ELECTRICAL & COMPUTER ENGINEERING
UNIVERSITY of WASHINGTON

Assignment 3

Team-7

EE P 545 A - The Self Driving Car: Introduction to AI for Mobile Robots

November 15, 2023

Line Following with PID Control (40 points)

Q1. What exactly was your error metric? How exactly did you choose which point the robot should navigate towards?

A1. The error metrics that we used are the Cross-track error (e_{ct}), Along-track error (e_{at}) and Heading error (θ_e). We utilized e_{at} to judge whether the points in the plan are behind or in-front of the robot. We used e_{ct} for the translation error and $(\theta_e - \theta_r)$ as the rotational error in the code. Where (θ_r) is the orientation of the robot wrt. to the global frame of reference.

The net error is :

$$E_{net} = translation\_weight * translation\_error + rotation\_weight * rotation\_error$$

$$E_{net} = translation\_weight * e_{ct} + rotation\_weight * (\theta_e - \theta_r)$$

Here, E_{net} is used as the error metric for determining the steering angle for publishing the control command to the robot.

So, the steering angle (δ) using the K_p, K_i, K_d controller gains is calculated as follows:

$$\delta = K_p * E_{net} + K_i * \int E_{net} \cdot dt + K_d * \frac{dE_{net}}{dt}$$

Q2. What are the 3 sets of PID parameters that you choose? Which set of PID parameters performed best? Why do you think this is the case?

A2. We used the following set of PID parameters:

- Set - 1 : $K_p = 1.5, K_i = 1.5, K_d = 3$
- Set - 2 : $K_p = 3, K_i = 2, K_d = 1$
- Set - 3 : $K_p = 1, K_i = 3, K_d = 2$

According to the error plot given in Q4, we see that set 2 performed best, it was able to join the trajectory sooner than the others. This is due to the reason that K_p is relatively higher and K_p focuses on the immediate response to the current error. So, the error at the start of the simulation where the robot tries to reach the plan, the higher K_p values makes it reach the plan faster in the starting stage and once when it reaches the plan using the values of K_p, K_i, and K_d the robot follows the plan and reaches the end point goal efficiently, owing to K_i, which takes into consideration, the past error values, and K_d which does a good job ensuring less future error values.

Q3. What is missing that is stopping us from implementing this on the real robot? (Hint: given a trajectory in the real world, think about what the robot needs to know in order to calculate the error for PID)

A3. The bag file provided to us to work with will not correspond the path in the real world, and the odds of the muSHR bot successfully moving in an intended direction (which again we have no way of communicating to the bot) is so bad. The real-time response factor is missing this exercise when we want to apply the same logic to an unconstrained real-world setting.

Moreover, the goal poses that need to be mimicked by the bot are not known to the bot, that gap in data inflow will restricts the muSHR bot with this implementation of the PID control logic from navigating the real-world path correctly.

Q4. The error plot described at the end of Section 2. You can refer to the plots generated from assignment 1 to generate this plot.

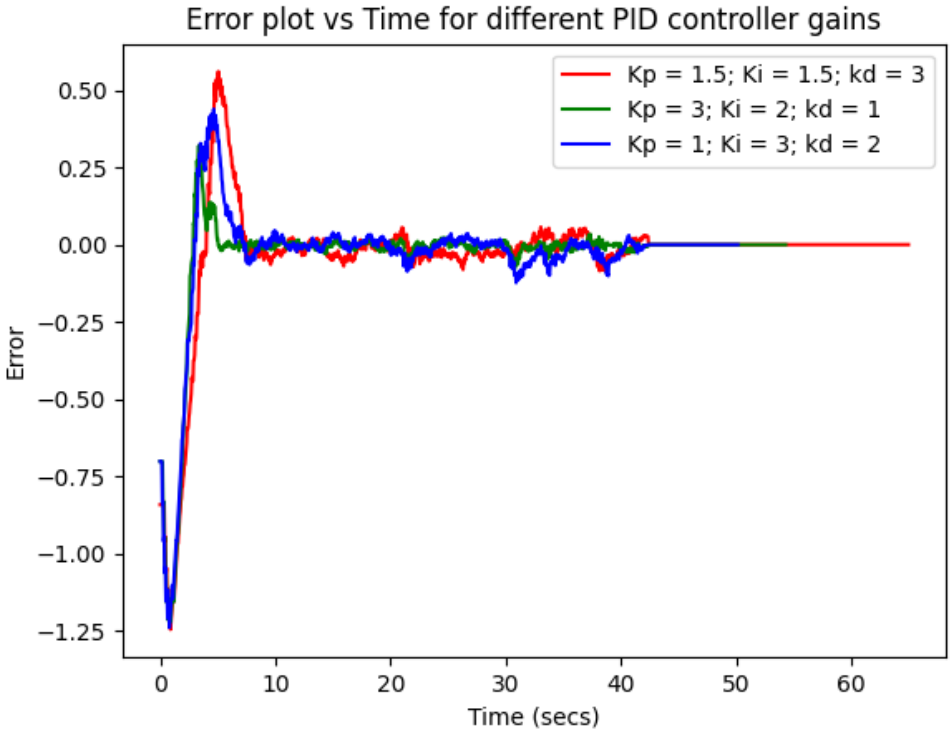

Figure 1: Error vs Time plot for different PID controller gains.

A4.

Q5. A video of your best PID controller following the path generated by the provided bag file `lab3/bags/line_follower.bag`. You should use the small basement map for this video.

A5. Please play the `PID_sim.mp4` file in the github repository. Or, Click the video link in the `Readme` section of the repository.

Wandering Around with MPC (60 points)

Q6. Describe the parameter values that you choose (e.g. T, delta_incr, etc.). Describe any other design choices that you made.

A6. We chose :

- `max_delta` $= -0.341$: is the minimum angle which the robot will take into consideration for forming the rollout trajectories.
- `min_delta` $= 0.34$: is the maximum angle which the robot will take into consideration for forming the rollout trajectories.
- `delta_incr` $= 0.1133$: is the subsequent angle interval which generates the number of different steering angle trajectories using `max_delta`, `min_delta` & `delta_incr`.
- `dt` $= 0.01$: is the time interval which decides the subsequent poses for the robot using the Ackermann Kinematic car model. Smaller the value of `dt`, smoother the car will move.
- `T` $= 300$: is the number of poses in a rollout.
- `laser_offset` $= 2.0$: is the offset which determines how far the robot should be before an obstacle is detected by the robot.

In code :

We tried varying the `min_delta` & `max_delta` which would increase the steering region , so that the robot can turn efficiently.

We also experimented with different values of `delta_incr` giving us different number of rollout tracjectories for the robot to choose.

We used different values of `T` from 100 to 400, to see how the robot behaves.

In the robot :

We also tuned the `speed_to_erpm_gain`, to make sure the distance that the code discerns is what the bot physically are the same.

We tuned the `steering_angle_to_servo_gain` to make sure the turn radius is consistent with what is expected from the car taking its dimensions into consideration.

We tuned the `steering_angle_to_servo_offset` to make sure the steering angle is not haphazard and steady enough to perform the expected journey intended by the controller.

With the following parameter set, our MPC model was able to traverse the `no-end-floor4_corridor` map in the simulation and ECE basement in the real world easily.

Q7. A video of your robot continuously wandering around the `no-end-floor4_corridor` map in simulation model for at least one minute when initialized by the provided bag. Make sure the robot's laser scan and the visualization you implemented are visible.

A7. Please play the `MPC_sim.mp4` file in the github repository.

Q8. A video of your robot continuously wandering around the basement of the ECE building (Level 0) for at least 30 seconds. You can pick an initialization point for the robot. We provided a map called `small_basement` that you can use to practice before using the real robot.

A8. Please play the `MPC_real.mp4` file in the github repository.