Giovanni Galasso
SENG 5831 Lab 2
Spring 2015

Note: Everything is in encoder counts per 10 ms. So for speed I might say speed:20, which is 20 encoder counts per 10 ms for my code, but in reality it might be more like 2000 encoder counts per second.
dt = 1

Part 1:

******
P Only
******
Set Kp = 0
Set Ki = 0
Set Kd = 0

Target Speed = 20

Kp = 0 - I wanted to test Kp=0 to see what happened, in that case the motor becomes unresponsive, either it maintains it current speed or won't move from speed 0 to any other speed.
Kp = .1 - never reach target of 20 speed, seem to stabilize around 12.
Kp = .3 - again never reach target, undershoot to about 18 speed.
Kp = .4 - hit my target of 20 speed.
Kp = .5 - overshoots just a little, more like 21 speed rather than 20.
Kp = .8, 1, 1.3, 1.5 - I gradually increased Kp. I did notice that values above 1 or so that my Motorspeed was higher than at Kp .4, both to maintain a speed of 20. I also noticed slightly oscillation when starting the motor at values of about 1 and higher.
Kp = 2, 3, 5 - When I got to 5 I started to see it oscillate some, but just slightly.
Kp = 8, 10 - saw some really large oscillation. At 8 and above I noticed the motor switching speed from forward to reverse.
Kp = 20 - started flipping around on the table. I assume the error was so large that the motor was overshooting the target and then reducing the speed causing the massive oscillation between forward and reverse directions. I did some print outs while viewing this behaviour and noticed the Motorspeed anywhere from +255 to -180 (opposite direction).
******
PI Only
******
I reset my P back to a known stable value .4, and added some I back in.
Kp = .4
Ki = .01
Kd = 0

Target Speed = 20

Ki .01 -produces a fairly close result to just using P. The I value balances close to zero between positive and negative. It overshoots slightly to 21 and then balances itself out very quickly to maintain 20 speed.
Ki .001 - made me go under to 19, and then up to 20 after a little time and finally maintain at 20.
Ki .02, and .03 - produced about the same result, pretty quickly hitting 20 speed with very little oscillation. Occasionally going under or over the target.
Ki .05 - Started to notice a slight oscillation at the very start, but quickly balances out and hits target speed pretty quickly.
Ki .1 - Started to notice a problem here, motor cannot maintain the target speed, and goes over and under never reaching the target.
Ki .2 .3 - motor browns out, oscillates so much that it becomes unstable and shuts down.

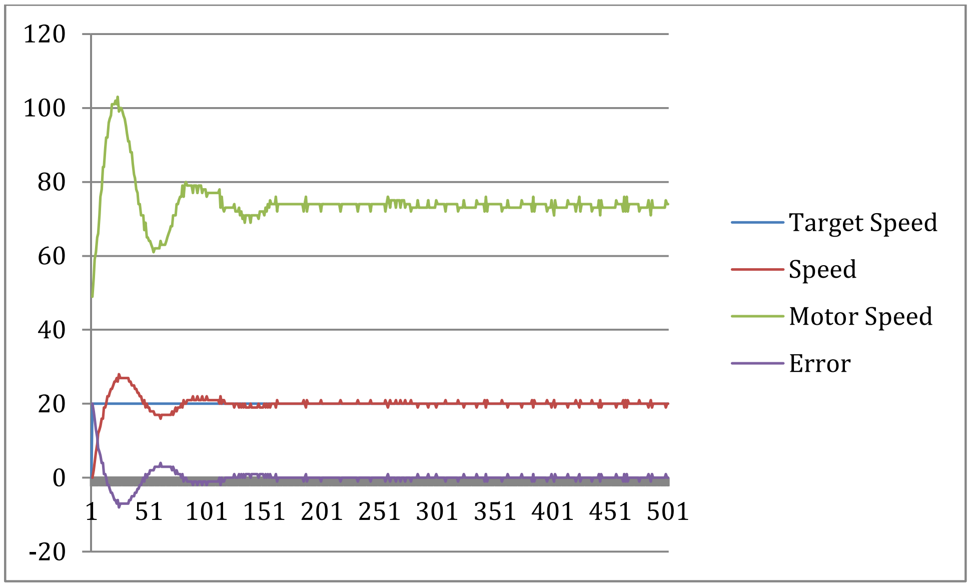
******
PI Only
******
I wanted to create some oscillation with I, to see if D balances it out. So starting Ki with .1 which was unstable before.
Kp = .4
Ki = .1
Kd = 0
Target Speed = 20

Kd = 1 - This works much better then just Kp .4 and Ki .1, much more stable with D added in to the formula but still not perfect, never really reach target of 20, go quickly from 19 to 21.
Kd = 1.5 - Started the same as above, flipping between 19 and 21 but after several seconds finally stabilized at 20 precisely and maintained a speed of 20.
Kd = 2 - Balances at 20 much faster than with 1.5
Kd = 3 - A little less stable to start with than 2, but balances out and maintains 20 speed quickly.
Kd = 5 - About the same as 3.
Kd = 10 - Slightly more oscillation at the start but balances out pretty quickly.
Kd = 20 - Started to notice a much bigger oscillation jumping from 25 to 15 speed and number in between, after maybe 15 seconds or so it balanced out at 20.
Kd = 30 - Very unstable, jumps all over the places and never comes close to the target.

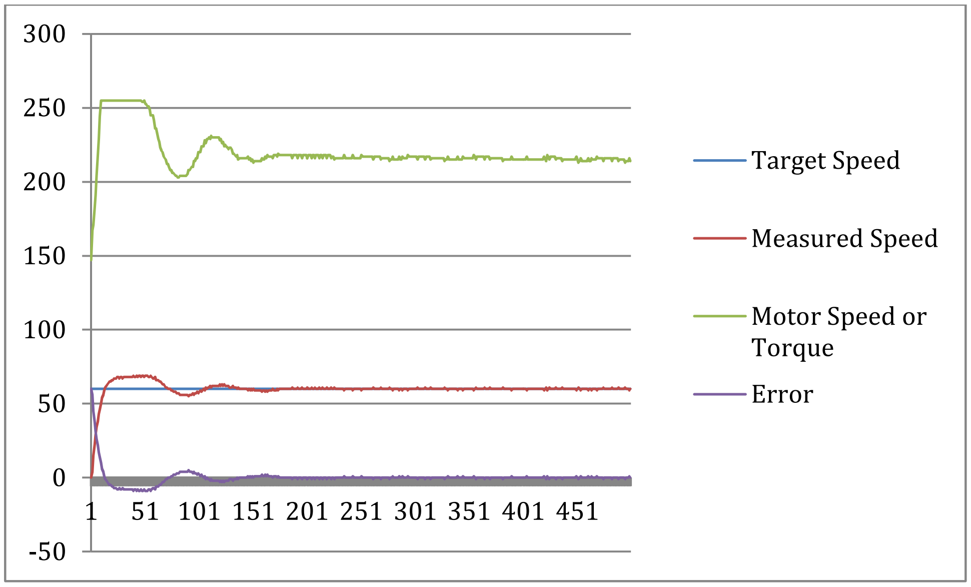
Observations - I gained a much better understanding of how the P - I - D terms effect the motor control. I was amazed to see how the I causing oscillation is balanced out completely by the D. I was also impressed by just how stable using a P value alone tended to be. I almost felt like enabling logging for these so I could get a better understanding of how stable it was with PID vs just P, or some values for PI, etc.

Couple graphs for speed:

This is to achieve a speed of 20 encoder counts per 10 ms. Graph shows the ratio of motor speed to settling speed of our target. Time is in 10ms scale, so 51 is about a half a second. As you can see it takes about a full second before our motor settles on 20 encoder counts per second. It would be interesting to graph more of these and see if we can find a faster settling time. We can see at the start of the graph how there is a slight oscillation, this is noticeable in both the error and speed.



This is to achieve a speed of 60 encoder counts per 10 ms, near the top of our max speed. Again this gives us a nice idea of how the motor speed reaches the top before settling down towards the desired speed of 60 (the top is about 74 or 75). We can see at the very start how the error shoots to 0 with very little oscillation. It also maintains a constant motor speed of 60 pretty well. Scale is in 10 ms , so 50 is roughly .5 seconds.



Part 2:

Start with last known stable speeds:
Kp = .4
Ki = .1
Kd = 2
Target Speed = 5, 10, 50, 70 - our max is about 74. So this will give us a good sample of a slow speed and a fast speed.

Speed = 10 - Starts a little slowly but then balances to 10 pretty quickly.
Speed = 5 - Again starts slowly but then balances to 5 with a second or so.
Speed = 50 - Seems to miss the target a little.
Speed = 70 - fluctuates around 72 to 68, before settling closer to 70 but never hitting 70 exactly.

Ki = .03
This was much better performance and much more reasonable. Eliminates the speed fluctuates that we were getting with Ki = .1
Speed = 5 - Again starts slowly but then balances to 5 with a second or so.
Speed = 10 - Starts a little slowly but then balances to 10 pretty quickly.
Speed = 50 - hits the target pretty quickly
Speed = 70 - fluctuates just slightly before settling on 70.

Ki = .05 seems to be even closer to my target, settling down on my target speed very quickly for all of the speeds. This seems to be the best gain I've used so far in getting to where I want quickly and not fluctuating too much.

Adjusting controller:

Controller = 4 * ISR or 40 ms adjustments
Speed = 20 - Shoots up to 25 and then takes several seconds before coming back down to 20.
Speed = 5 - Fluctuates around 5, 6 or 4 for a few seconds before settling on 5
Speed = 10 - Goes over and then under, 12 to 9 before settling on to 10 after a few seconds.
Speed = 50 - Goes way over, up to 62, before going towards mid 40's and settling around 50 and finally settling at 50.
Speed = 70 - Goes up to max speed, before finally getting closer to 70, after awhile it settles precisely at 70.

Controller = 10 * ISR or 100 ms adjustments
Speed = 5 - Goes much slower up to 6, before settling onto 5 after a little while.
Speed = 10 - Slowly builds up to 10 and passes it to 12, and then comes back to 9 and eventually lands on 10.
Speed = 50 - Large oscillation, spins up slower than normal and eventually goes up to 62. Goes back down to 47 before fluctuating around 50, between 52 and 48 and finally lands on 50 after several seconds.
Speed = 70 - Goes up to max speed for a while, 15-20 seconds before finally slowing down slightly. Takes another 20 seconds or so to get to 70 precisely.

Observations: It seems to really make a big difference in how often you adjust your PID controller. At 40 ms controller setting I found the motor to be much more jumpy, often over or undershooting. Attempting it at 100ms made an even bigger difference, up to the point where the motor took a long time to get to a high speed and often overshot the desired speed by quite a bit. Oscillating was much more noticeable at these lower frequency updates.

Just for fun I threw in a controller update rate of 500ms (50x my normal update frequency) and set the target speed to 20. This was interesting to watch, it was like watching what my motor might normally do in slow motion and I'm guessing with even greater over and under shoot due to the slower updates. It seems to be that the frequency of your updates is almost as important as your PID implementation. If you update too slowly it will take much longer to get to the target speed.

Part 3:

******
Positional
******
Set Kp = 0
Set Ki = 0
Set Kd = 0
Position = 2000

Starting with Kp for adjustment.
Kp = .5 - brown out
Kp = 1 - brown out
Kp = .02 - Never reach target of 2000, get to 1667 and stop.
Kp = .01 - Even worse, only get to 1321. My error is 679, but multiplying by .01 only gives me a 6 for torque, not enough to spin the motor any further.
Kp = .03 = Gets to 1778 so moving in the right direction.
Kp = .05 = 1874
Kp = .07 = 1921
Kp = .1 = 1937
Kp = .15 = 1962
Kp = .2 = 1976
Kp = .25 = 1985
Kp = .265 = 1985
Kp > .265 - causes the board to occasionally brown out.

Adding in Ki
Kp = .25
Ki = ?
Kd = 0

Ki = .001 - gets me to 1988
Ki = .01 - gets me slightly over 2000 and slightly under, very close to my actual desired position and stops completely.
Ki = .015 - this value oscillates at the target position, e.g. if we want 2000 we'll switch often between about 2020 and 1980, taking a long time if ever to reach our target.

Adding in Kd
Kp = .25
Ki = .01
Kd = ?

Kd = .1, .2, .5 - Not a huge difference between these three values, all of them seem to work pretty well.
Kd = 1 - Close to my target.
Kd = 4 - Close to my target with a little overshot.
Kd = 6 - A little more off than 4.

Not sure what value works best here, going to go with .3 since that was about the lowest overshot.

******
Starting off really slow

******
Kp = .01
Ki = .01
Kd = .3

This accomplished getting to my target with a slower speed. Reducing the Kp gain really reduces how fast you will go to start because the error is not huge to begin with. The difference now compared to before when only using P is that our I and D will help us get to our final destination rather than just stopping the closer we get.

******
Using only PD
******
Kp = .25
Ki = 0
Kd = .3
target = 2000

target = 2000 we get to 1976
target = 5000 we get to 4977
target = 20000 we get to 19977
target = 300 we get to 281
target = 50 we get to 25
target = 30 we don't ever move from 0

It's pretty obvious that without an I we will always undershoot our position because we don't have the cumulative error to keep us moving when the error starts approaching to 0.

******
Positional gains for overshoot
******
Kp = .25
Ki = .01
Kd = .3

This provides slightly going over the target location.

Adjust a little to see if we can still go fast but not go over quite so much:
Kp = .2
Ki = .01
Kd = .3

Still going over too much, so try slowing it down a little:
Kp = .15

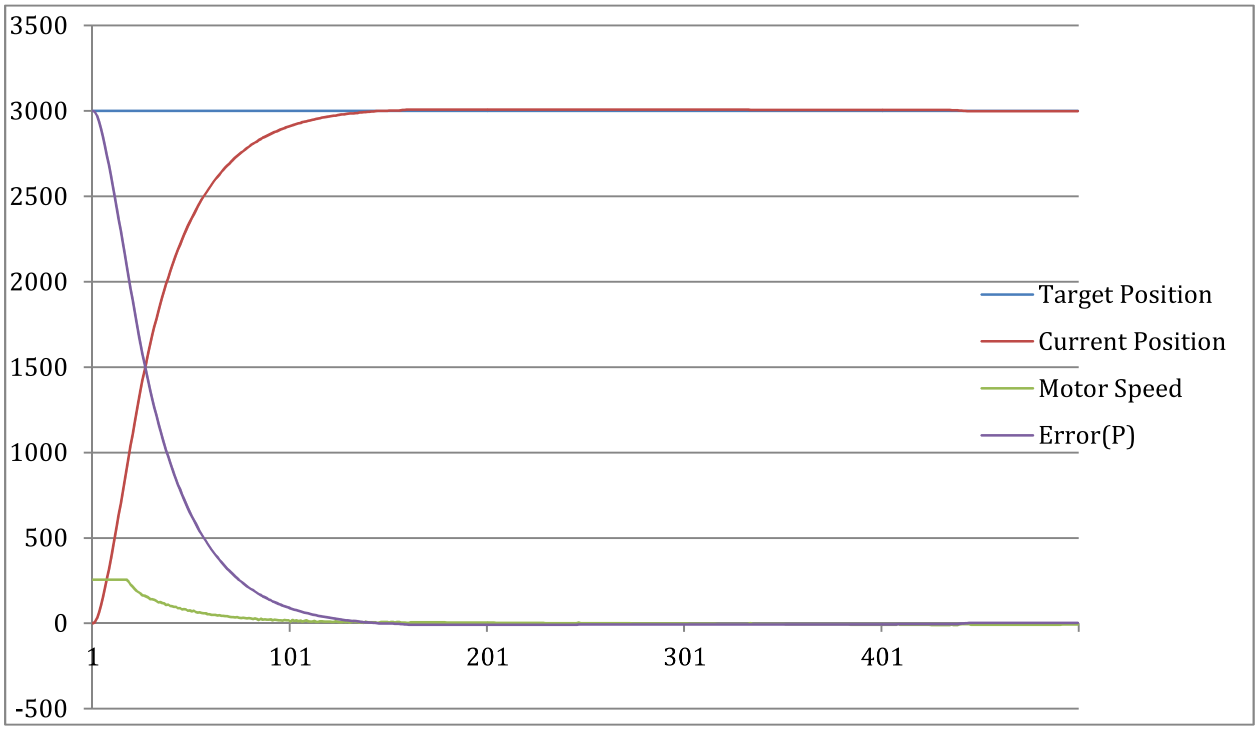
Ki = .01
Kd = .3

Realized that adding in more D would help it not overshoot:
Kp = .25
Ki = .01
Kd = 1.5

Still going over a bit, so dampen even more:
Kp = .25
Ki = .01
Kd = 4

The above was accurate in terms of getting to the position without going over and maintaining a reasonably fast speed right until we get near our target. There is a slight overshoot occasionally but usually not more than about 10 encoder counts, which is within a degree of precision.

We took the below sample to target = 3000. Logging 5 seconds worth of data with a sample every 10 ms.

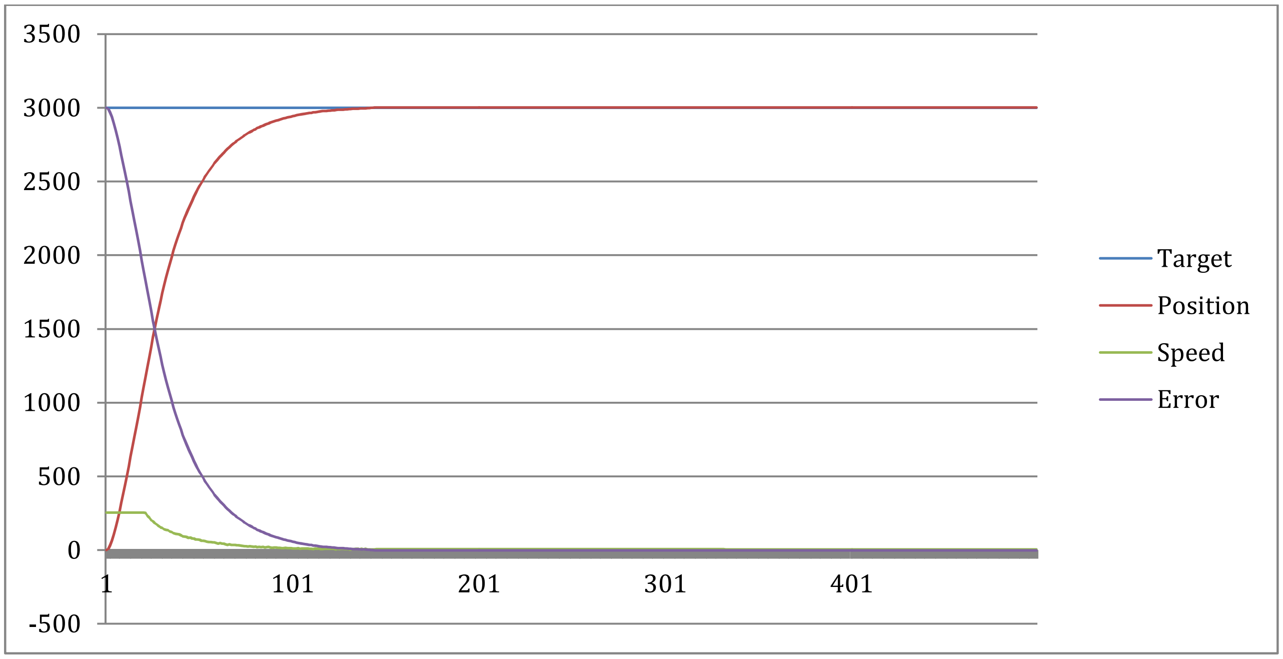


Y axis is in encoder counts
X axis is in samples per 10 ms, so 101 is like 1 second 10 ms.

Because I didn't quite settle on my position exactly where I wanted I decided to try playing with my values again.

This should help reduce my overshoot a little and get me closer to my target.
Kp = .25
Ki = .0077
Kd = 3.1

This was about as precise as I could get my positional without spending a lot of time tuning it. This reduced my overshoot to usually no more than about 2 encoder counts and usually got me within 1 and often right on the target.



Y axis is in encoder counts
X axis is in samples per 10 ms, so 101 is like 1 second 10 ms.

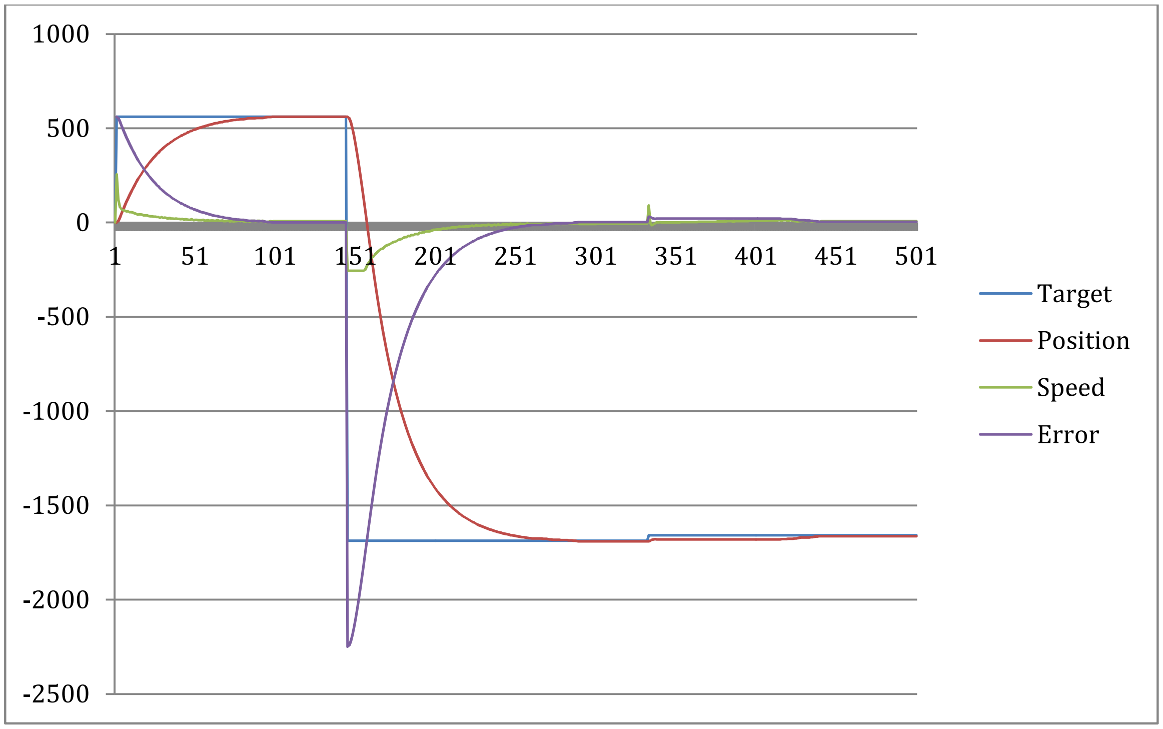
You can't really see by the graph but the second constant values are much tighter.

Part 4:

Using the values determined in part 3 for optimal target hitting with good speed.
Kp = .25
Ki = .0077
Kd = 3.1

Interpolator set up by using 'U' to go 90 degrees forward, reverse 360 degrees and then go forward 5 degrees. Pausing by .5 seconds after reaching each target.

For measuring .5 seconds after reaching the target I created a settle counter that is +/- 6 (or 1 degree).



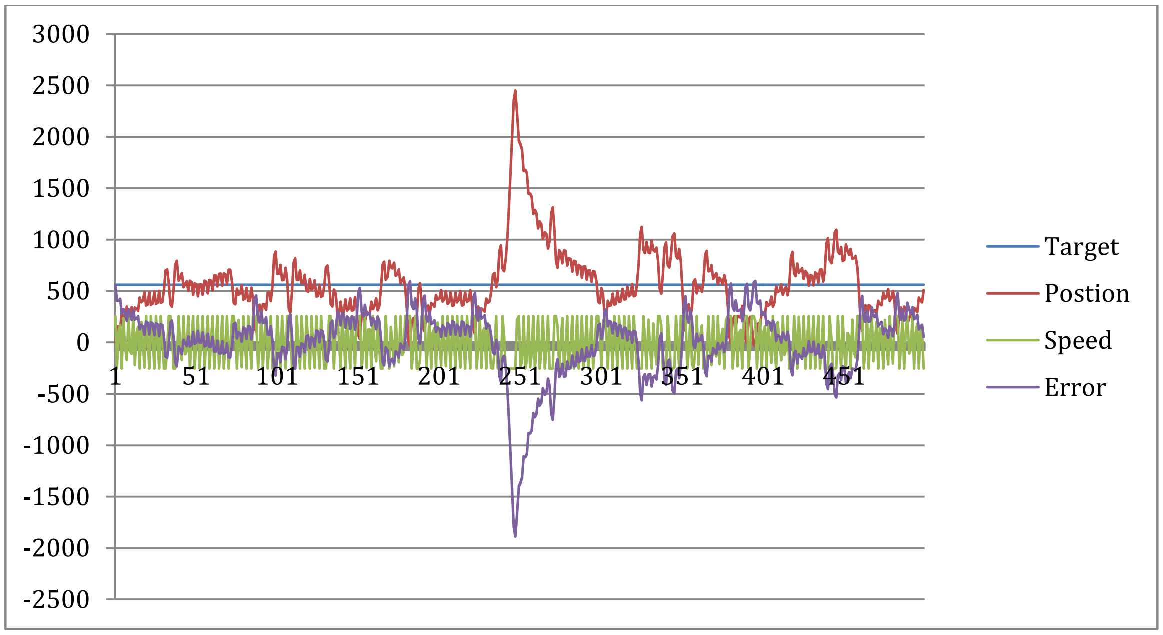
Y axis is encoder counts and X axis is in 10ms counts, so 51 = 510 ms, or .51 seconds.

You can't see this from the graph but my great positional control when using one command at a time was not very good here. It had a real hard time only moving 5 degrees.

Part 5:

I was getting some flakey behavior of hitting my target, I'm not sure what changed, but I decided to increase my Ki a little for the next experiment. Before changing the frequency of the controller I ran the interpolator for the below values and was pretty accurate.

Kp = .25
Ki = .008
Kd = 3.1

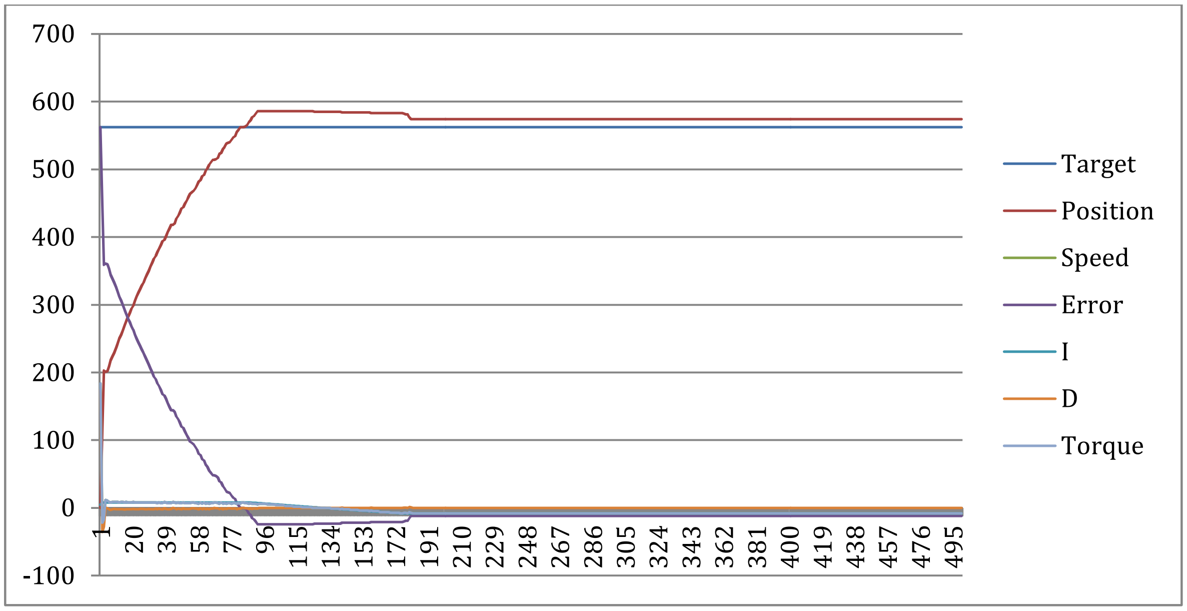


Starting with a frequency 50ms. The controller should update every fifth fire of the ISR.

This just went crazy, never even reaching my first target. I'll have to reduce the speed to see if we can do any better.

Tried – still crazy oscillation.
Kp = .1
Ki = .008
Kd = 3.1

This worked a little better, but I never quite got to my target.
Kp = .01
Ki = .008
Kd = .31



Y axis is encoder counts and x axis is based on 50 ms values, so 50 is about 2.5 seconds.

This worked a little better, but I never quite got to my target, but at least I didn't get stuck. It seemed the board browned out at the end, I can only guess this is basically of a quick change between forward and reverse
Kp = .05
Ki = .04
Kd = .6

Got me to my first target but browned out the board when trying to go to the second. Obviously must be going too fast at some point.
Kp. = 09
Ki = .007
Kd = .6

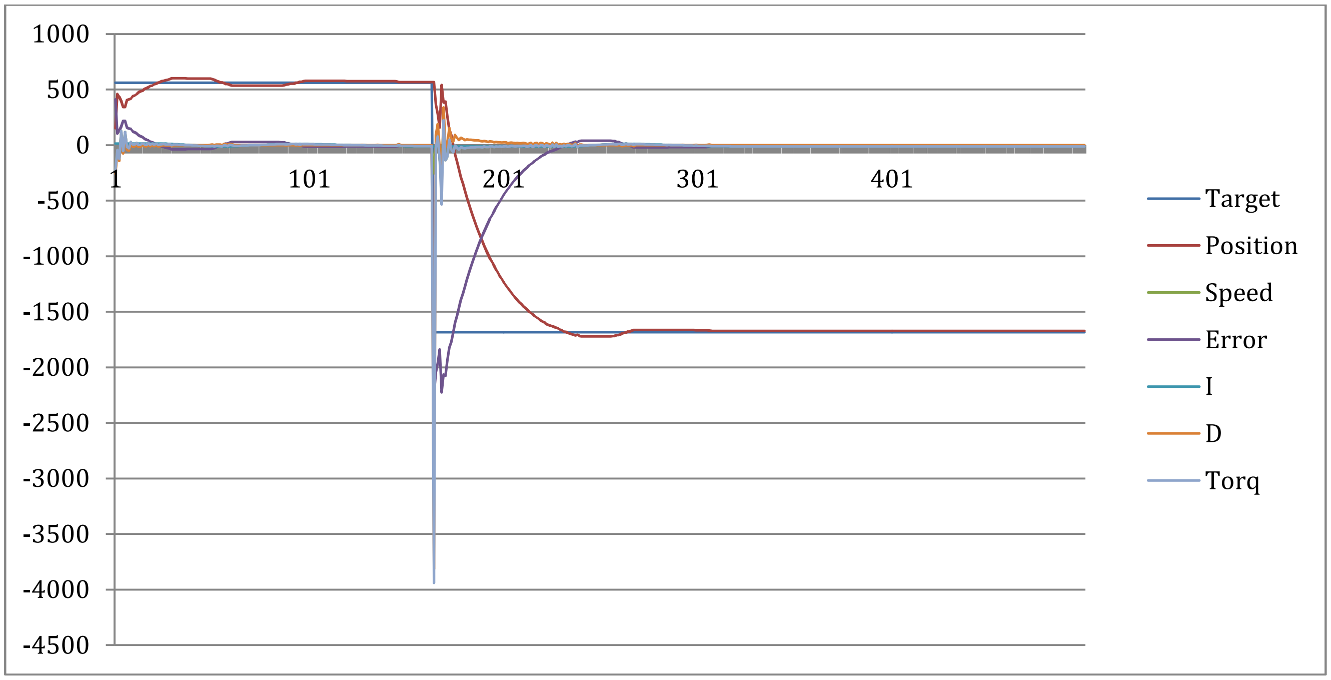
Hits the first, almost gets to the second!
Kp = .05
Ki = .008
Kd = .9

Got to all three positions! It took awhile and had quite a bit of oscillation but it worked.
Kp = .05
Ki = .014
Kd = 1.5

Tuned a little.
Kp = .05

Ki = .013
Kd = 1.5

I had to delay my logging period in order to capture everything (I was unable to boot my board if I made my log array's too big).
The period for the log is every 330 ms or 3 times per second. So for the x axis each 100 is roughly 33 seconds.



It eventually balanced out on the third location but this wasn't captured in the log.

I tried using a controller frequency of every 200ms as well and was surprised by how stable it was compared to my 50ms frequency. There was very little difference between the two. My guess is there was already a maximum of error that was occurring.

This was a hard one to understand, I'm not sure there is an exact answer, but what I learned is that tuning the PID can be very difficult especially when the frequency can be changed. A slight change in frequency requires a great deal of tuning to approach any previous accuracy. I also believe that the slower your frequency calculations the greater the fluctuations will be in reaching your destination. I would also guess that the faster you calculate your PID the more accurate you could be, and was curious to test a faster sampling rate (say every 1 ms) to see what the effect would be.