MECH 540A: PROJECT REPORT

WAYPOINT TRACKING WITH CARLA SIMULATOR

Participants:

Aashkaran Dhillon (#63377956)

Jonas Chianu (#31298391)

Vibhas Vinod Nampalliwar (#65113532)

1. Project Summary:

For this project, we acquire waypoints from Carla simulator by driving the vehicle around a track and storing the x-y coordinates and velocity data generated by the in-simulator sensors for the trajectory. This waypoint data is then used as an input reference for the PID and Pure Pursuit controllers that output steering angle and throttle data for the path tracking application. A snapshot of in-game data acquisition is shown in figure below.

Figure 1:In-game screenshot of Data Acquisition with autopilot & recording turned on.

The project files interact as shown in the figure 2. First, the waypoint_generator.py file sets up the simulator environment and initiates the acquisition of waypoint data via autopilot mode under specified conditions such as no traffic and always green traffic lights. The car is driven around a track and the sensor data for X-Y coord and velocity is stored in a text file.

This data is then read by the main.py file and the TargetCourse() class is used to extract the target waypoints at a look ahead distance, which is then fed to PID and Pure Pursuit controllers in the controller.py file. The PID controller outputs throttle for longitudinal controls while the Pure Pursuit controller outputs the steering angle values.

The controller values are fed back to the main.py file, which updates the state variables of the vehicle and performs live plotting of path tracking in action, which aids in testing the controller performance. Furthermore, plots of speed, tracking error, throttle and steering are generated via the Test.py class, which help to further the understanding of underlying problems and refinement of controller gains.

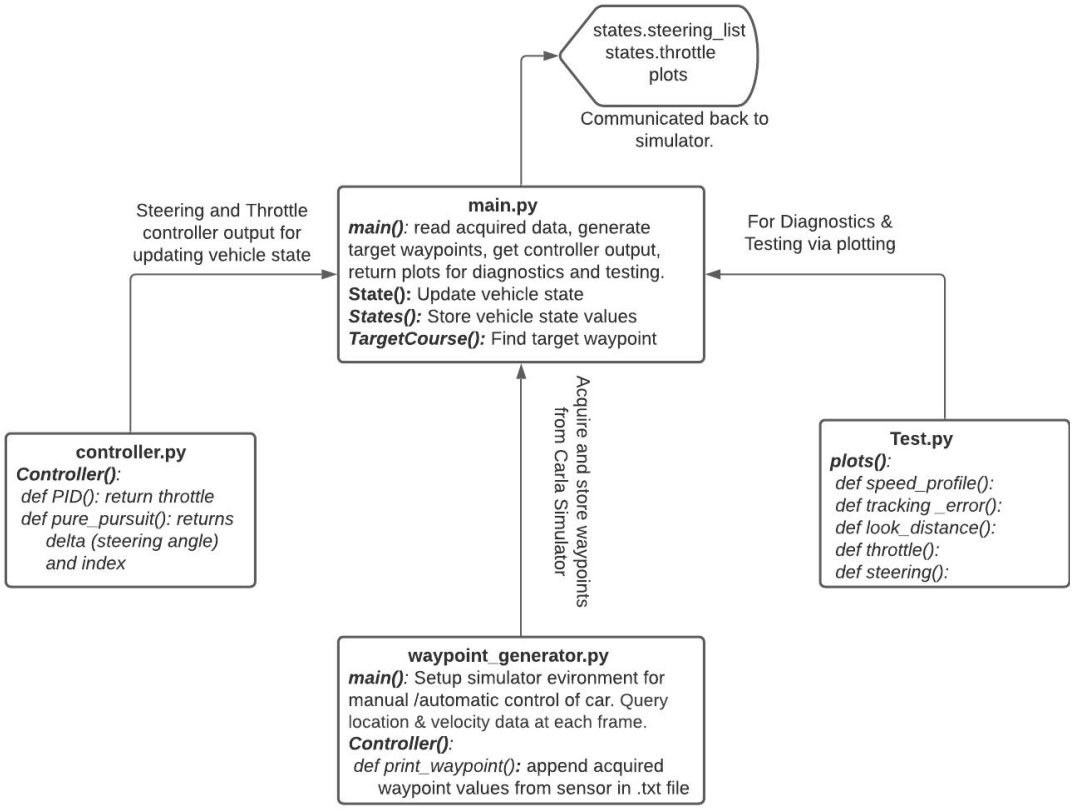

Figure 2: Flowchart of interaction with the main.py file.

2. Control Design:

There are two different controllers that are used, the PID and Pure Pursuit controllers. The PID controller is used to control the throttle and brake, while the Pure Pursuit controller is used to control the steering.

PID Controller

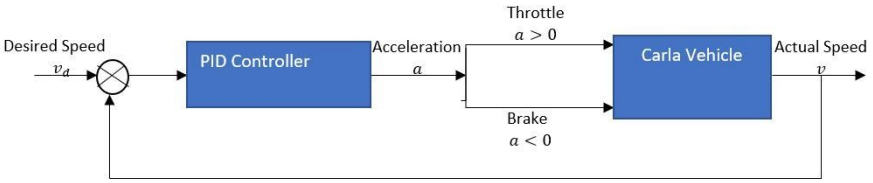

Figure 3: Controller block diagram

$$a = K_P(v_d - v) + K_I \int_0^t (v_d - v)dt + K_D \frac{d(v_d - v)}{dt}$$

Controller	Steady state error	Overshoot	Rise time	Settling time
K_p -Proportional gain	Decreases	Increases	Decrease	Small effects
K_I - Integral gain	Eliminates	Increases	Decrease	Increase
K_d - Derivative gain	Small effects	Decreases	Small effect	Decrease

Pure Pursuit Controller

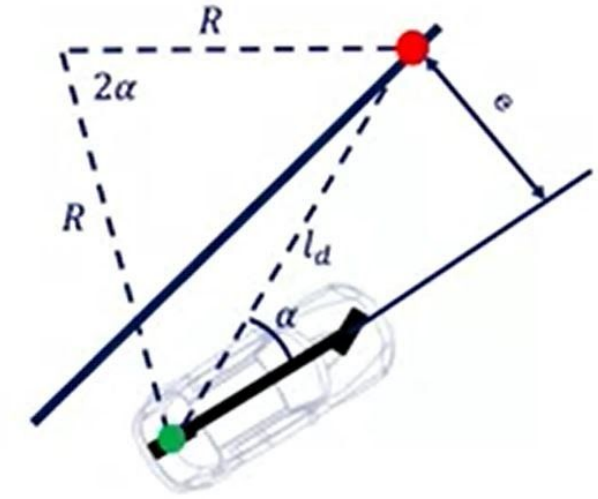

Figure 4: Geometric layout for bicycle model employed in pure pursuit

$$l_d = Kv$$

$$\delta = tan^{-1}(\frac{2\,WB\,sin\alpha}{l_d})$$

l_d : *Lookahead distance*

K : *Lookahead gain*

v : *Vehicle speed (forward velocity)*

δ : *Steering angle*

WB : *Rear length of vehicle*

α : *Angle between Rear length of vehicle and lookahead distance*

3. Capabilities:

The capability of this application is that it can perform waypoint tracking of the desired trajectory using steering angle and throttle output from the controllers, which are then used to update the state of the vehicle over time to reproduce tracking, as it would happen in real life application.

The path tracking performance of the vehicle is tested in real time for acquired waypoints using the live plotting as shown below. As seen, the controller does a great job of following the desired trajectory that was predefined in Carla Simulator *(waypoints_carla.txt)*.

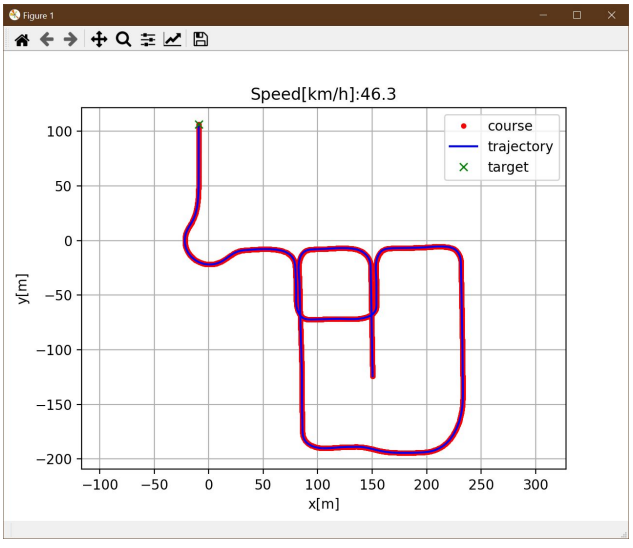

Figure 5: Waypoint tracking performance of the application.

Moreover, deeper insight on the performance of the controller can be observed from a variety of plots (Figure 6) that were obtained for testing, diagnostics and fine tuning of the controller gains. The Tracking error across X and Y axis are very acceptable at ~1.75m (mostly longitudinal from the vehicle chasing a target point), with the lateral tracking error being near to 0, which means the vehicle sticks to its predefined waypoints.

Although a work in progress (see waypoint_follower.py), these controller outputs (steering and throttle) can be communicated back to the Carla Simulator API to directly perform path tracking within the simulator environment and evaluate the performance for the desired trajectory. Owing to time constraints, we couldn't fix some errors produced for this, so we stick to real-time plotting for analyzing.

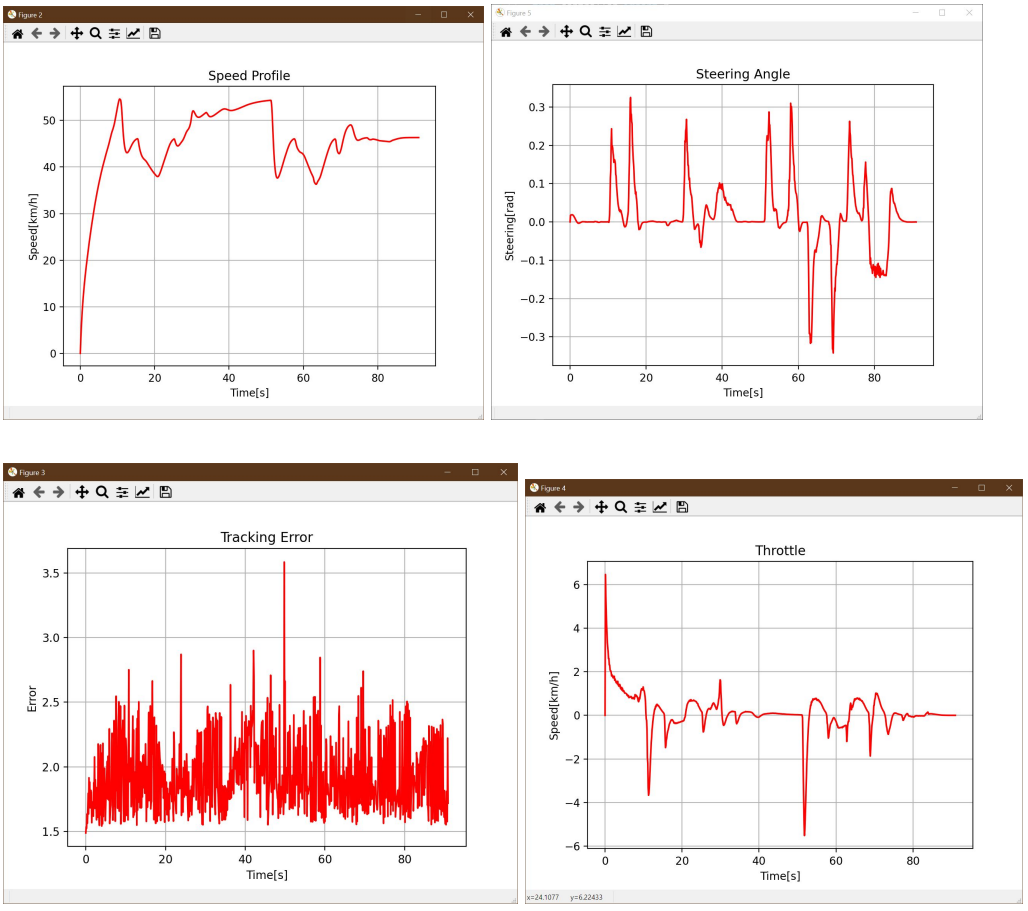

Figure 6: Plotting for testing, diagnostic and tuning of controllers.

4. Discussion :

In our proposal we defined the best case scenario, which was acquiring waypoint data and then talking back to Simulator API to recreate path tracking to see the controller performance. We also mentioned a more realistic objective, which involved plainly acquiring the waypoints from Carla simulator API and then graphically plotting to test for controller performance. In this project, we successfully achieved the latter, with the former still being in the work in progress stage due to shortage of time.

There is definitely a room for major improvement in terms of controller choice and the processing of waypoint data. One way we could have bettered our path tracking performance was by picking more advanced controllers like Stanley and MPC. We picked Pure Pursuit because it was relatively a simple one to understand and implement in the context of the term project. Secondly, we could have pre-processed the acquired waypoint data with methods such as interpolation before feeding to the controller. That would have resulted in less janky outputs as the waypoints are essentially acquired in discrete intervals and the interpolation would have smoothed it out.

Lastly, with everything in hindsight, we could have picked a relatively lesser challenging project, because this project required a lot of research and study of the existing controllers and various vehicle geometric models, and on top, familiarization with a very extensive API. Nonetheless, the project allowed us to push the envelope & go an extra mile.

5. Post-Mortem and Conclusion:

The project was a great opportunity to test our OOP knowledge which we obtained over the semester. Moreover, it provided us with the opportunity to utilize our programming skills in developing an application that helps solve a controls engineering oriented problem relevant to our major i.e. Mechatronics Design.

Not only, did the project help us consolidate our programming skills, but we also learned how to collaborate on a project as a team using git. Most importantly, we also got great practice in the formatting and readability of the code, which is a major and yet underrated part of software development.

This project motivates us to continue with some of the unaccomplished tasks such as talking back to the Simulator API to directly run from controller output. Nevertheless, whatever we achieved through the project can be called a clear success based on what we aspired for in the proposal.

6. Appendix:

Class hierarchy are presented below for the application.

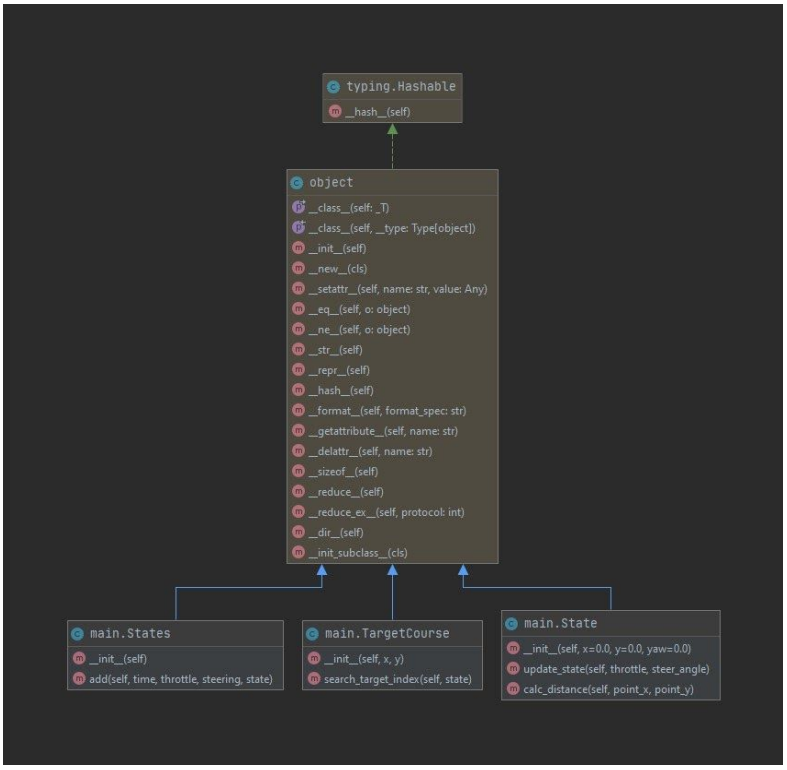

Figure A.1: main.py

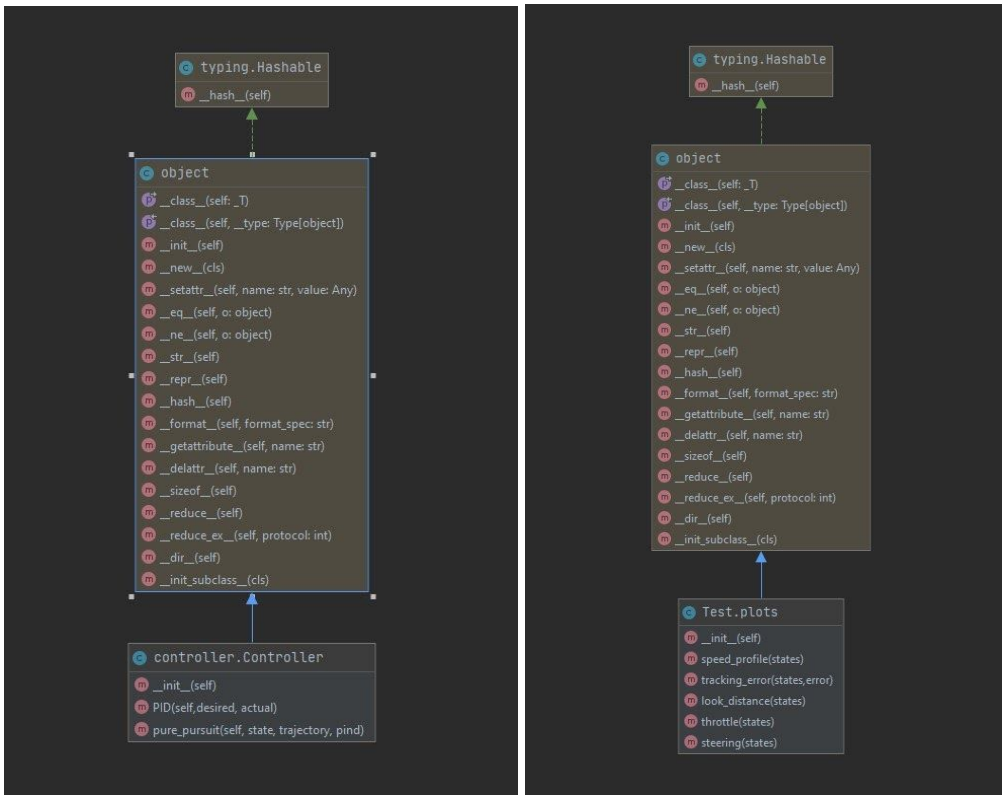

Figure A.2 & A.3: controller.py and test.py

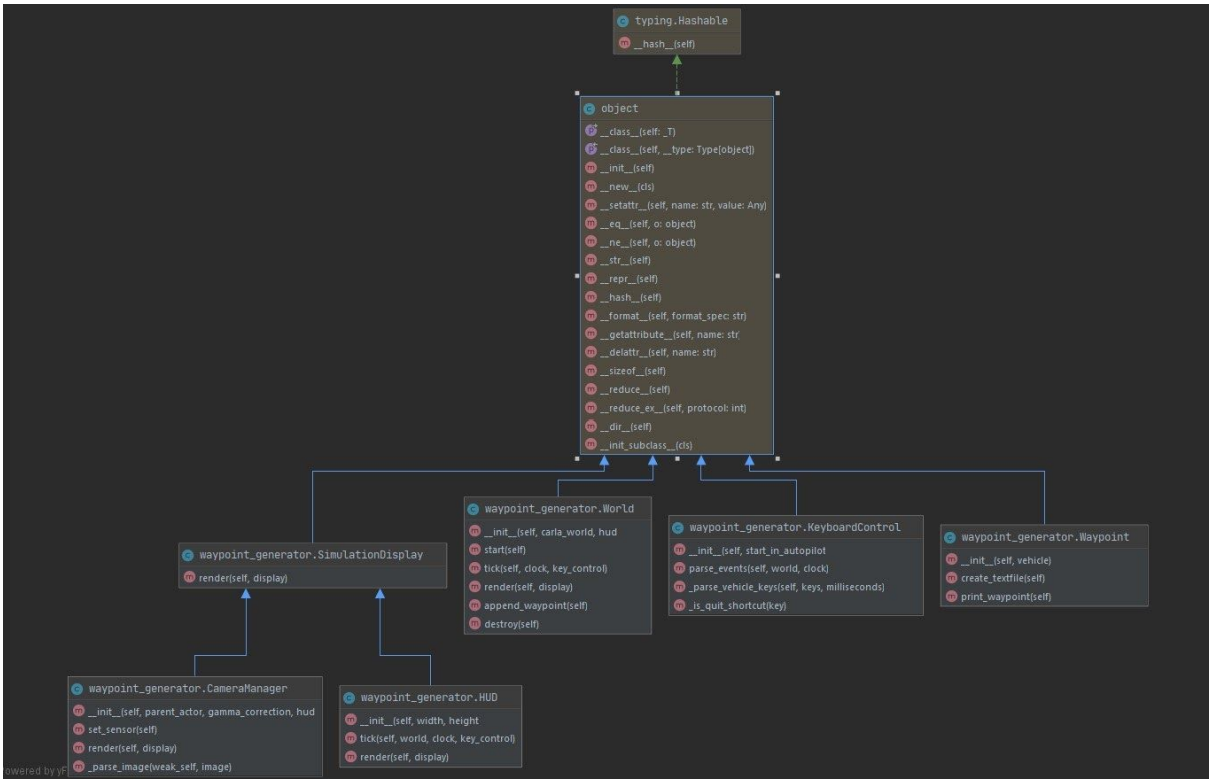

Figure A.4: waypoint_generator.py