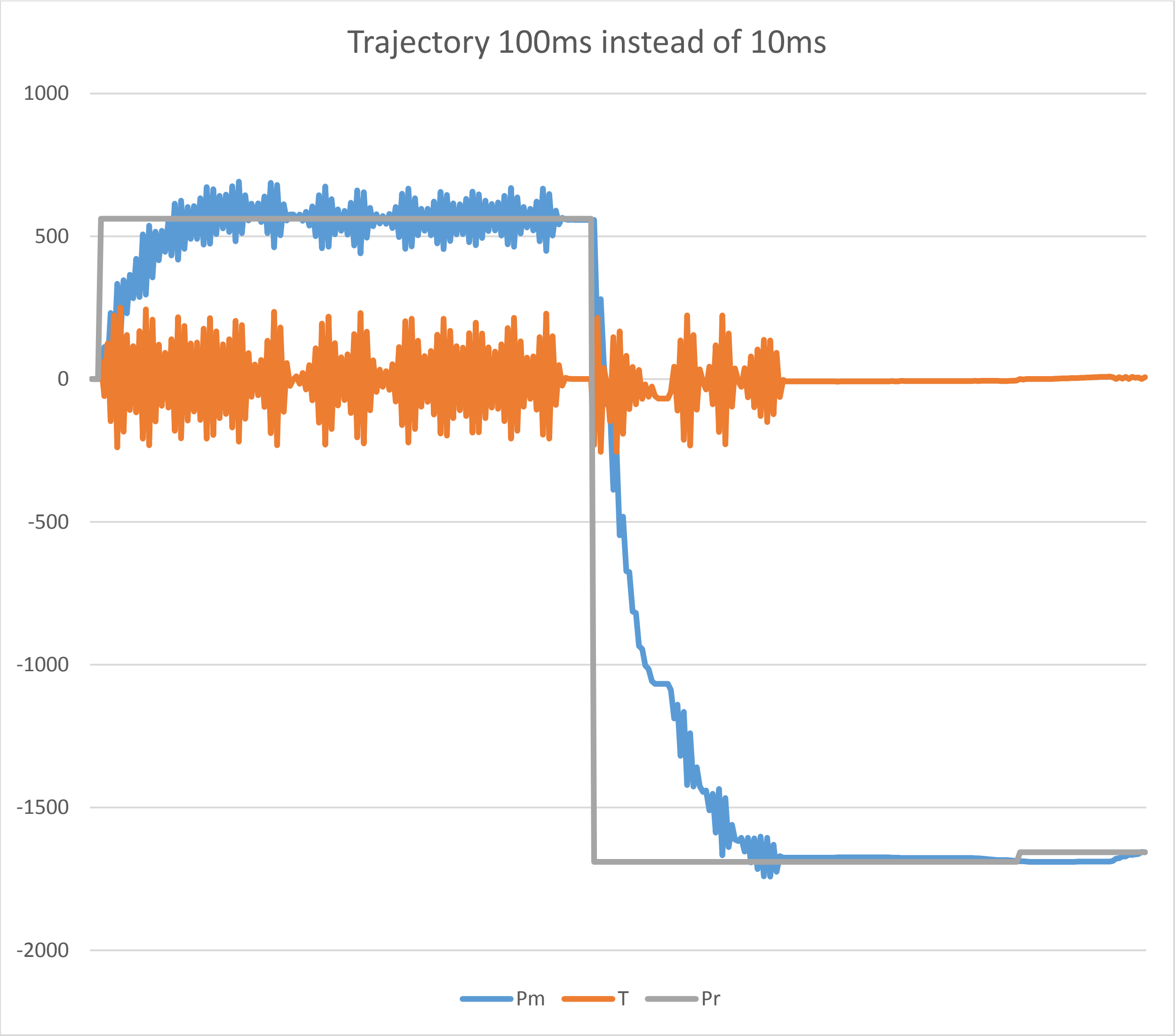


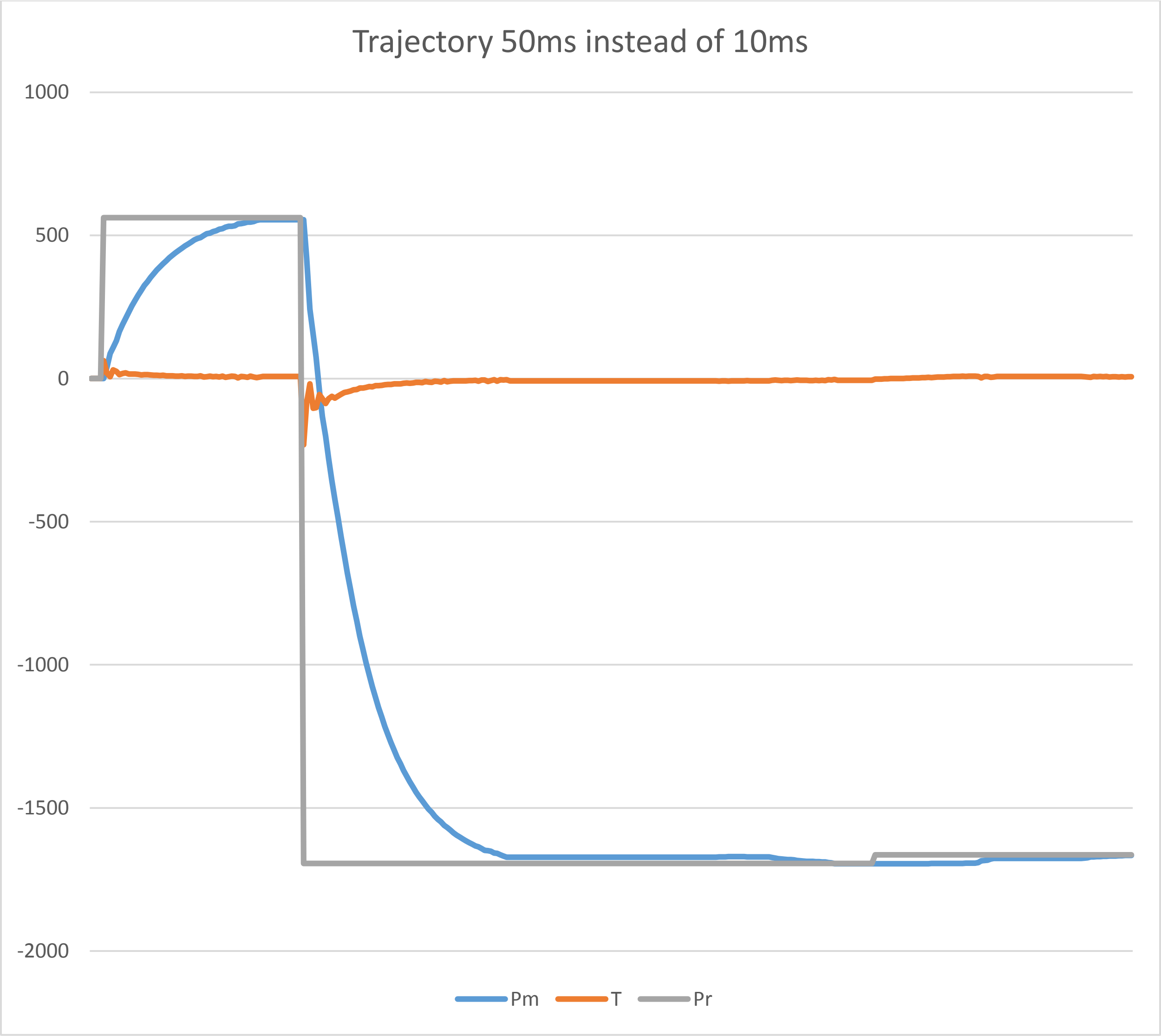
Question 4.

At this point in the assignment I was resetting the boards power very frequently. I realized the motor a and motor b were different than everyone else, so I corrected. I ended up with a Kp of .1, Kd of 1, and Ki of .01300 to get the board to achieve a value close to by target rotational value. My problem with resetting the board had to do with how fast the motor was spinning down and then going in the reverse direction.



Question 5.


Trajectory 100ms instead of 10ms
1000
500
0
-500
-1000
-1500
-2000
Pm
T
Pr



Noticed that the longer the timing for the PID routine the more jerky the motor got. For the 100ms pid loop instead of the default 10ms pid loop my board would routinely turn off. I found that if I held onto the motor myself to slow down the oscillation that it would work a bit better. The 50ms example didn't allow the overshoot so much so it was easier to make it to the targets. The 100ms, due to the frequency of how often we were entering the pid loop caused the motor to overshoot its target. This caused much oscillation.