Dr. Axel Krieger

EN.530.666

Sai Avva, James Kaluna, & Diana Shaughnessy

Lab 3

Observation of P controller (10 points)

The students' first iteration of trials and tuning was with their P controller. After developing an algorithm to detect their desired position, current position, and error, the students were able to create conditional statements that would turn on the necessary coils to "pull" the magnet towards the desired position. High Kp gain for the P controller resulted in high instability and serious overshooting problems. Reducing the Kp gain reduced the overshoot but increased steady-state error.

Observation of PD controller (10 points)

After fine tuning the P controller, the team moved on to developing and tuning their PD controller. The students first started by initializing the previous error ... more words... We used the critical damping value of $K_d = 2\sqrt{K_p}$ as a starting point for the derivative gain. The PD controller performance before tuning was a lot of oscillation around the target area, symptomatic of instability. To fine tune this, the students first started by scaling the K_d value to be lower and lower until it reached a performance the students were satisfied with. When comparing the PD controller to the P controller, the robot was able to reach its target position in a much smoother, unified fashion, with minimal overshoot and more stability, i.e., minimal oscillation.

Observation of PI controller (10 points)

The students then moved on to the PI controller. The students first ran the PI controller without fine tuning to see its default performance. The robot moved in a well performing manner at its base. However, the team decided to investigate how minimizing the K_i would alter the performance. The result was an even better performance despite the original trial still being an appropriate time response.

Observation of PID controller (10 points)

Finally, the team decided to test their PID controller using the tuning they had from the P, PD, and PI controller. By running the PID controller with their previous tunning, the robot's performance was more erratic than expected, it overshot more than anticipated. The students, due to their overshooting, decided to alter their K_p first. This resulted in a much smoother execution to the target, but the students believed they could improve. Afterwards, the students chose to alter their K_i as well. Their end performance allowed the PID controller to reach the target nearly seamlessly.

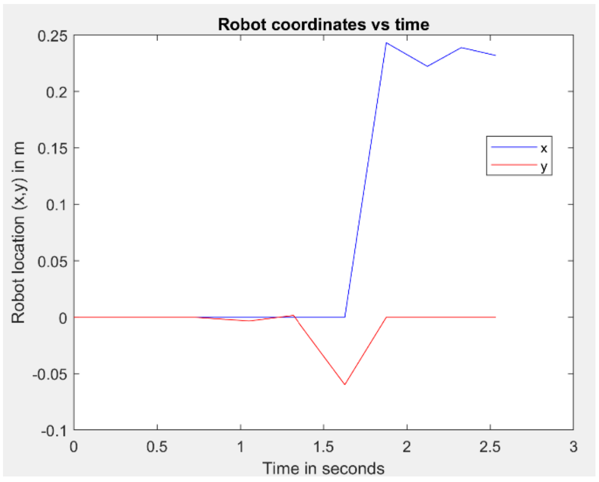
Does your control performance vary with desired position? Is your PID controller designed to address differences in starting and desired positions? (10 points)

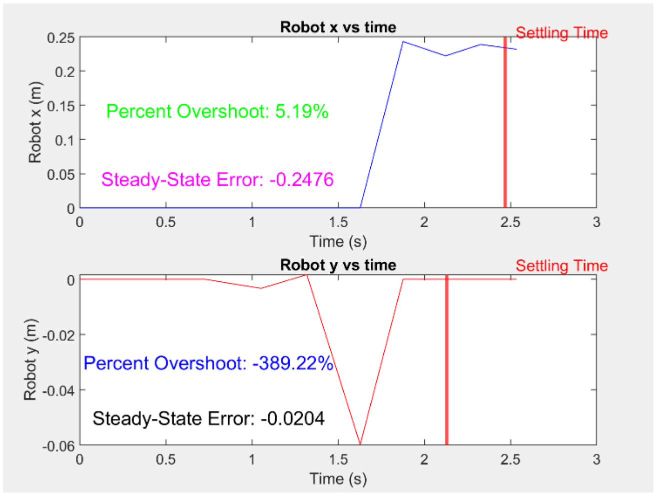
The control performance does not vary with desired position. The coils may activate at a greater intensity initially depending on how far the robot is from its desired position. However, in terms of performance, the algorithm that the students developed to get the robot to the desired position does not vary per the selected desired position. The students' PID controller is designed to address differences in starting and desired positions as it is taking the error between those two positions to initialize the gains that will be implemented in the PID controller.

Comparison of different controllers (10 points)

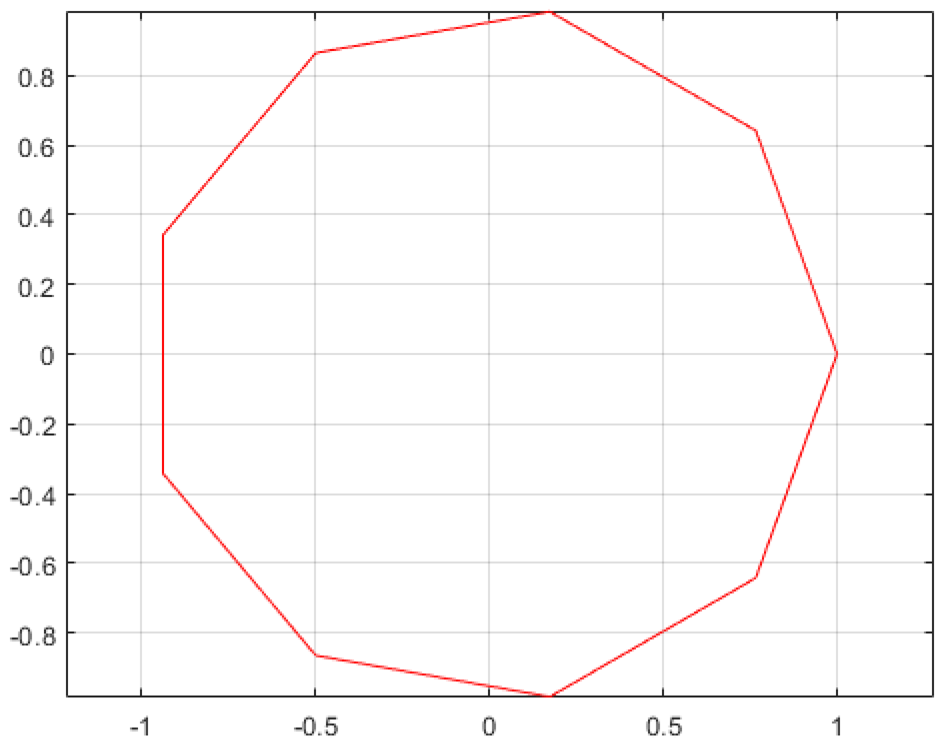
The various controllers allowed the students to test and observe the performance of the robot when certain dependencies are activated, being K_p, K_d, and K_i, in different combinations. The controller that had the most difficulty getting the robot to the target position was certainly the P controller, even after fine tuning the gain error. In contrast, the PID controller performed the best as it was able to direct and pull the robot to the correct desired position with no overshoot and in a smooth fashion. The students did mess with the activated gain errors for each controller type, with PID coming out on top as the best performing.

Error and control time analysis (20 points)





For 10 equidistant points, we utilized the origin as the center and generated 10 points with a chosen radius similar to the following plot.



The code in *TenpointEstimation.m* would find the center of the petri dish and move the robot to origin and then move the robot to the defined point. It would repeat this for all 10 points and saves the workspaces and video for each iteration.

Extracting the data using *Keyframingusingimtool.m* we find that the max error in x is 3.6% and while the max error in y is 1.7. While the standard deviation in errorx and error in y is 1.49 and 1.4 respectively.