PID Controller Lab
Andrew Eldred

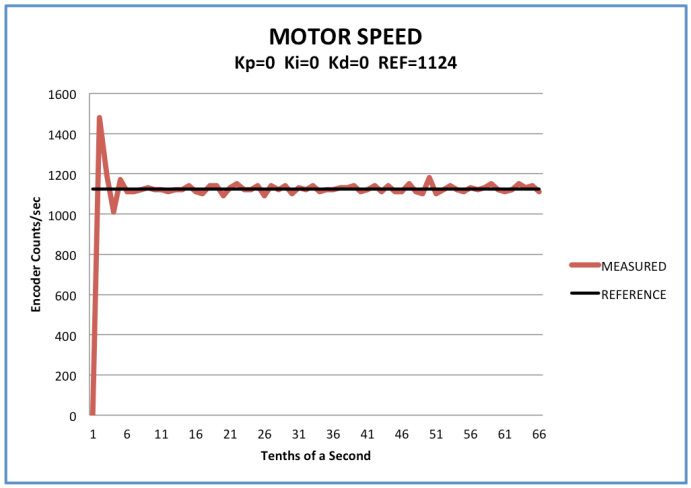
1. Speed Controller
   The first thing I noticed when adding in the proportional gain (P) was that it was difficult to determine at what point my speed was oscillating because it never arrived at the exact speed I set. When I continued to increase the proportional gain I found that there did come a point (around 2.9) where my speed would hover around the target and then suddenly dip way down and then way up again. I took this to mean that I had found the point at which it was oscillating.

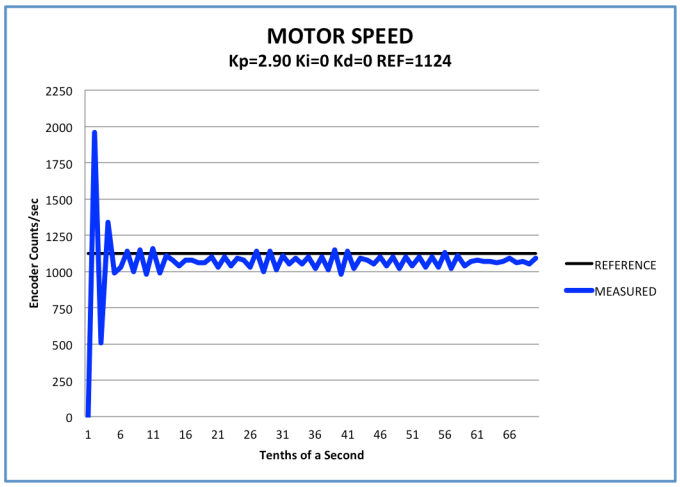
   The other observation I made when adding in P was the steady-state error that prevented the motor from ever really getting up to the desired speed. As I increased the integral gain (I) I saw that the values began to move closer to the desired speed, although with a lot of oscillation. I settled on an integral gain of 1.33 at which the average motor speed was close to the desired speed.

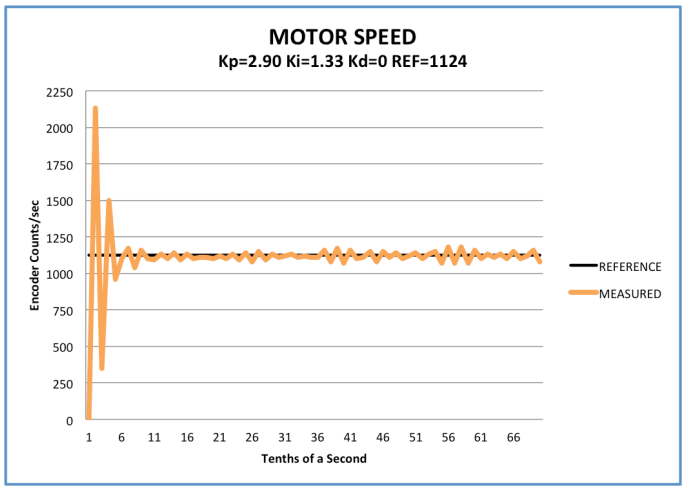
   When I first added in the differential gain I started with a value that was much too high compared to the P and I gains and the motor oscillated between positive and negative directions with each loop through the PID controller. On reflection, this made sense since the differential is a negative value that serves to dampen the oscillation and too much of it would over-dampen and result in extreme values. I adjusted the differential gain until the oscillations had mostly disappeared for the first three-tenths of a second. At this point increasing the D gain only seemed to make the results worse and I still had a huge spike at ramp up, so I reasoned that I needed to adjust P and I.

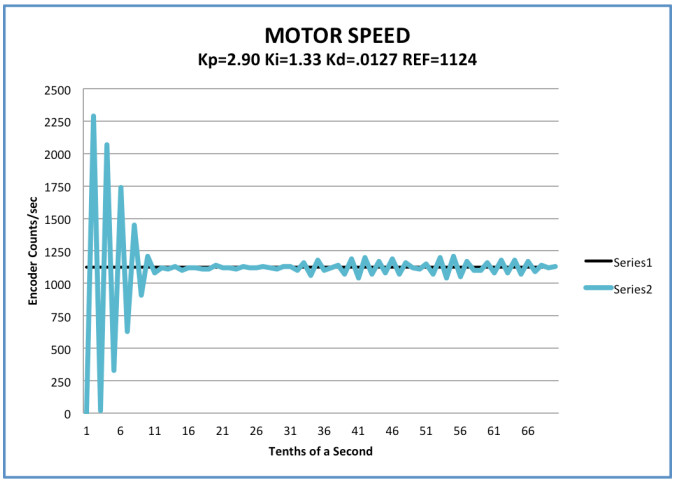
   I discovered that by decreasing P while also increasing D I was able to get a smoother start to my motor. By increasing the differential gain as I was decreasing the proportional gain I found that I didn't have to decrease P as much to get a smooth curve. The integral gain seemed to have a much more subtle effect on the results that helped to fine-tune the PD control.

   In the end I found values that created a decent ramp up of the motor speed without overshooting (too much) the desired value.









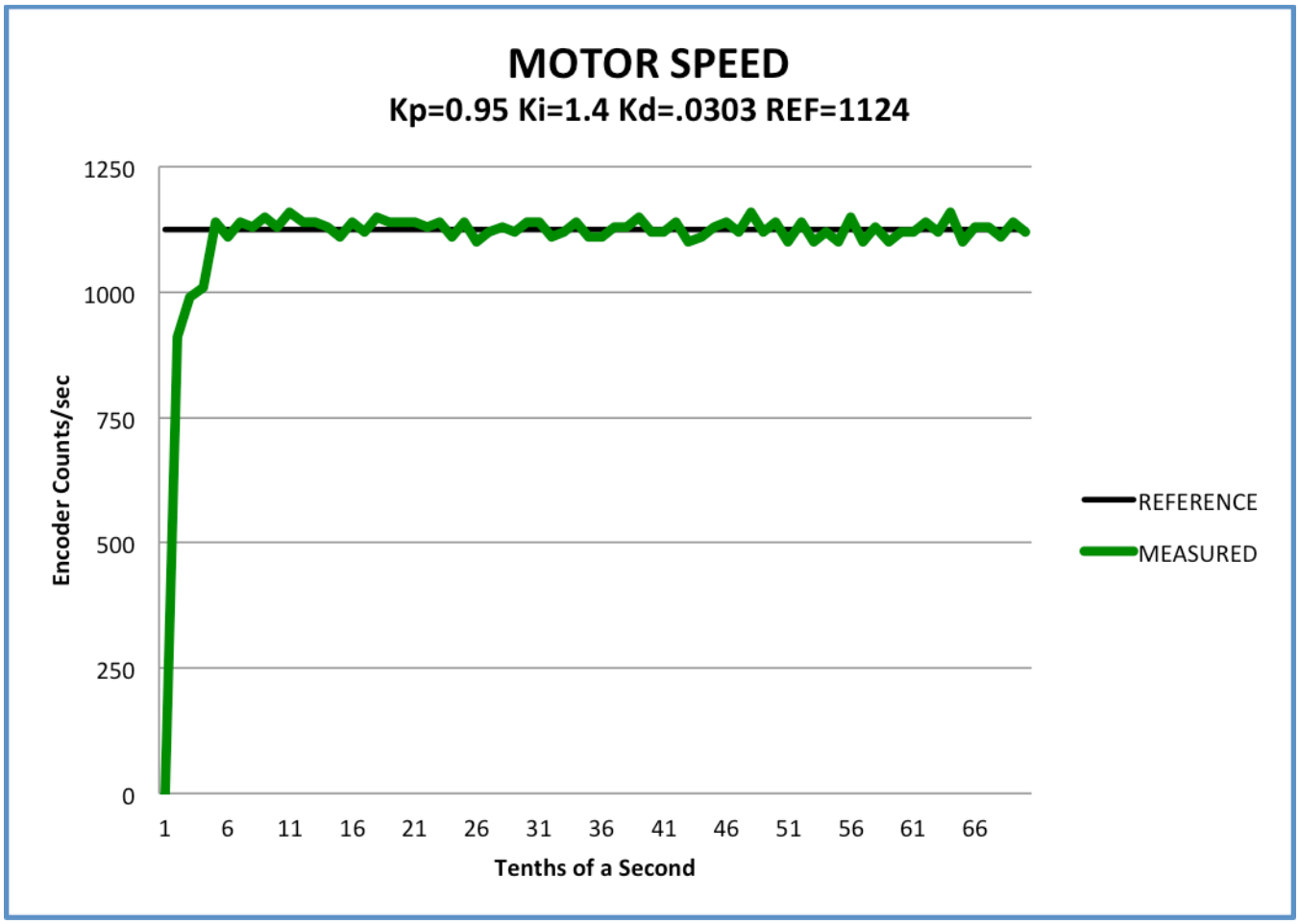


Figure 1 - Final PID values for speed control

2. At very low speeds (200 encoder counts/sec) it takes longer for the motor to reach the desired position and the calculated value is less than 1 for several timer cycles. Eventually it rises to the reference speed, but with a high oscillation around the desired speed. If I increase the speed slightly to 500 encoder counts/second, the graph looks much similar to the findings above with a slight overshoot that eventually levels out (mostly). At high speeds, there is a greater initial overshoot around the .5 second mark on the x-axis, but it then levels off to a similar, if not more stable control of the motor than at average speeds.

The overshoot at high speeds makes sense since the controller is inherently dealing with larger values. Likewise, at the lowest speed there is not enough differential gain to dampen the oscillations, the result of inherently smaller values being fed into the controller. If we were dealing with a very wide range of values, I wonder if there could be a need for different sets of gains that vary depending on a minimum or maximum value given to the controller.

When playing with the frequency of the speed calculation, I first wanted to see what would happen if the calculations were done at a frequency greater than that of the encoder interrupt (a period of .001 seconds). Not surprisingly, the torque values returned from the controller were all over the place and ranged from -28,933 to 31,489. Because the PCINT for the encoders takes priority over the calculation of speed in TIMER1, the speed calculation is not going to happen consistently within the period specified. This means that TIMER1 is only going to be triggered when PCINT hasn't taken priority and that will only happen a fraction of the time it is meant to. Thus, the change in encoder counts doesn't reliably correspond with the period for my TIMER1 software interrupt and the speed cannot be reliably calculated.

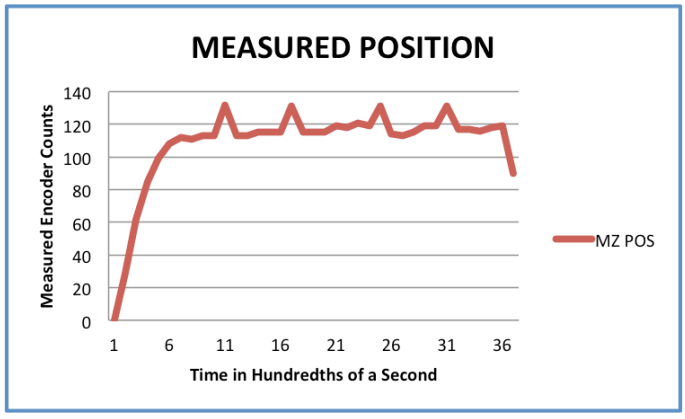
If I set the frequency to ¼ that of the PCINT interrupt, I get results that are more erratic due to the less frequent sampling of speed information. During this investigation, I tried a shorter period (.02 and .05 seconds) between calculations than what I used to initially tune my gains (.1 seconds). This created a smoother transition between each calculation (which I could hear in the motor more easily than see in the numbers), but I found it more difficult to tune the gains so that it didn't oscillate or initially overshoot my goal speed. I suspect that I would need to use smaller gains to smooth out the oscillations since the values between calculations are also smaller.

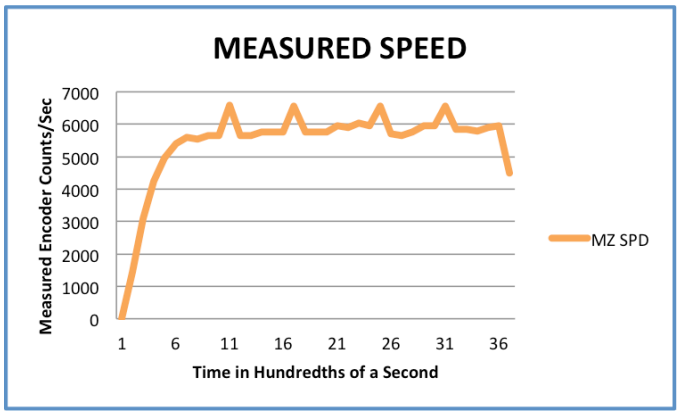
3. With low gains I observe consistent undershoot of the desired position. There is a minimum pwm frequency at which the motor will actually turn and if the value from the PID controller falls below that value then the position is never reached. I handled this minimum frequency by checking for a 0 count change in position when torque is being applied to the motor. If this check returns true for X sampling periods (I used 10) in a row, then the program assumes we've gone as far as we can.

   With higher gains the motor is more likely to overshoot the desired position. I found it difficult to tune the gains so that the motor consistently did not overshoot the position. After playing with the system and trying different combinations, I settled for an overshoot of less than 10 counts as acceptable for a stable system. In my tuning I found that the integral gain helped keep the motor running at a faster speed, although including the I gain required additional differential gain to stabilize the oscillations of the motor. After tuning the system my gains were defined as:

   P=1.15 I=0.19 D=0.0345

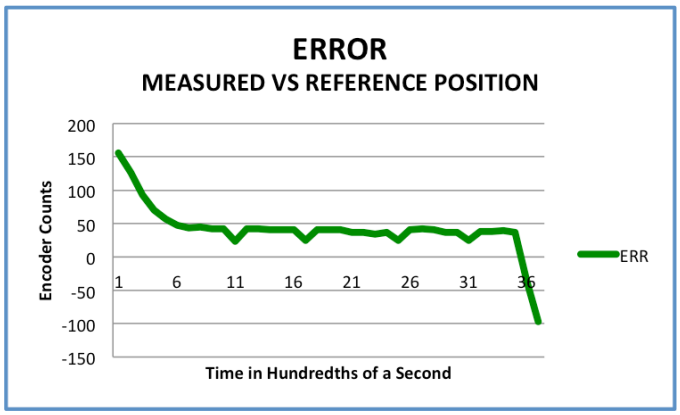
   These gains resulted in the following data for a position of 4000 counts.

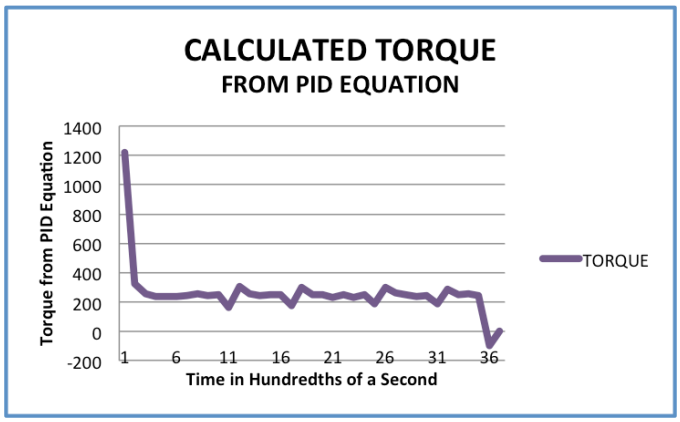




The two graphs above show the relationship between the measured position and the measured speed in relation to time. The shape of the graph shows that the initial ramp-up of the motor occurs within the first 6 hundredths of a second. Once the motor reaches a settled speed there are a few oscillations in the power sent to the motor, a result of my differential and integral gains not being tuned perfectly. When playing with the system I noticed that if I supplied too much differential gain compared to the other two, I would get a lower than expected torque. This makes sense because the differential gain serves to dampen the oscillations of the integral gain. I suspect this is a case of too much dampening as the supplied torque dips to a value that has been dampened too much. At first I thought that it must be a result of the integral gain being too high or not dampened enough, but then I noticed that the peaks exist without corresponding valleys. If the integral were the cause I would expect the value to also dip below the settled position and speed.

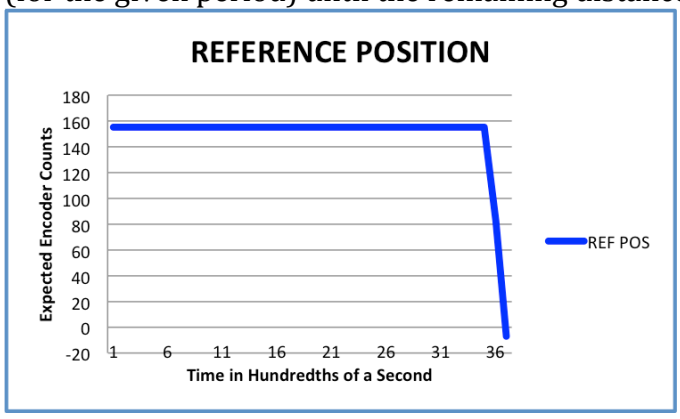
The inverse of these peaks is also reflected in the graph of the position error. At each dip in the calculated error we also see a dip in the torque that is fed to the motor. This is a result of the PID equation using the difference between the current and previous positions to estimate the torque needed to match the reference position.



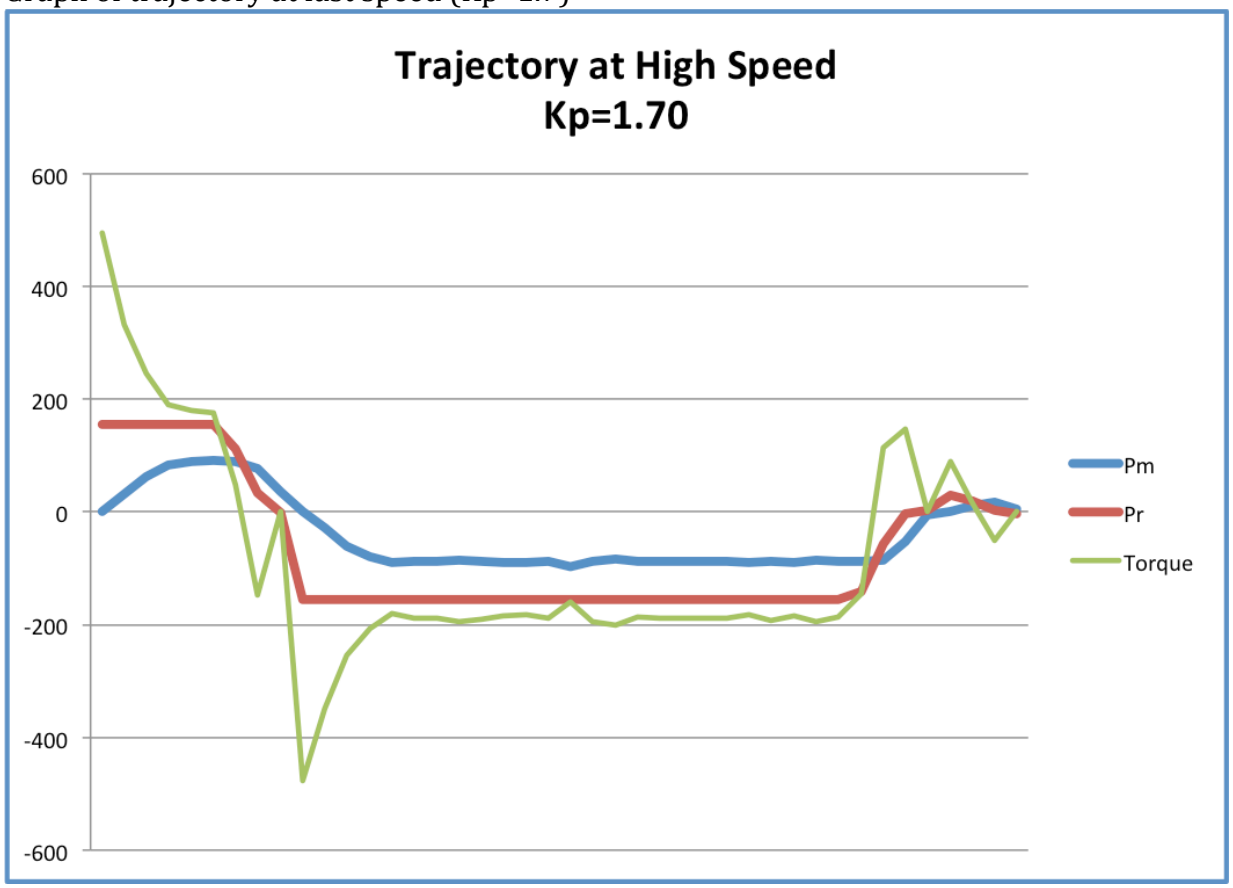


At the very bottom of the torque graph, there is a slight dip below 0. This is the value returned from the PID equation when the motor overshoots the desired position. In my implementation, when a torque is opposite in sign from the remaining distance to travel, a 0 value is sent to the motor indicating that we have gotten as close to the position as we are going to.

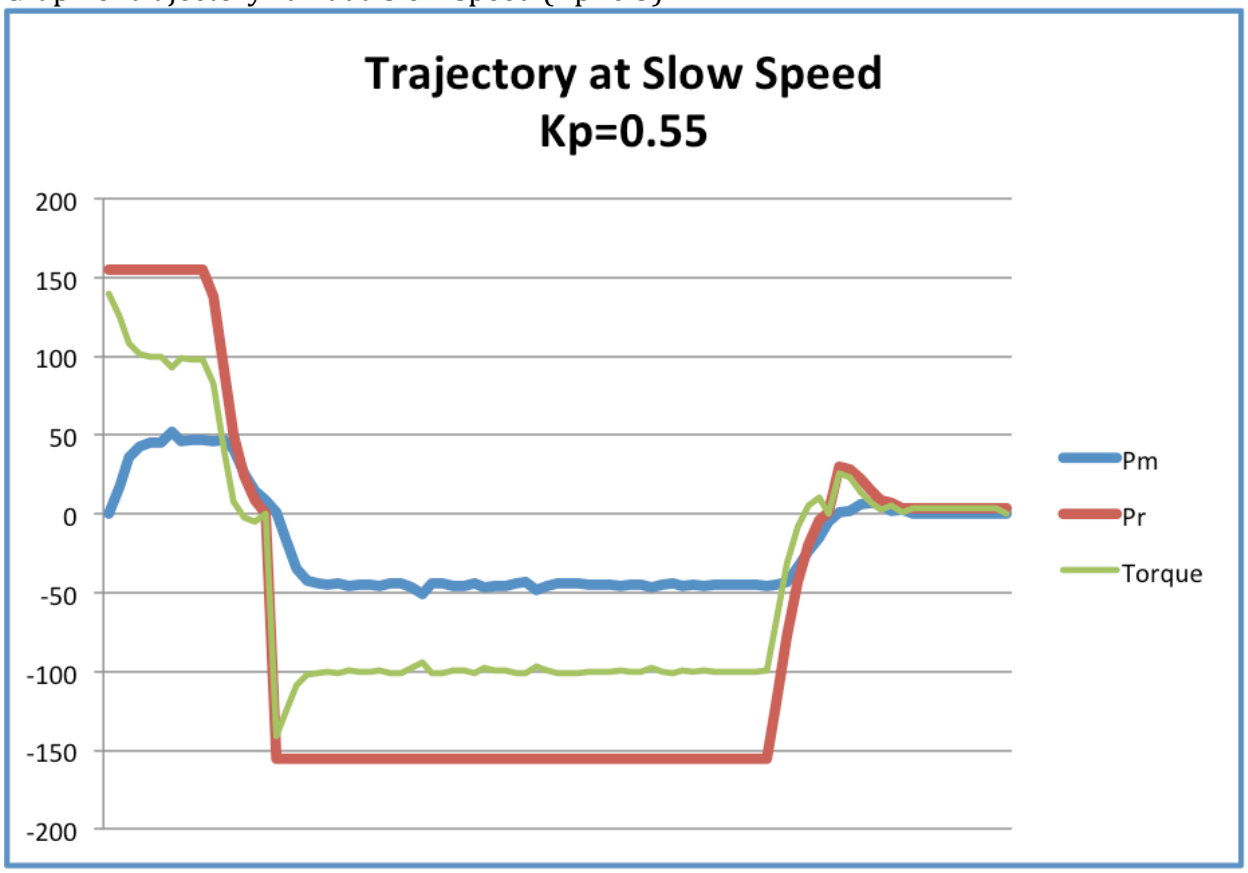
The graph of the position shows that the reference position remains at the maximum possible value (for the given period) until the remaining distance drops below that max.



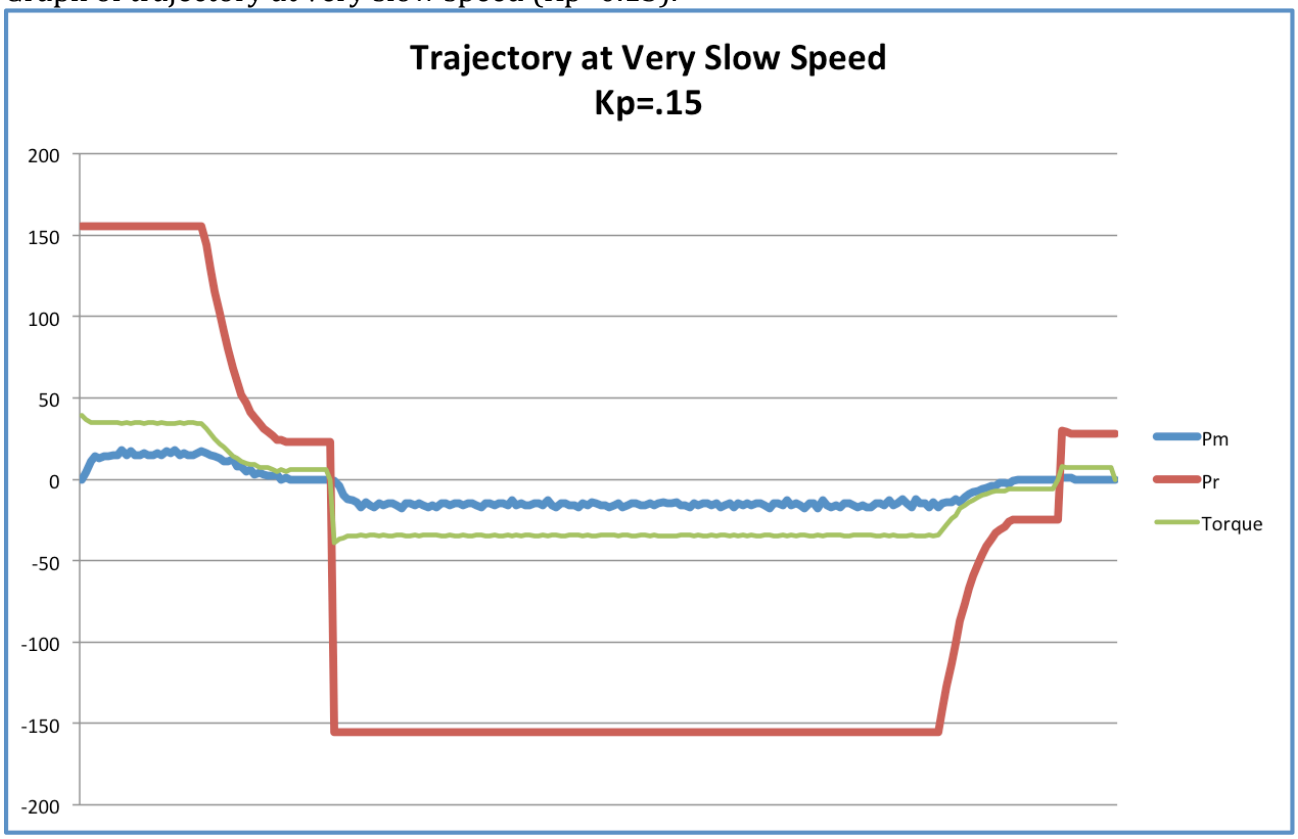
4. Graph of trajectory at fast speed (Kp=1.7)



5. Graph of trajectory run at a slow speed (Kp=0.5)



Graph of trajectory at very slow speed (Kp=0.15).



From the three graphs above it is clear that maintaining consistent control over the motor is easier to do at low speeds than at high speeds. As I tried to tune the PD controller for position, I found that at near-maximum speeds it was very difficult to not overshoot the destination. In fact, I wasn't able to find a combination of P and D gains that consistently produced no overshoot. On the other hand, at slower speeds, there was more opportunity to gather data on the error and to correct for it sooner. Thus the range in torque that was sent to the motor is less at low speed than at higher speeds. This produced much smoother motor function.

At slower speeds I consistently undershot the destination for each segment of the trajectory. I also found that the D gain at very slow speeds had little to no effect on how close I could get to the desired position. In general, the gains at slower speeds had a less dramatic effect on the result of the PID equation, as evidenced by the less erratic torque value when the motor changed speed and direction.