Closed Loop Control

Zohaib Khan, Electrical Engineering Undergraduate Student
Department of Electrical Engineering, University of California, Riverside, CA
zkhan003@ucr.edu

I. Problem Statement

With the emergence of robotics in the market, engineers sought to automate these robots to increase the accuracy of their task. Due to real life parameters and software, the system can vary different results compared to the desired results. Last lab, we worked on an open loop system where the Turtlebot follows a square trajectory. This lab, the Turtle will follow the same pattern, however this time will be in a closed loop where a PID controller will be implemented to increase the accuracy of the desired results and compensate for the real life parameters that affect the system. Moreover, this report will compare and contrast the performance of an open loop systems vs a closed loop system.

II . Intro to PID Controls

Pid uses three different controls to apply on a system to increase the accuracy and overall performance. P stands for proportional where it uses a gain factor of ‘K’. Moreover, I accounts for past values and integrates each other. Once you add the proportional gain to the I, it allows the system to reduce the error. Lastly, D takes account of future values to be able to compensate for the error that is expected. The visual representation of a PID control can be seen below in figure 1.

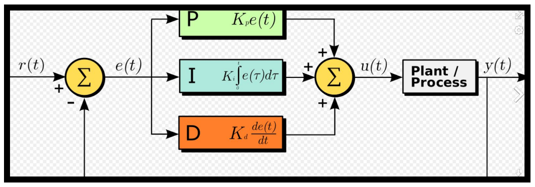


Fig.1. *Pid Controller*

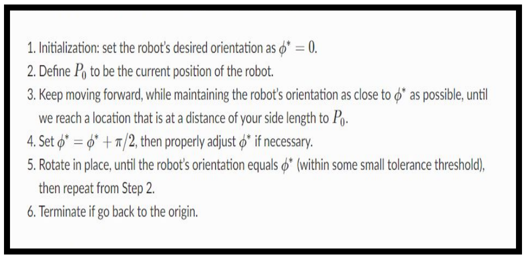
III. Approach and Description

A. High Level Description

The approach to solve this problem is to understand what a PID controller is and what components make up a PID controller. Secondly, how to set what variable with the appropriately values. The equation, of Kp * error is the equation that must be implemented to create a P controller. Kp is a dynamic value where as error, is the system set angular minus current angular. The derivative term can be expressed as u(t) = Kp(e(t)) + Kd(de(t)/dt)). The code initializes the desired orientation to be zero. Once the bot travels the appropriate position, it will properly adjust theta with Pi/2. From there, the Turtlebot will follow those sets until it reaches the origin and stops.

B. Low Level Description

The low level design and approach to this system follows the pseudo code below. For the P implementation, there is conditions loops where the counter reaches will then result in an action. The first turn theta is set to be 1.57 because Turtle bot goes from pi to -pi. Then subtract setpoint with initial point and multiply with Kp which is the new angular velocity. The next turn were set as 3.14, -1.57, and 0 respectively. For the PD implementation, very similar to the P controller however this time we have subtract the error with previous error and multiple with the Kd value. All of this will be added to the P controller which thus will be put in the angular.z function.



```
1. Initialization: set the robot's desired orientation as $\phi^* = 0$.
2. Define $P_0$ to be the current position of the robot.
3. Keep moving forward, while maintaining the robot's orientation as close to $\phi^*$ as possible, until
   we reach a location that is at a distance of your side length to $P_0$.
4. Set $\phi^* = \phi^* + \pi/2$, then properly adjust $\phi^*$ if necessary.
5. Rotate in place, until the robot's orientation equals $\phi^*$ (within some small tolerance threshold),
   then repeat from Step 2.
6. Terminate if go back to the origin.
```

Fig. 2. *Pseudo Code of the System*

.

IV. Results and Data Analysis

A. Implement P controller and discuss the results of different values of Kp

The implementation of the P controller can be seen in the code and also follows u(t) = Kp(e(t)). Kp is the gain terms. Having higher values of Kp allows the bot to have an overshoot which can be seen below in figure 3. Similarly, having a small Kp value will also results in the Turtle bot missing the origin. Therefore, through trial and error, the valid kp value would be around 0.6-0.8. This is because u(t) = 0.7* (1.57 - 0) = 1.099, which is close to the value of one that u(t) was designed to be.

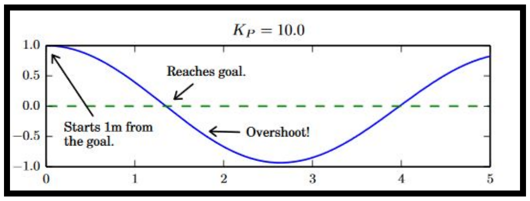


***Fig. 3**. Large Kp Overshoot*

B. Implement PD controller and discuss the results of different values of Kp and Kd.

The implementation of the PD controller can be seen in the code and also follows u(t) = Kp(e(t)) + Kd(de(t)/dt)). Where de(t)/dt is the change in error. If the error is decreasing the term would be negative and if the error is increasing the term would be positive. Furthermore, the bigger Kd value gives the system more to compensate for the error which can be represented in the figure below. If it is too big and too small it will deviate from the square resulting in more error, therefore it is every important to tune the value to work with the system.

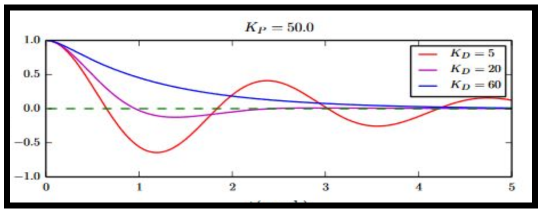


***Fig. 4**. Different Values of Kd*

C. Plot the trajectory of your robot with matplotlib.

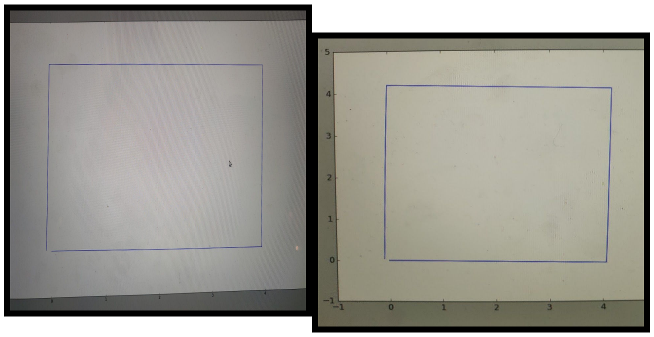


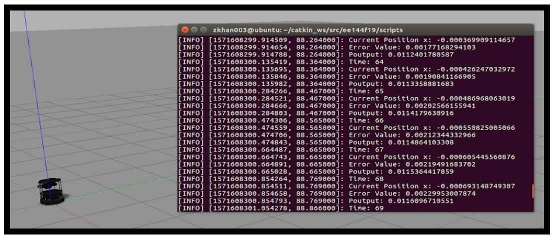
***Fig. 5**. MatLibPlot*

V. Discussion of Error

The difficulties of this lab was every time the user ran their script, it would result in the Turtlebot doing some type of different path. It was soon figured out that the odometry does not reset unless you close Gazebo or you manually reset it in terminally. The other problem was that the orientation of the Gazeb uses (Pi to -Pi). With these discoveries, the lab was completed.

VI. Conclusion

In summary, the significance of this lab was to understand how a PID controls affects a system. Moreover, this lab expresses how to create a PD and P controller and compare the results between them. The experiments verify that the accuracy of a PD controller is much better than just a P controller because there is an error that occurs between two points when P is used alone.Furthermore, the lab thought how to plot the trajectory using matlibplot



```
zkhan003@ubuntu: ~/catkin_ws/src/ee144f19/scripts
[INFO] [1571608299.914509, 88.264000]: Current Position x: -0.000369909114657
[INFO] [1571608299.914654, 88.264000]: Error Value: 0.00177168294103
[INFO] [1571608299.914788, 88.264000]: Poutput: 0.0112401780587
[INFO] [1571608300.135419, 88.364000]: Time: 64
[INFO] [1571608300.135695, 88.364000]: Current Position x: -0.000426247032972
[INFO] [1571608300.135846, 88.364000]: Error Value: 0.00190841166905
[INFO] [1571608300.135982, 88.364000]: Poutput: 0.0113358881683
[INFO] [1571608300.284266, 88.467000]: Time: 65
[INFO] [1571608300.284521, 88.467000]: Current Position x: -0.000486968063019
[INFO] [1571608300.284666, 88.467000]: Error Value: 0.00202566155941
[INFO] [1571608300.284803, 88.467000]: Poutput: 0.0114179630916
[INFO] [1571608300.474306, 88.565000]: Time: 66
[INFO] [1571608300.474559, 88.565000]: Current Position x: -0.000550825005066
[INFO] [1571608300.474706, 88.565000]: Error Value: 0.00212344332966
[INFO] [1571608300.474843, 88.565000]: Poutput: 0.0114864103308
[INFO] [1571608300.664487, 88.665000]: Time: 67
[INFO] [1571608300.664743, 88.665000]: Current Position x: -0.000605445560876
[INFO] [1571608300.664891, 88.665000]: Error Value: 0.00219491683702
[INFO] [1571608300.665028, 88.665000]: Poutput: 0.0115364417859
[INFO] [1571608300.854264, 88.769000]: Time: 68
[INFO] [1571608300.854511, 88.769000]: Current Position x: -0.000693148749307
[INFO] [1571608300.854658, 88.769000]: Error Value: 0.00229953007874
[INFO] [1571608300.854793, 88.769000]: Poutput: 0.0116096710551
[INFO] [1571608301.054278, 88.866000]: Time: 69
```

***Fig. 6**. Turtlebot on Square Trajectory*