Jesse Estum

Embedded Systems

Lab 2

Spring 2015

0. Source Code

Please see: https://github.com/estu0002/EmbeddedSystemsHomework/tree/master/pid

1. Speed Controller

I set the frequency of my controller to execute once ever 100ms. This was based on my motor warm-up exercise calculations which found the following:

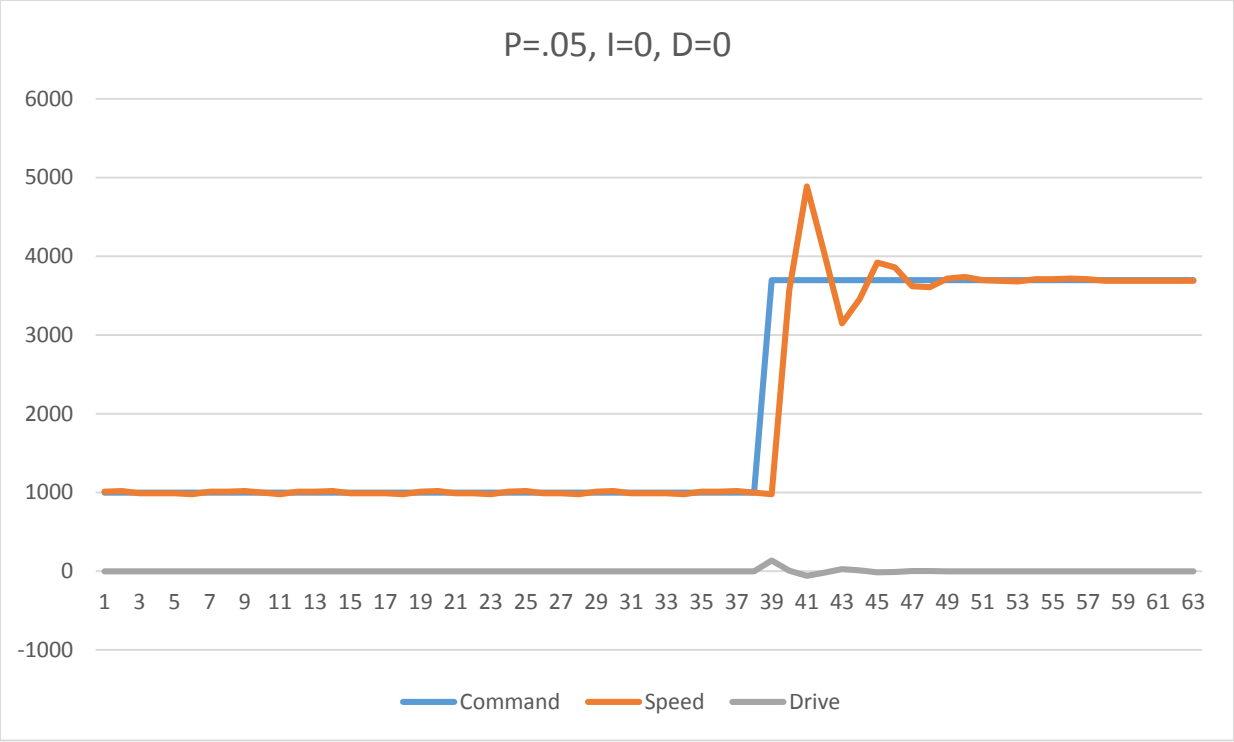
Motor speed signal (range -255 to +255)	Microseconds to get 48 counts (average based on 100 revolutions and 5 trials)	Microseconds to get 1 count (column to left divided by 48)	Counts per 1ms	Counts per 10ms	Counts per 100ms
7 *	455413uS	9488uS	.1053	1.053	10.53
255	6885uS	143uS	6.993	69.93	699.3

* 7 was chosen as the slowest possible value that could be used because anything lower was insufficient to overcome the static friction of the motor and gearbox

In order to calculate speed, you must divide change in counts by change in time. In order to work at low speeds, the number of change in counts only becomes significant with the higher time interval (100ms). Thus, in order to not get violent oscillations at low speeds 100ms was chosen for the speed controller interval.

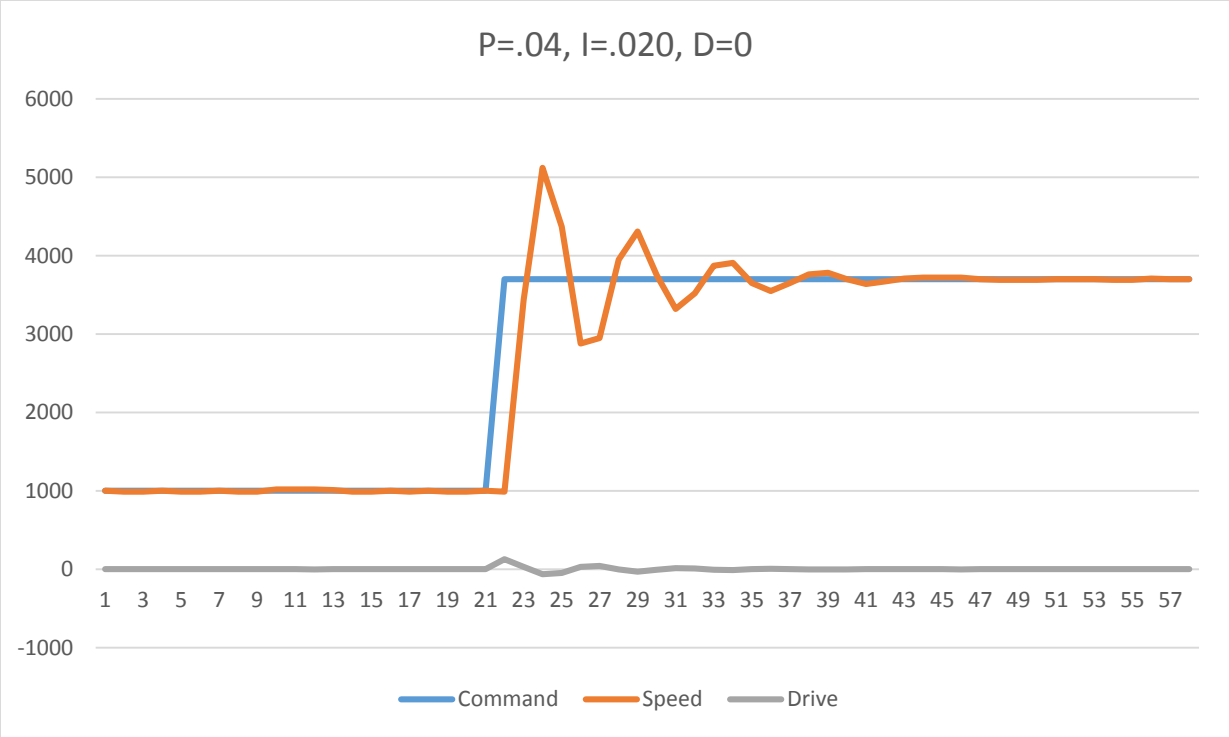
The maximum speed of the motor is about 7500 counts/sec (c/s), thus I interpreted an “average speed” to be about 3700 c/s. I tuned the controller to go from 1000c/s to 3700c/s.

Starting in with the proportional term, I worked in .01 increments. I could get up to .05 and arrive at the desired speed quite quickly. When I increased it to .06 (or anything beyond that), I experienced a bad overshoot and correction. The rapid reverse did not offer the opportunity to get oscillation because it caused my board to shut off. I suspect this is due to some sort of electrical backflow and the board has a built in protection mechanism for such instances.

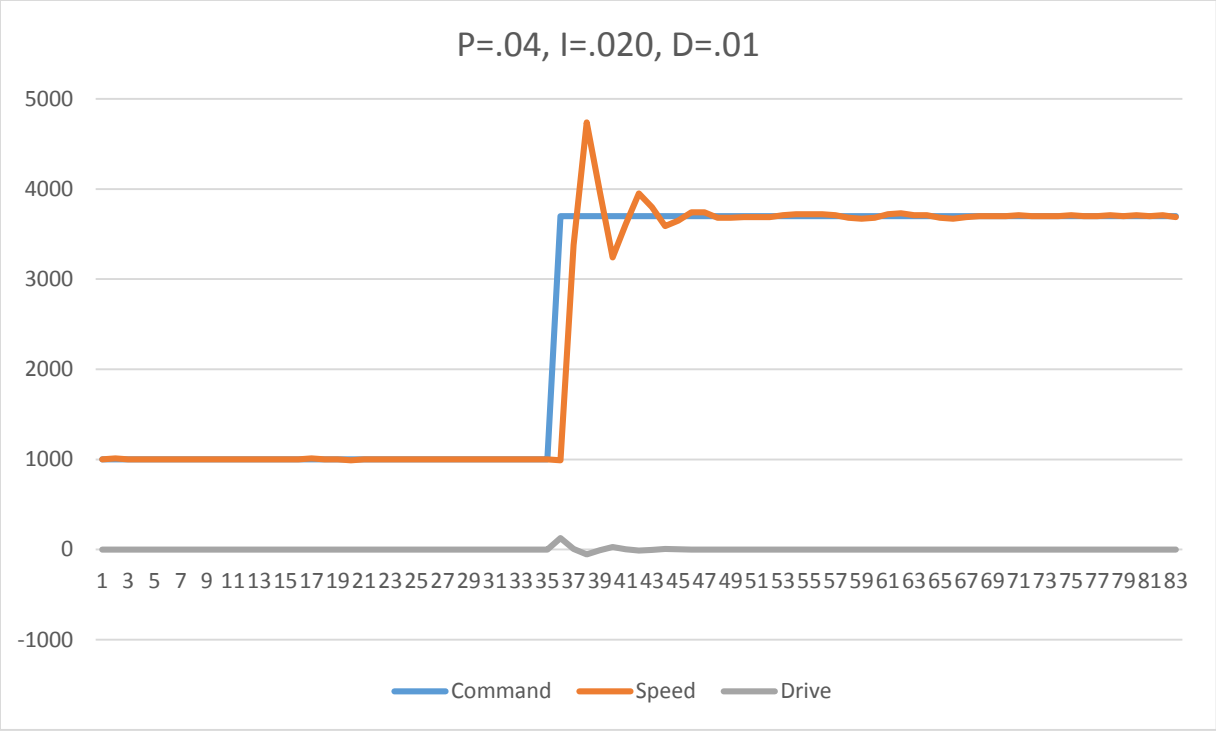


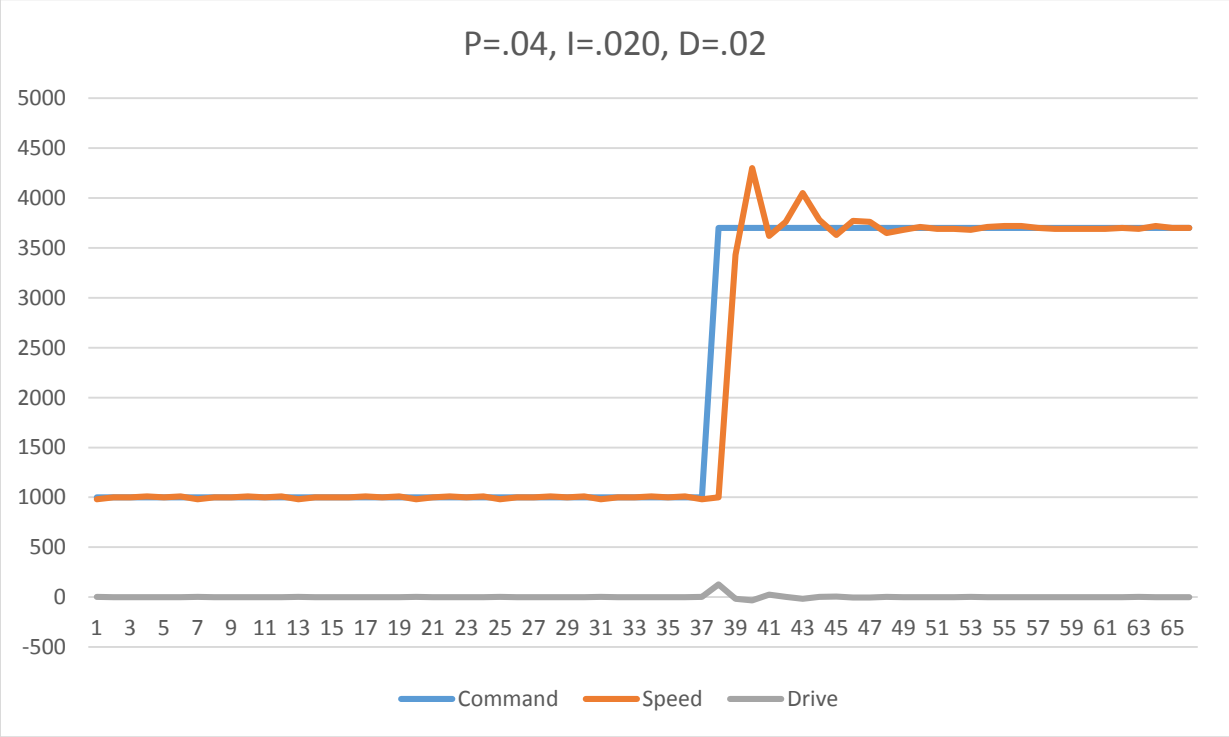
I then added in an I value, leaving P=.05. With my lowest possible I value of .01, I was able to go from 0c/s to 1000c/s with little noticeable oscillation. After setting the speed to 3700, however, I got a violent direction change that caused the board to shut down. I then experimented with increasing values of I (up to .15) and observed that the motor oscillated worse and worse at 1000c/s with higher values of I. I changed my increment amount for the I/i command to use steps of .005 instead of .01 and this still wasn't low enough.

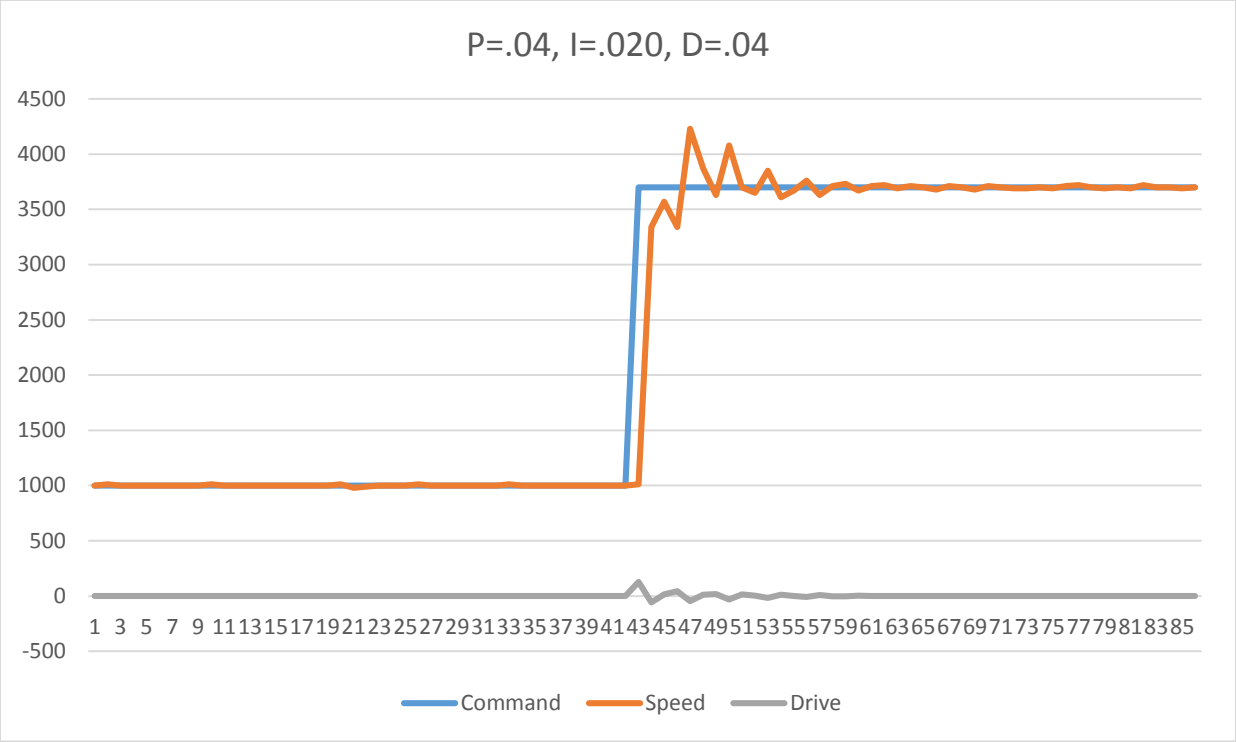
I then determined that P needed to go down. As such, I reduced it to .04 and was able to use an I value of .010. This produced a little bit of oscillation. I increased the I value until I go the board to shut off at I=.025, thus my max I value was I=.020.

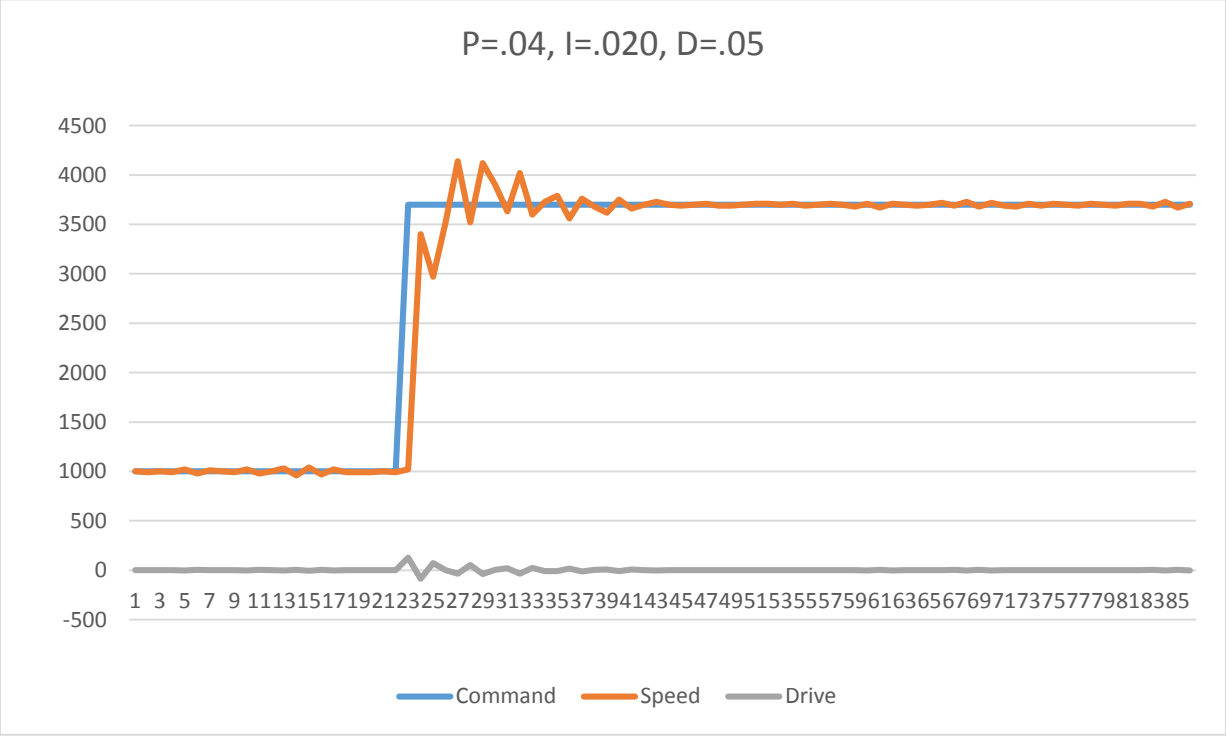


Next, I added in D value to try to dampen the response and eliminate oscillation. Per the assignment, I held P and I values fixed at P=.04 and I=.020. As I added in increasing values of D, I saw that the overshoot was dampened and the minima and maxima got closer to our desired speed of 3700. A negative side effect was that the system seemed to oscillate more (albeit in smaller amounts) with the higher values of D. Following are graphs to document the results.


P=.04, I=.020, D=.01
5000
4000
3000
2000
1000
0
-1000
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83
Command
Speed
Drive

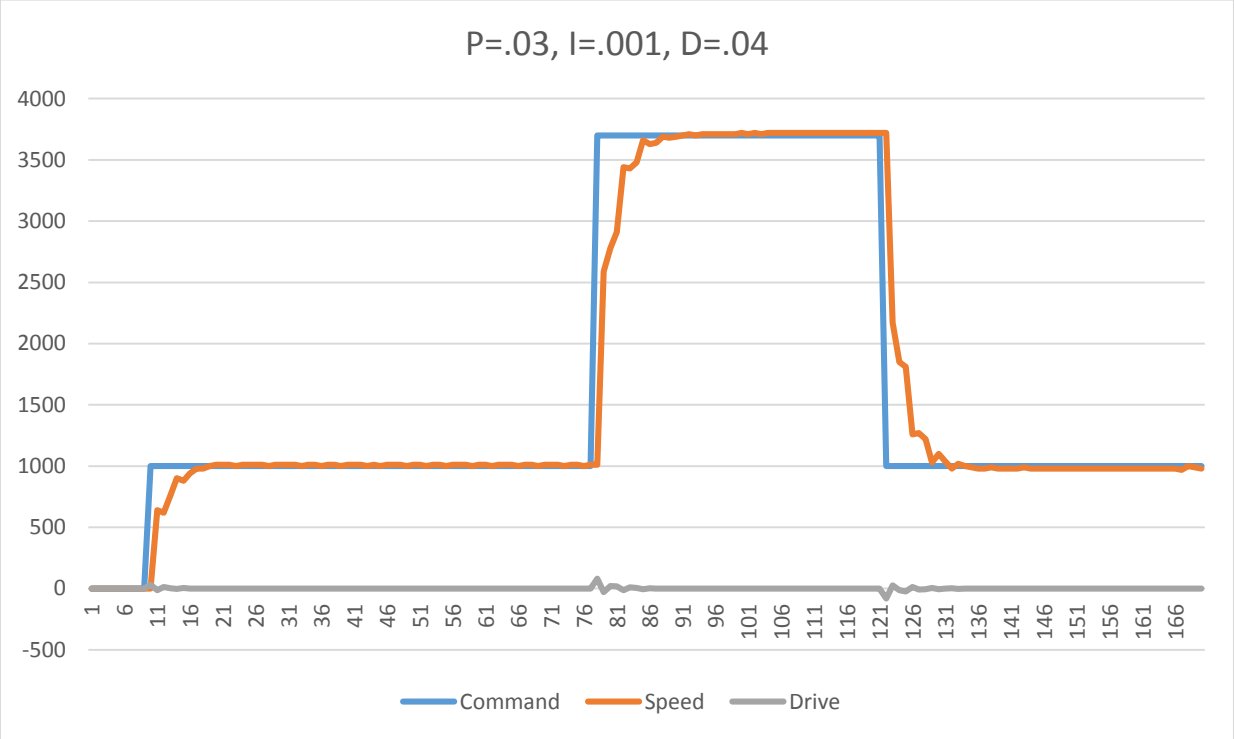

P=.04, I=.020, D=.02
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
-500
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65
Command
Speed
Drive





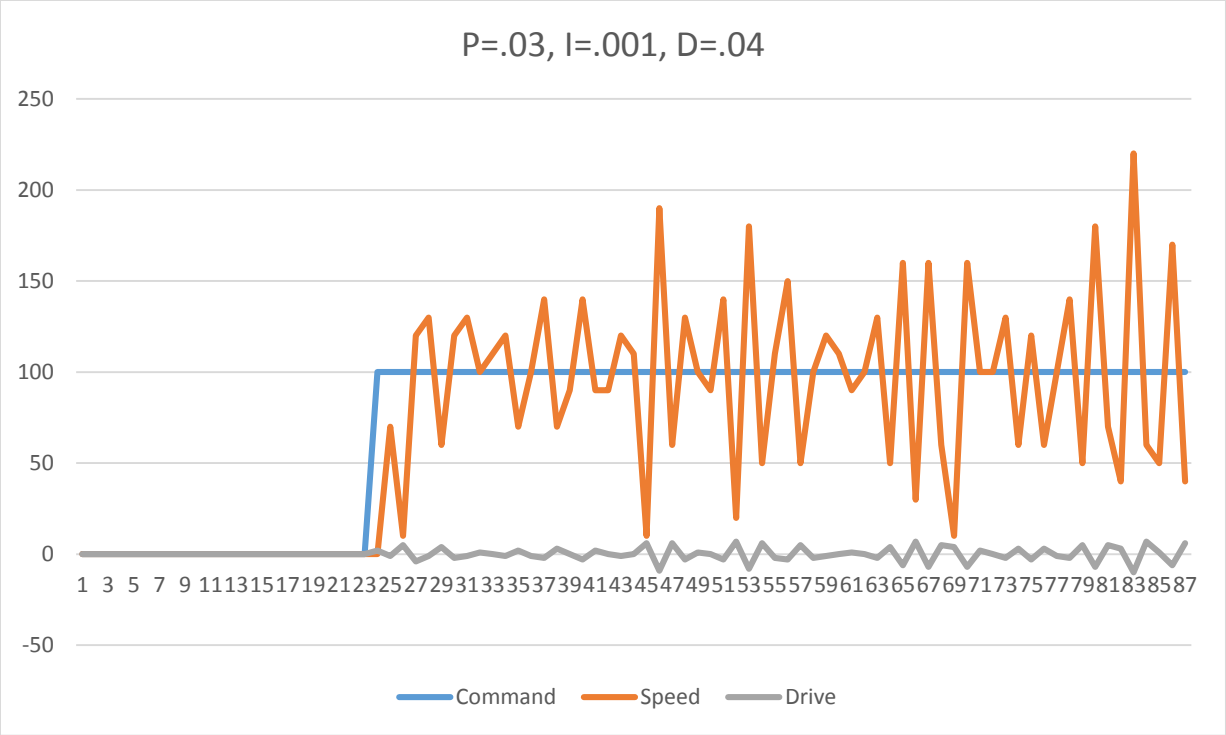
When I tried P=.04, I=.020, D=.06 the system was unable to change speeds from 1000 to 3700 without shutting off the board.

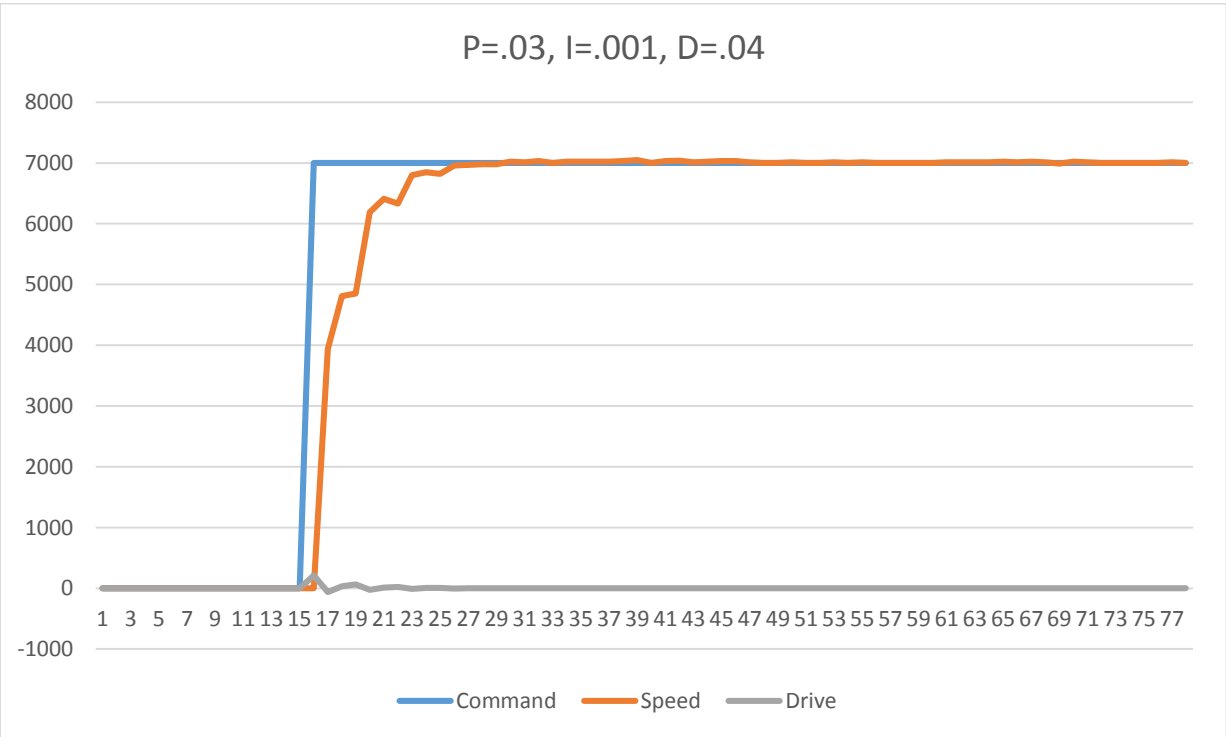
Taking into account the results from this D addition, I wasn't really satisfied with the way the graph was turning out. I decided to reread the PID without a PhD article and tune the PID controller by changing all values. I settled on the values illustrated in the following graph.



2. Exercising Gains at Various Speeds and Controller Frequency

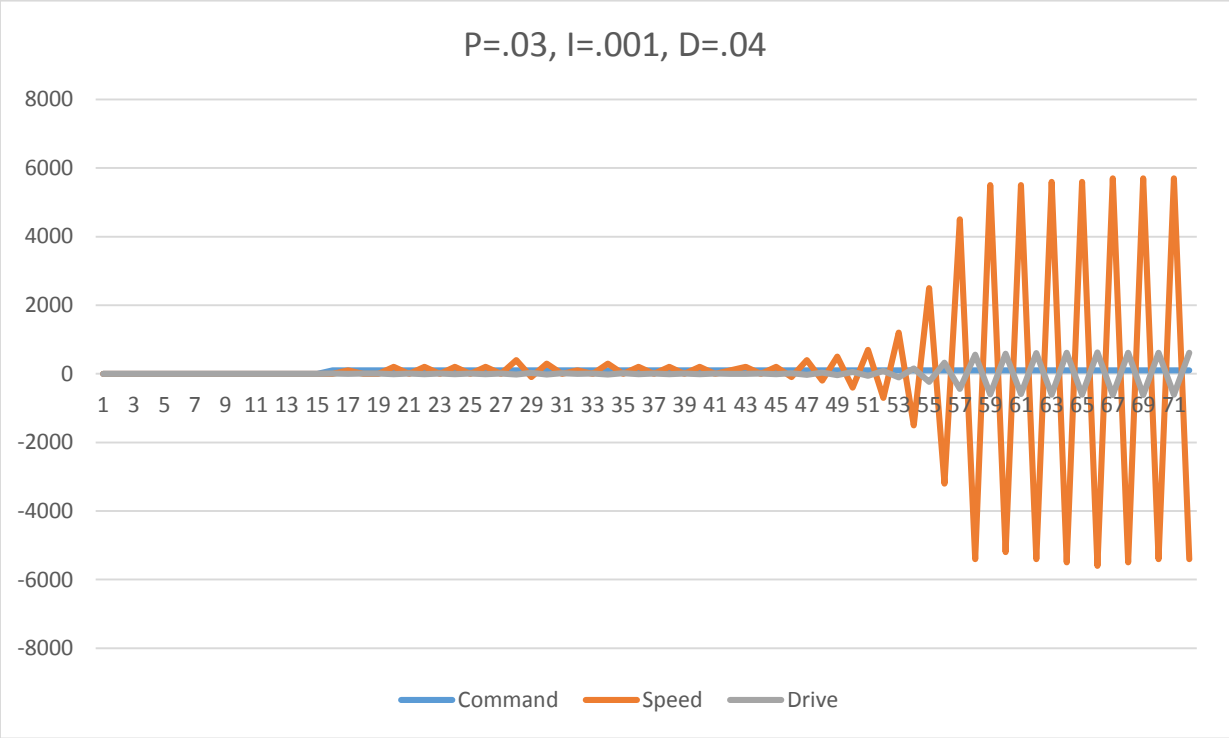
Using the controller config of P=.03,I=.001,D=.04 yields very different results at slow and fast speed extremes. In summary, it performs poorly at very slow speeds and exceptionally at very fast speeds. This is illustrated in the following graphs.





Given these drastically different performances at different speeds, I suspect there will be no “one size fits all” controller configuration for our motors, given their great variance in possible speeds. One possible solution would be to have multiple PID configuration values and to change those out based on speed ranges.

The assignment says to consider what would happen if the speed calculation was done more frequently. For this, I changed the speed calculation from happening once every 100ms to once every 10ms. At low speed (100 counts/sec.) this became an unstable system. At high speed (7000 counts/sec.), the system was so unstable that the board would shut off. A graph of the low speed scenario follows (high speed scenario data could not be obtained due to the board shutting itself off).



If anything, the speed should be calculated on a lower frequency than higher. This is because speed is a ratio of counts logged (numerator) to change in time (denominator). A smaller period (i.e., higher frequency) for speed calculation results in a smaller denominator which increases the quotient. The smaller period also decreases the amount of time to collect counts which increases margin for error. The increased quotient is then magnified because we have to convert the counts/period to counts/second which requires positive number >1 multiplier since our period is less than 1 second.

In summary, I find the best general results with a PID value of P=.03,I=.001,D=.04 and a 100ms period for calculating speed.

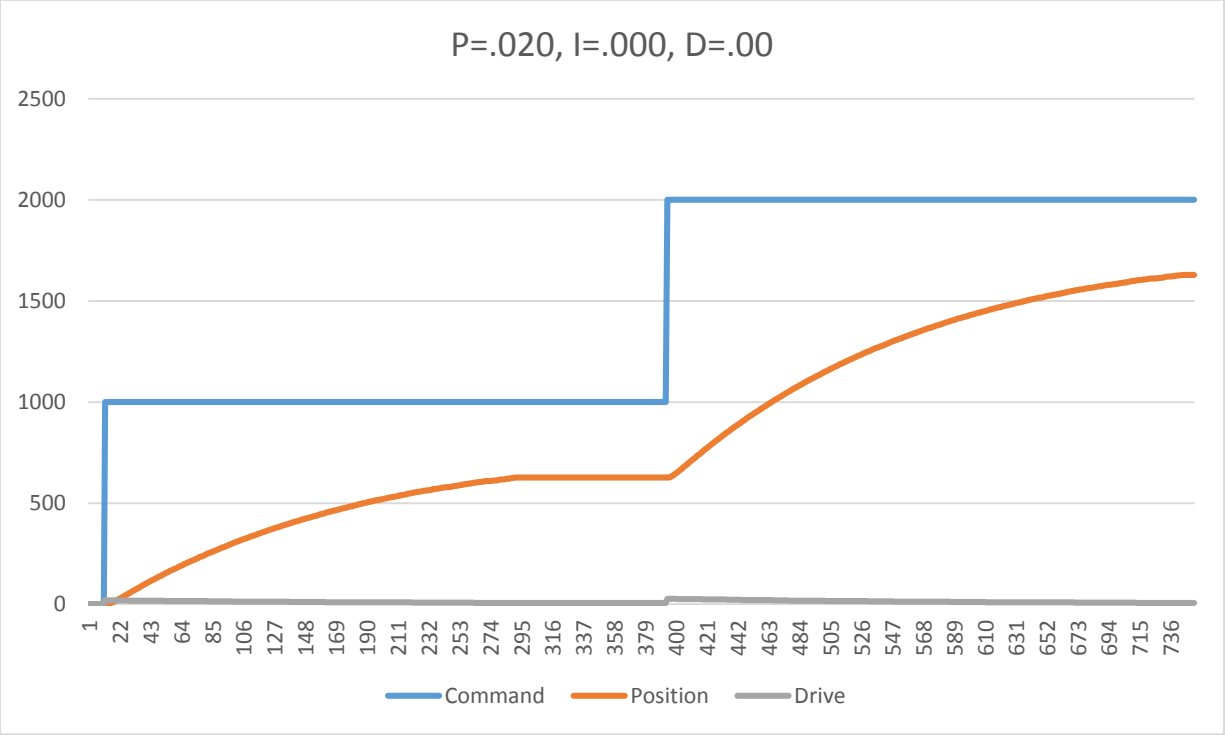
3. Position Controller

For this controller, I changed the way signal to the motor was being computed. For speed, incremental signals were being generated for the motor drive. For example, if the PID controller computes a drive value of 10 and the absolute signal to the motor was 100, then the absolute motor signal would change to 110. This is good because we're trying to sustain speeds.

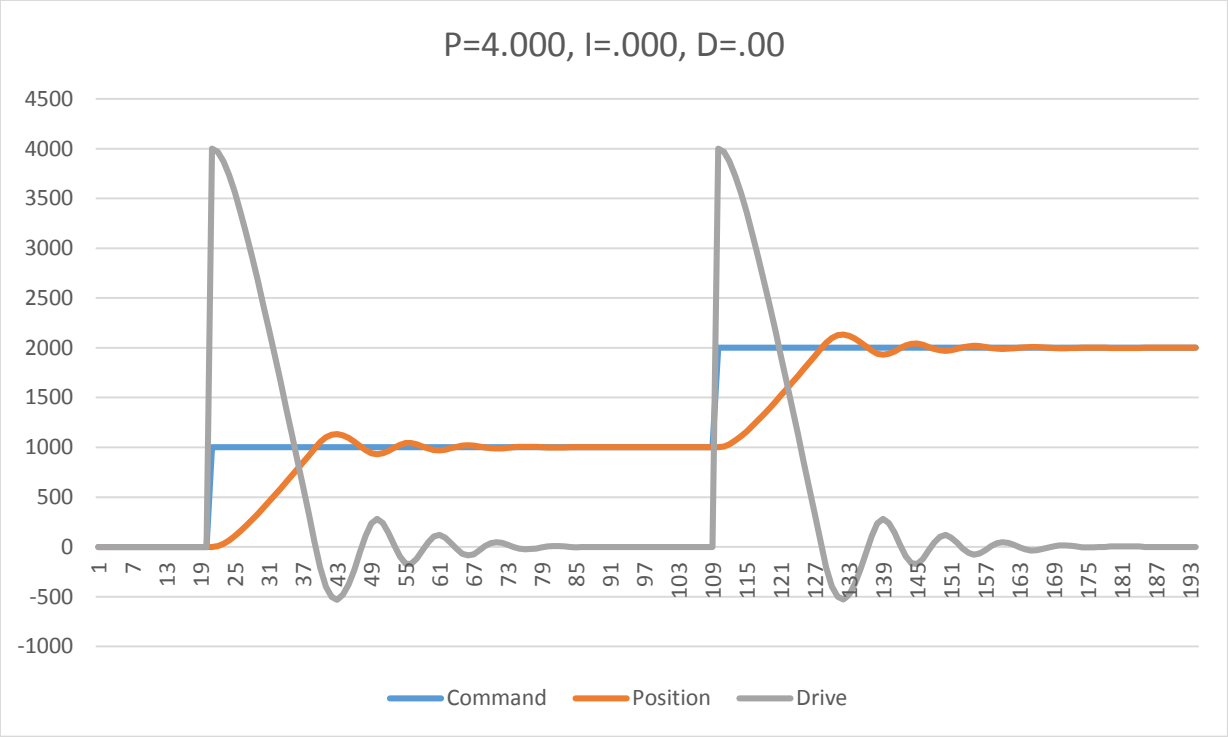
For the positional controller, we want to speed up then slow down quickly. It's important to get the absolute motor drive signal down to zero quickly once we're approaching the reference position. As such, the absolute motor drive signal will be applied directly from the computed drive value in the PID controller.

I also took into consideration that the rate of change in the encoder is much higher than that of speed. Thus, I increased the controller frequency to 100HZ (i.e., period of 10ms).

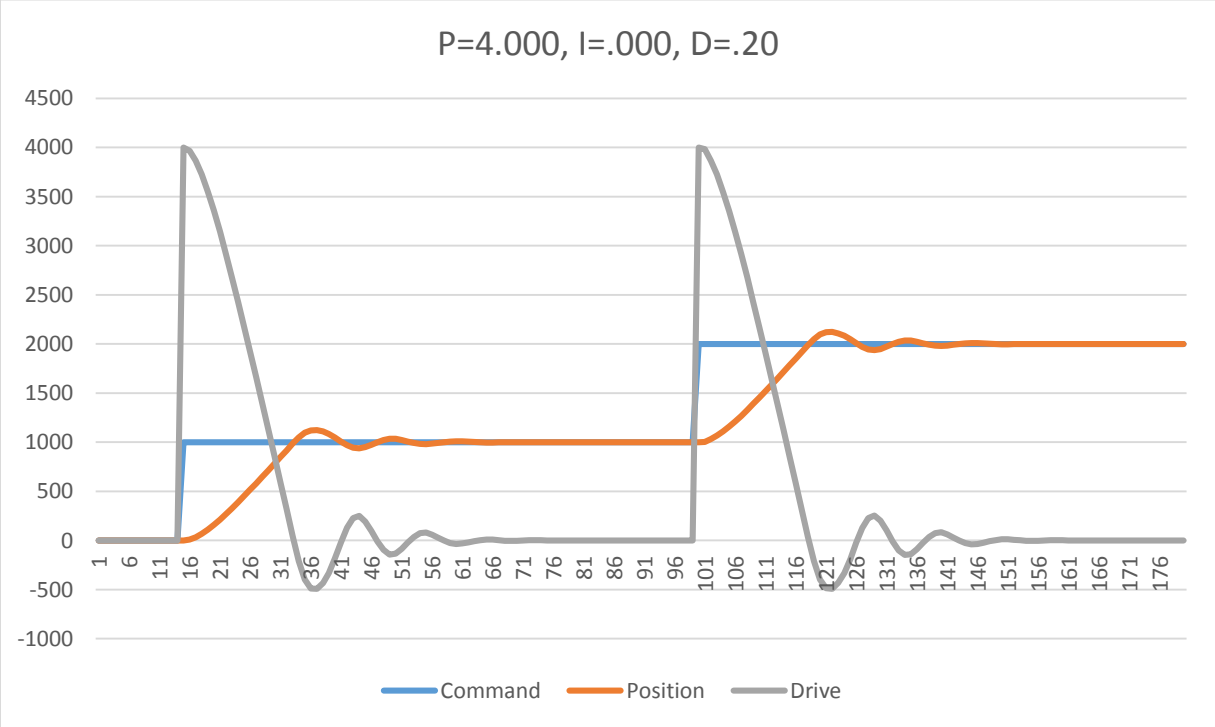
Initial values for the controller were P=.020, I=.000,D=.00. This resulted in a definite undershoot. Although it is difficult to see in the graph, the drive actually never goes down to zero once it's been given a reference position. Instead, it goes down to 7. When the drive signal reaches 7, there is no longer a sufficient drive to overcome the static friction of the motor and transmission. This explains why it stops short. See the following graph.

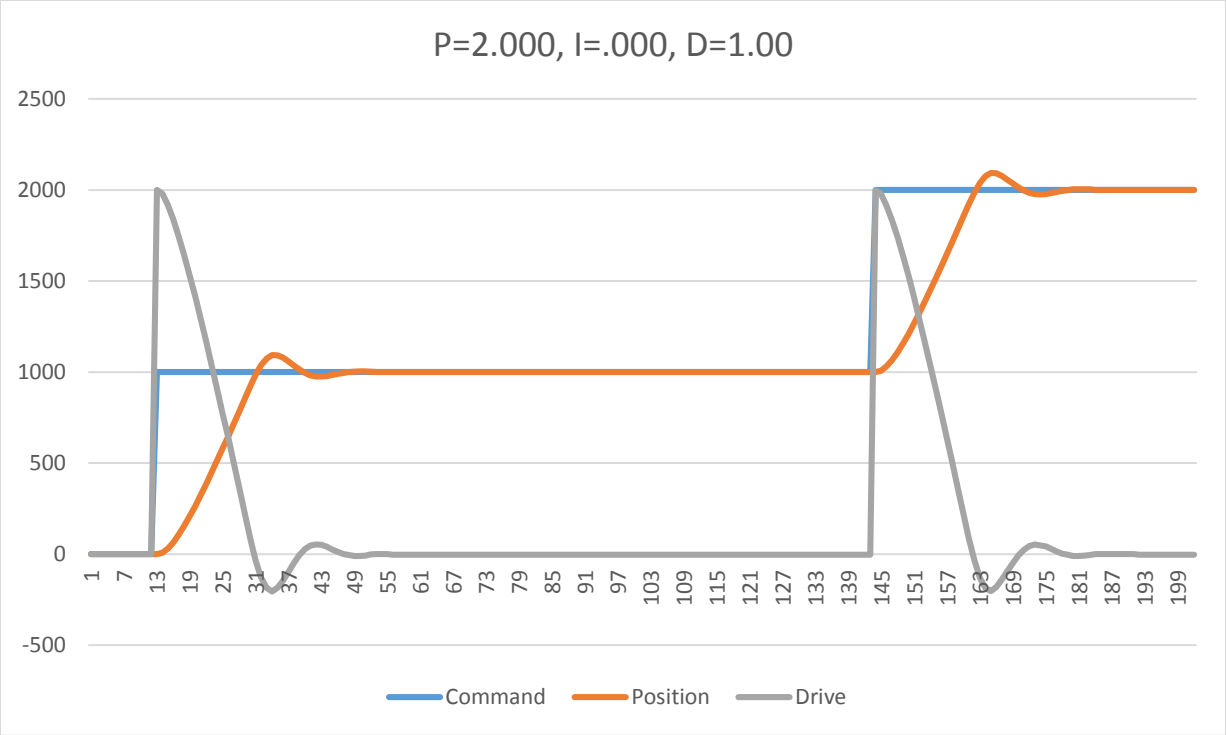


Next, I turned the proportional gain up substantially such that the motor would get up to full speed. I repeated the experiment and found that the motor could not only get quickly to the reference position, but it could get much closer (or, in my case it could *exactly* stop on the reference position). The down side is that there was some oscillation involved. Note that the drive plotted on the graph below is the value obtained from the PID controller, not what was sent to the motor. The maximum drive value that can be sent to the motor is 255.



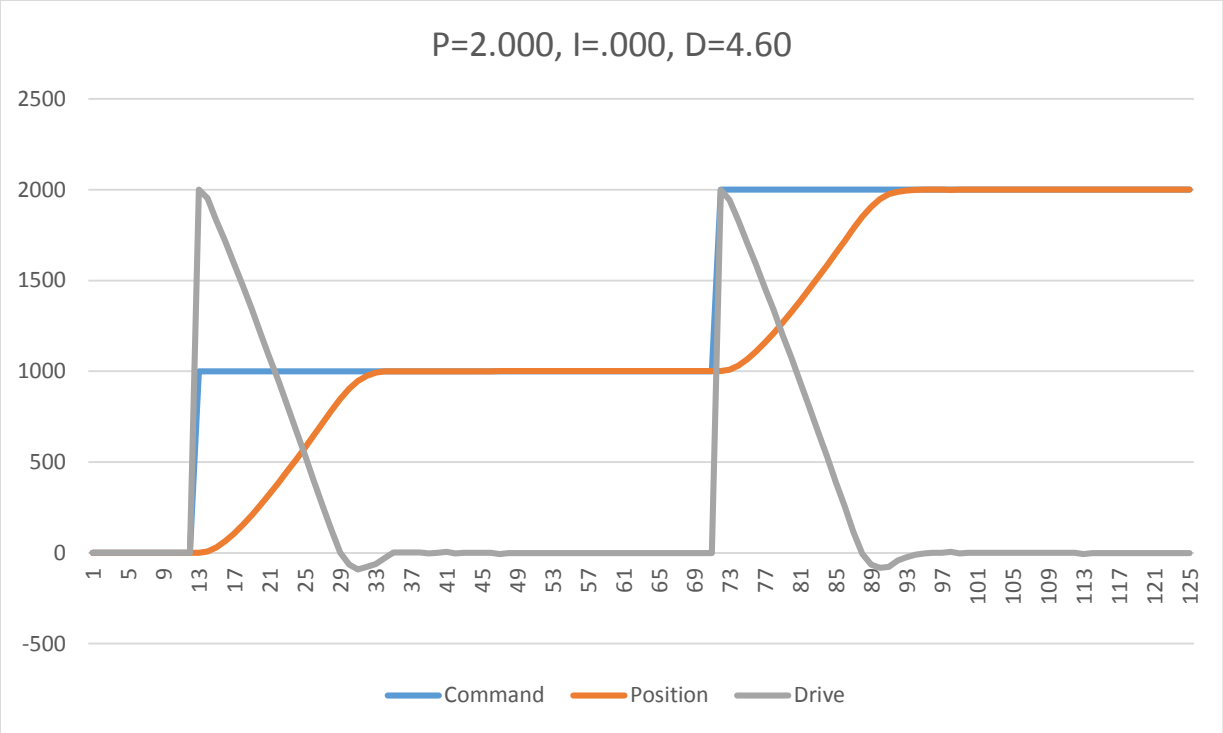
I then tweaked the gains a bit to try to reduce overshoot.





4. Optimize Position Controller & Implement Trajectory Interpolator

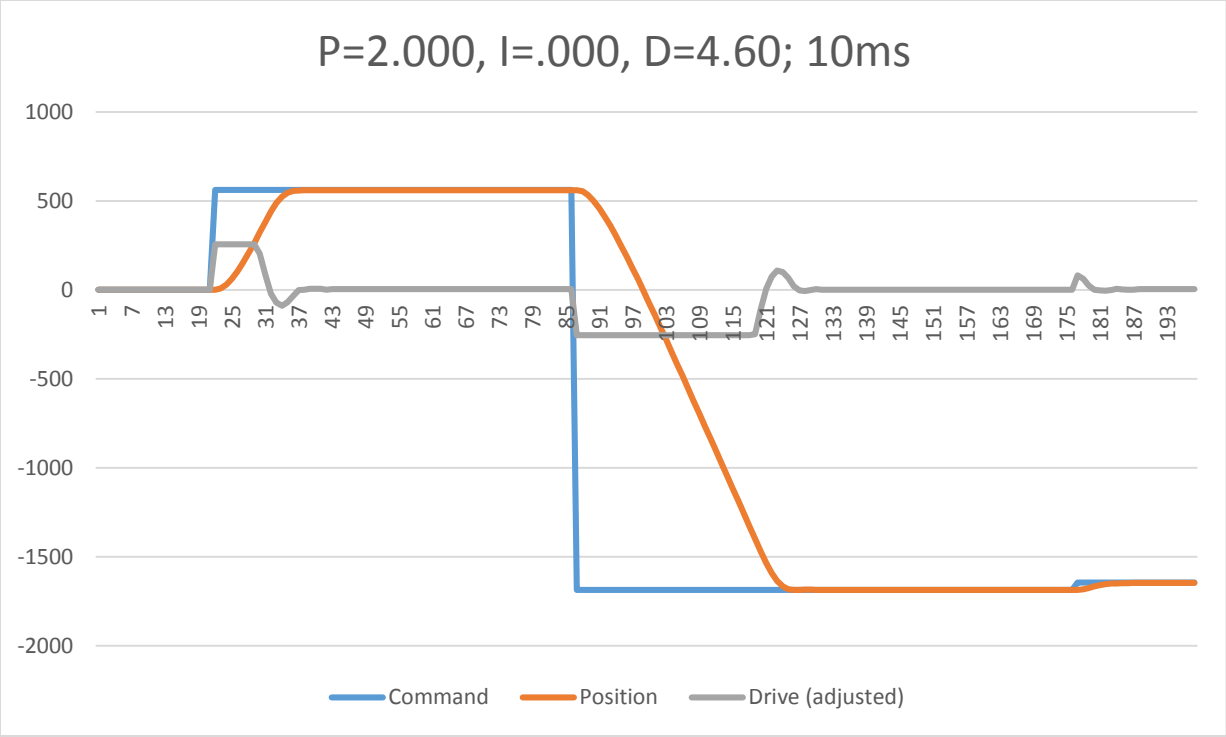
I experimented with values to optimize the position controller. I first reduced the proportional term so that it still got the motor up to full speed but did not create an unnecessary amplification effect on the oscillation after reaching the command value. This took me to P=2.000. Next, I increased the differential term to dampen the oscillation. I found that values higher than 4.6 did not decrease the overshoot and that values less than 4.6 did increase the overshoot, thus 4.6 was the optimal differential term dampener. The performance of the PD controller was so good that I did not add in any integral term gain. I kept the PD controller loop running at a period of 10ms. The following chart shows the result.



For the trajectory interpolator, I hard coded a series of reference positions (in counts). Given that the encoder produces 2248 counts per revolution of the output shaft, I computed the following values for the trajectories:

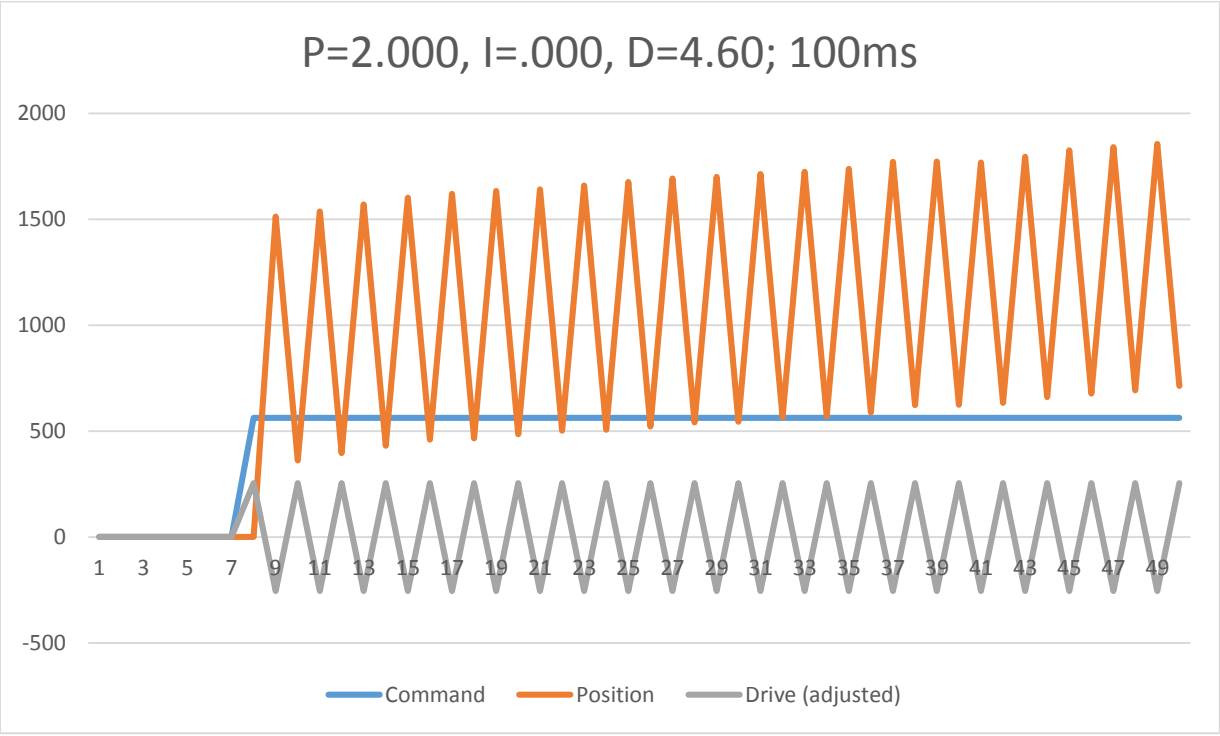
Trajectory #	Degrees	Degrees as Counts	Relative Reference Position Calculation (in counts)	Relative Reference Position Value (in counts)
1	+90	+562	(start)+562	(start)+562
2	-360	-2248	(start)+562-2248	(start)-1686
3	+5	+31	(start)+562-2248+31	(start)-1655

Implementing this yields the following graph (each data point is at 10ms).

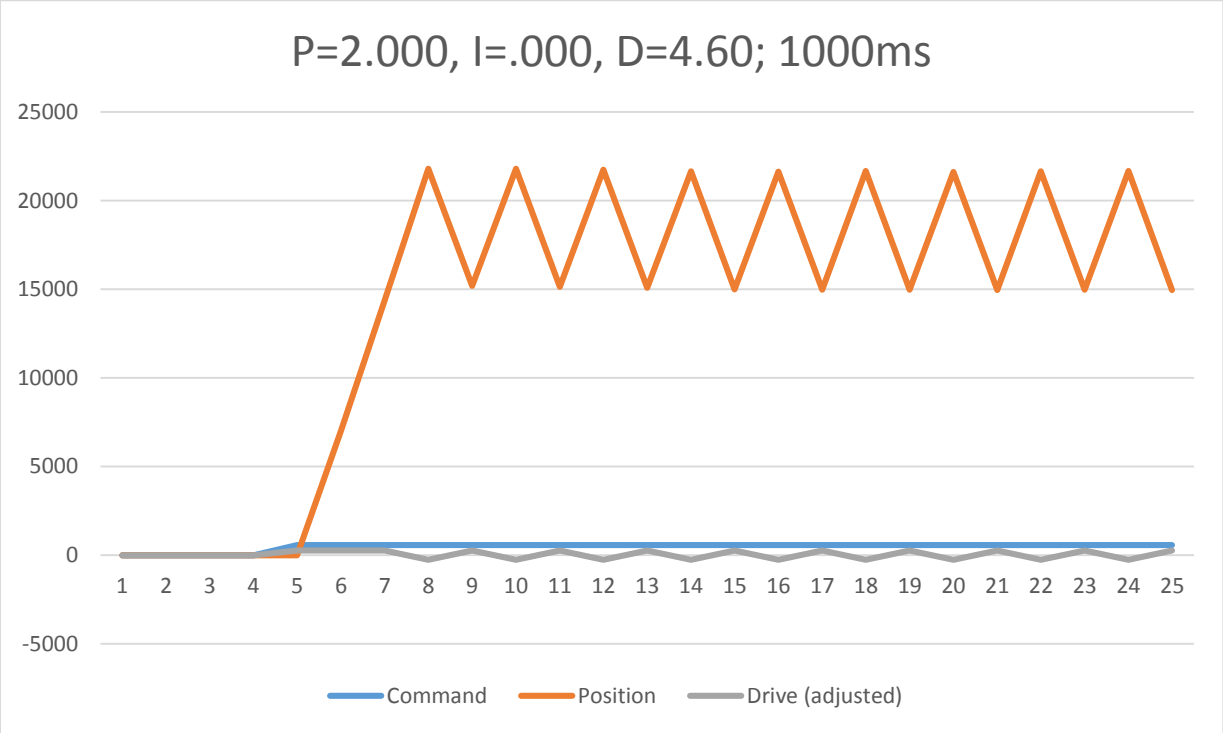


5. Trajectory Interpolator with Controller at Slow and Very Slow Rates

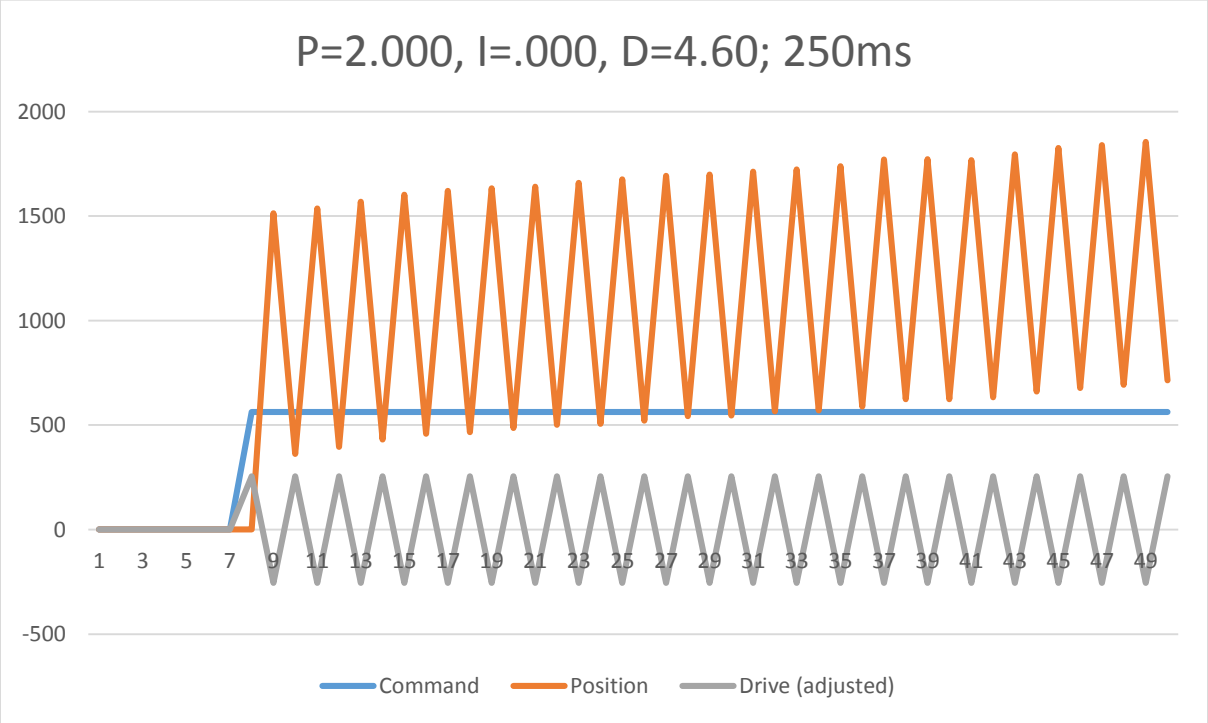
In the previous exercise, the controller was run at 10ms. When running at a "slow" rate of 100ms the following graph results.



When running at a “very slow” rate of 1000ms the following graph results.



Just for fun, I also tried it out at 250ms.



All of the slower controller periods (100ms, 250ms, 1000ms) exhibited similar unstable system results. The system never settled on its reference position. I believe this is because our proportional gain is much too high for these longer periods.

When the proportional gain is high, the motor moves quickly. With the slower controller periods, the controller does not have a chance to stop the motor at the reference position and, thus, a huge overshoot is experienced. This overshoot is then then followed by an undershoot and so on and so forth.

The proportional gain value of 2.0 was tuned at a 10ms controller period. Going to a 100ms period is a full order of magnitude upward change, which should probably result in [at least] a full order of magnitude downward change (i.e., 2.0 -> 0.2) in the proportional gain to slow the rate of change enough to let the system stabilize.