Writeup

(a) PID Controller

i. Document the tuning process for your controller.

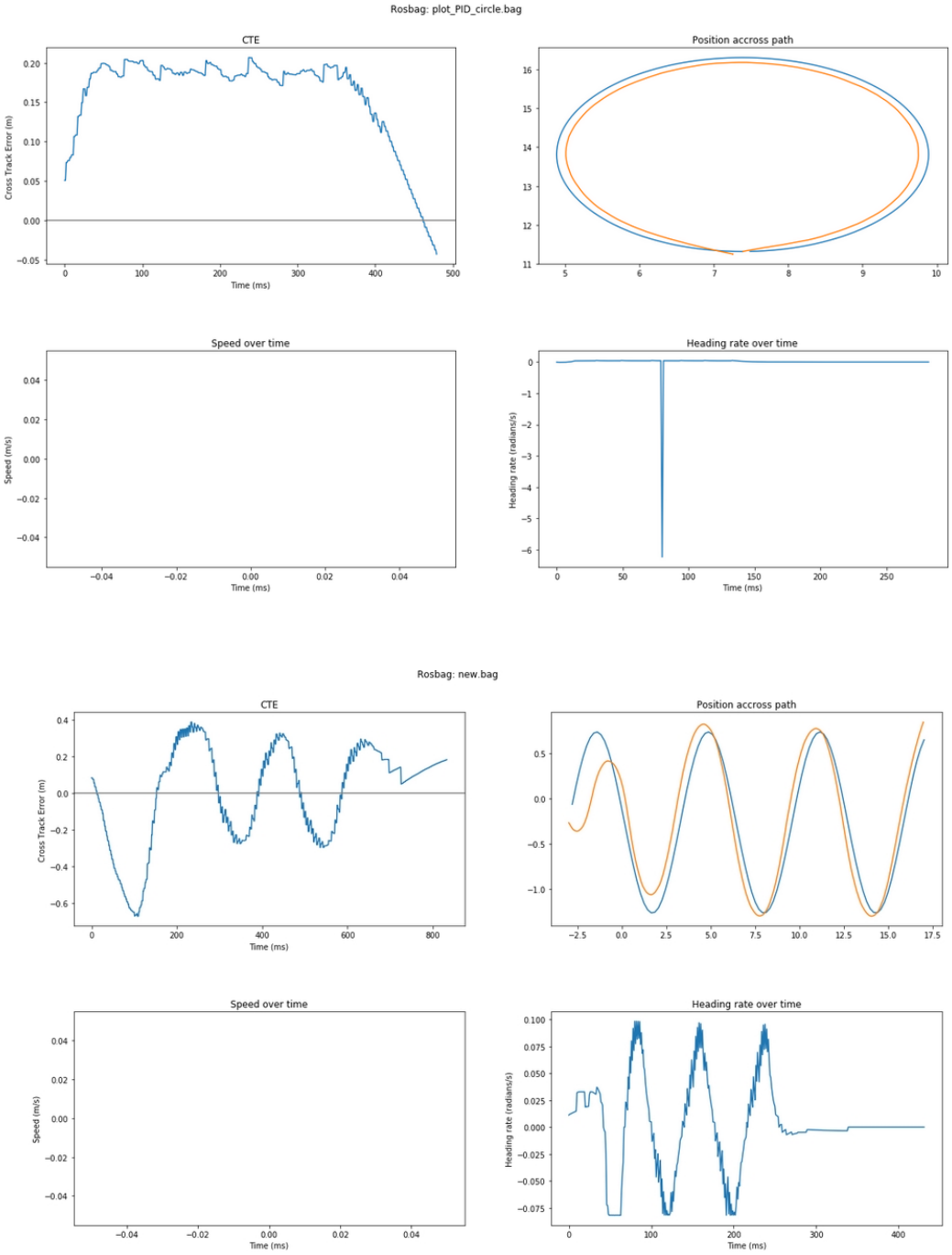
a. What metrics did you use to evaluate the quality of the controller?
We considered how fast it converged to the trajectory if initialized far away from it, and how much overshoot we saw while following the trajectory. If it does overshoot, we also consider how fast it corrects itself, and what the error was like at the end.

b. What was its performance profile before tuning?
Using left turn as an example, when following the straight line there is no car offset from the line, but after attempting to turn, the car overshoots and the offset from the line increases. For the circle trajectory, the car has a small constant radial offset.
When it was initialized further from the trajectory, it converged very slowly to the trajectory. In terms of correcting itself, the car is slow to correct its self. The error at the end of each trajectory is usually pretty high.

c. How much did these metrics improve from tuning it?
If initialized further from the trajectory, convergence speed increases, overshooting decreases, and final error is smaller. This applies to all of the given trajectories.

d. What tradeoffs did varying the gains Kp and Kd have on the controllers performance?
We've observed that Increasing kd tends to make the car undershoot when turning with the trajectory, whereas increasing kp makes the car overshoot.
With high kd and low kp, the car's convergence onto the trajectory is much slower, with high kp and low kd, there is clear overshooting when making turns.

e. What Kp and Kd values did you select for your controller? Substantiate your answers with plots and tables where necessary. We encourage you to use and modify Controller Plotting.ipynb as needed.
Final values are: .50 for lookahead distance. .45 for kd, and .30 for kp.
We got the following results with these values:



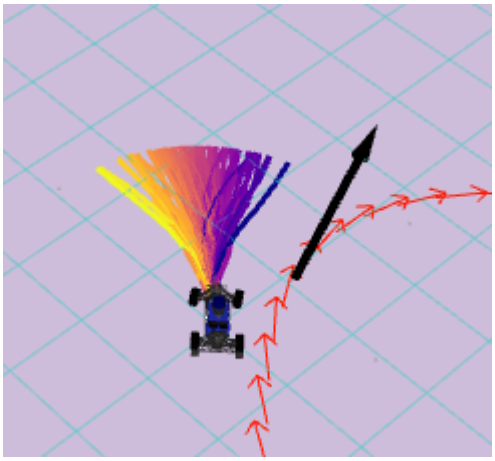
ii. Compare the two ways of computing de(t)/dt . Which do you prefer and why?

Analytic derivative performs much better than finite difference derivative. Finite difference overshoots dramatically at every turn and fails to converge properly. Finite derivative method also accumulates over time, since we do not have an integral controller this was problematic. With all that, our group is in agreement that Analytic derivative is better.

(b) Model Predictive Control

i. Trajectory generation

a. Plot the trajectories generated by your Model Predictive Controller using the kinematic car model. Your result should resemble Figure 3. We provide a helper function viz_paths_cmap in rosviz.py that you can use.



b. What are the number of rollouts (K) and number of time steps (T) you selected? Document your process for finding these parameters.

We started off 62 rollouts and 8 time steps. There seems to be no issue with processing this many rollouts with 8 time steps each, and going beyond this worsened car performance and slowed down the computational speed of the program. Therefore we decided to keep the values we started with.

ii. Cost function

a. What is your cost function?

i. Sum of collisions plus total distance of trajectory plus distance between the final point of our trajectory and our target waypoint.

b. How did you find your cost function? What worked and what didn't?

i. We wanted to penalize the number of collisions, length of trajectory and distance between endpoint of trajectory and desired waypoint. We did a weighted sum of these values to get a simple cost function and it worked right away.

ii. If the obstacle is very wide, the car could not go around it because it could not generate enough trajectories that go

around the obstacle, and attempting to go around exceeds the max distance to waypoint threshold. The cost function never checks if a trajectory is impossible to complete, such as in the case of the end point being in an obstacle.

c. Did you need to tune gains for different terms in your cost function? If so, document your process for tuning the gains.
    i. We incorporate tuning for collisions, distance from end, and total distance. We initially set every weight to 1, which seemed to cause quite a bit of overshooting and wobbling on straight lines, but obstacle avoidance seemed pretty good. Eventually we found that setting the values to 10 for collisions, 15 for end distance, and 5 for total distance worked well. These values seemed to reduce wobbling a bit, and also overshooting was not as bad, and obstacle avoidance is still good.

(d) Overall findings:

Systematically compare each controller and summarize your findings. Think about the trade offs between each controller, such as:

i. In which settings is each controller most advantageous?
    a. PID controller works very well when driving through clear, obstacle free environments. If sticking directly to the drawn out trajectory is the goal, PID is the best at that.
    b. MPC is good when going through environments with obstacles, especially when those obstacles obstruct the trajectory.

ii. Which controller is most robust?
    a. MPC. Considering obstacles seems much more important than staying close to the drawn out trajectory, and therefore is more robust.

iii. Which controller worked best in high speed regimes?
    a. At high velocity MPC would work best. MPC is much better at recovering from overshot turns than PID, and due to not having to rely on specific tuning of kp and kd. MPC is generally better at turning than PID when velocity varies, because MPC does not depend as strongly on its tuning parameters.

iv. How did sim-to-real transitions compare between controllers?
    We had to increase the lookahead distance for MPC a little bit to reduce wobbling along straight lines, but after doing that MPC seemed to work almost perfectly, turning very smoothly and driving fairly straight along

lines. PID controller was very good when following straight lines, but struggled when turning compared to MPC’s turning. Increasing lookahead for PID seemed to make it undershoot its turns when turning into the end point, but with lower lookahead it overshoots. Either way it seems PID struggles just a bit more in sim-to-real transitions compared to MPC. The main issue seems to just be turning into endpoints.