Alaa Alajmy 201700095
Moemen Gaafar 201700873

Friday, May 1, 2020

CIE 318: Control Systems

Simulation-based Project Documentation

In this project, we aim to design a PID controller to steer an autonomous car simulated on Carla on a pre-determined racetrack. We will use Matlab, Simulink, and Matlab's System Identification Toolbox to model the car into a suitable transfer function then design a PID controller using Simulink's Control System Toolbox that meets acceptable specifications on both the transfer function and the CARLA simulator.

Simulating a car:

We utilized the CARLA Simulator, using a release of Python 3.6, to mimic an autonomous vehicle that moves on an otherwise empty racetrack.

To do the simulation, we start a CARLA server with a fixed time-step of 30 frames per second, or 33 milliseconds per simulation frame. We then connected a python client that was provided by the course's instructors under the title "module_7.py". The python client simulates the car on a set of racetrack waypoints and uses a controller file (controller2d.py) to produce throttle, brake, and steer commands that alter the car's speed and yaw angle.

To test the simulated vehicle before proceeding with the PID design, we created a simple controller (we refer to it as Controller1) that produces:

- a brake command of zero.
- a throttle command of 0.5 till the car reaches a speed of 20 km/h.
- steer commands of either -0.5, 0, or 0.5 rad/sec depending on the relationship between the car's current yaw angle and its next desired yaw.

With this simple controller, the car was able to drive through the whole racetrack, but slightly unsteadily.

When completing this step, we noted that the provided python client only runs on older versions of CARLA that are not compatible with releases of python 3.7. We also noted that said client's liveplotter.py script requires version 3.0.0 of the matplotlib library, a deprecated version, and hence running the client might require that you downgrade your installed matplotlib package to version 3.0.0.

Identifying the system:

In this simulation, our plant is a simulated self-driving car that takes throttle and steer commands and outputs a change in speed or direction of motion.

Since, in this project, we are only concerned with controlling the car's lateral orientation, we determined that the car will accelerate to and run on a constant speed throughout the simulation. The plant we will be controlling is hence a vehicle that takes an input of steer commands (ranging from -1.22 rad per second to 1.22 rad/sec) and outputs a change in yaw in radians.

To pick the speed on which our car will run, we first surveyed all the speeds our python client enabled and found that the following throttle commands give the following constant speeds after a few seconds of application:

Throttle	**Speed (in km/h)**
0.4	11
0.5	16
0.6	22
0.7	31
0.8	43
0.9	72

We filtered out the speeds 22 km/h, 43 km/h, and 72 km/h as the most reasonable for our car. We then identified our system as it ran on these three speeds and picked for our simulation the speed that gives the most manageable transfer function model.

To identify our system, we applied five types of steer inputs to each of the speeds we picked:

1. **An Impulse Steer:** Applies a steer command of 0.1 rad/sec for 0.5 seconds at time = 15 seconds. The command is not applied at the beginning of the simulation to allow the car to first accelerate to the target speed.
2. **A Step Steer:** Applies a steer of 0.1 rad/sec from time = 15 seconds till the end of the simulation.
3. **A "Dance" Steer:** Applies a steer of 0.1 rad/sec for 0.5 seconds followed by a steer of -0.1 rad/sec for 0.5 seconds from time = 15 seconds till the end of the simulation.
4. **A "Controller2" Steer:** Applies one of 9 different steers depending on the magnitude and direction of the error between the desired yaw and the current yaw. We found using this second controller input to be necessary because system identification works more accurately when many magnitudes of steer commands are entered, and if these many commands were not orchestrated by the error, the car would collide too early in the simulation, preventing the collection of a sufficient amount of data.

5. **A "Controller3" Steer:** Works exactly like Controller2 but with different magnitudes of steer commands.

We entered 15 sets of steer inputs (each of the 5 steer types applied at each of the 3 speeds) and the corresponding resulting changes in yaw extracted from CARLA into Matlab. We then used the System Identification Toolbox to find the best-fitting transfer function model for each speed. We used the "Controller2" input to estimate discrete time transfer functions with different numbers of poles and zeros then used the Impulse, Step, Dance, and Controller3 inputs to validate the resulting models.

For speed = 22 km/h, we reached:

- A system with 3 poles and 1 zero and output accuracies of 98.9, 87.68, 74.57, 99.24, 96.43 percent for the Controller 2, Controller 3, Dance, Step, and Impulse inputs respectively.
- A system with 4 poles and 2 zeros and output accuracies of 98.72, 87.68, 15.55, 97.72, 72.66 percent for the Controller 2, Controller 3, Dance, Step, and Impulse inputs respectively.

  We picked the model 3 poles and 1 zero to be best representative of the car at this speed.

For speed = 43 km/h, we reached:

- A system with 3 poles and 1 zero and output accuracies of 98.06, 97.45, 89.17, 94.93, 96.91 percent for the Controller 2, Controller 3, Dance, Step, and Impulse inputs respectively.
- A system with 4 poles and 2 zero and output accuracies of 99.3, 98.7, 90.07 , 98.88, 98.28 percent for the Controller 2, Controller 3, Dance, Step, and Impulse inputs respectively.

  We picked the model 4 poles and 2 zeros to be best representative of the car at this speed.

For speed = 72 km/h, we reached:

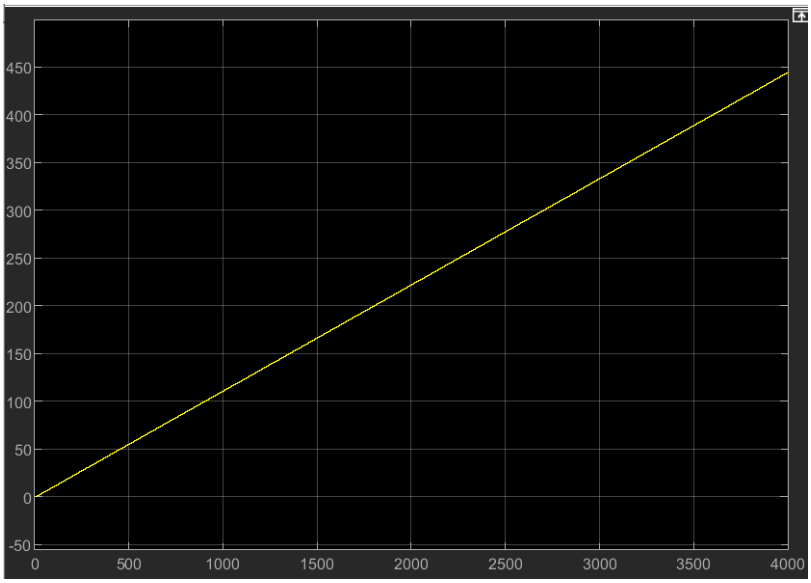
- A system with 2 poles and 1 zero and output accuracies of 79.74, 68.25, -29.89, 70.45, 32.68 percent for the Controller2, Controller 3, Dance, Step, and Impulse inputs respectively.
- A system with 3 poles and 1 zero and output accuracies of 79.32, 68.04, -41.55, 66.61, 29.12 percent for the Controller 2, Controller 3, Dance, Step, and Impulse inputs respectively.
- A system with 4 poles and 2 zero and output accuracies of 80.2, 68.57, -26.86, 72.15, 36.24 percent for the Controller 2, Controller 3, Dance, Step, and Impulse inputs respectively.

  We picked the model 4 poles and 2 zeros to be most representative of the car at this speed.

In the end, we picked the speed of 43 km/h and chose the transfer function with 4 poles and 2 zeros to model our vehicle. The transfer function we produced is:

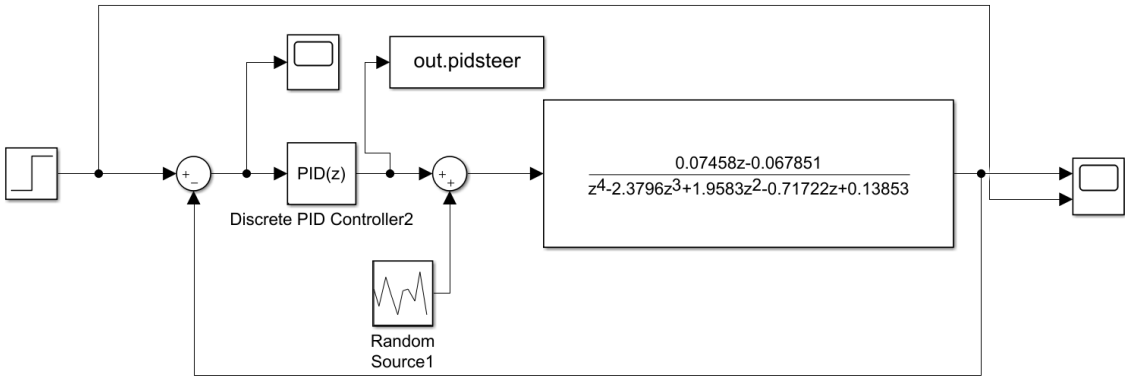
$$\frac{0.07458z-0.067851}{z^4-2.3796z^3+1.9583z^2-0.71722z+0.13853}$$

With the step response:



Representing the yaw of the car in response to a step input of steer = 1. This means that with a constant input steer, the yaw position of the car will keep on changing without halting. This matches our observation from CARLA.

Our closed loop system's block model:



In modeling the car and in all the steps that follow, we use the time-step of the CARLA simulator as our time unit rather than the second. On that basis, we set our

sampling time to 1 time-step. When simulated, this time-step equals to 33 milliseconds.

Setting a start point for the controller's specifications:

To start the controller design process, we agreed on a set of initial specifications that we deemed reasonable for the controller to meet.

Initial specifications:

Specification	**Value**
Rise time	12 time-steps
Settling time	100 time-steps
Maximum overshoot percentage:	9%
Steady state error	About 0
Gain Margin	20 dB
Phase Margin	70 deg
Maximum overshoot due to disturbances	9%
Settling time due to disturbances	100 time-steps
Steady state error due to disturbances	0.015
Controller effort	25

We define these specifications as follows:

- **Rise Time:** The number of time-steps taken until the step response hits an output of 1.
- **Settling time:** The number of time-steps taken until the step response settles to a value with an absolute error of 0.0005 from the output of 1.
- **Controller effort:** the square sum of the steer commands in the first 2000 time-steps. This definition was reached by observing a few simulations of PID controllers with arbitrary parameters in which we noticed that the error stabilizes to a value in the -14^{th} order after about 1800 time-steps.
- **Steady state error:** The average error of 2000^{th} till the 2500^{th} time-step. This definition was reached due to the same observation.
- **Gain Margin, Phase Margin and Maximum overshoot percentage** are defined according to their standard definitions.
- **Disturbance** is applied to the output of the controller under step input and the same specifications with the same definitions apply.

Implementing a PID controller on the CARLA Client:

On Carla, we implemented a PID controller difference equation using the Forward Euler integral and derivative formulas (square approximations). These are the same formulas implemented by default on Simulink's PID blocks.

$$\boldsymbol{u[k] = b_0\, e[k] + b_1\, e[k-1] + b_2\, e[k-2] - a_1\, u[k-1] - a_2\, u[k-2]}$$

$$\boldsymbol{where\ a_1 = -1, \quad a_2 = 0}$$
$$\boldsymbol{and\ b_0 = K_p + K_i + K_d, \quad b_1 = -(K_p + 2K_d), \quad b_2 = K_d}$$

Designing the PID controller:

We used Simulink and Matlab's Control Systems Toolbox to iteratively design two PID controllers that meet the specifications we chose above: one using a grid search implemented by Matlab's PID Tuner Tool and the other using the Root Locus method using Simulink's Control System Designer. We compared the performance of each PID controller in the Simulink model to that in the CARLA simulator.

To perform said comparisons for each controller, we subjected our controller-plant systems to the same set of desired yaws that resulted from implementing this particular controller in the provided CARLA Racetrack map.

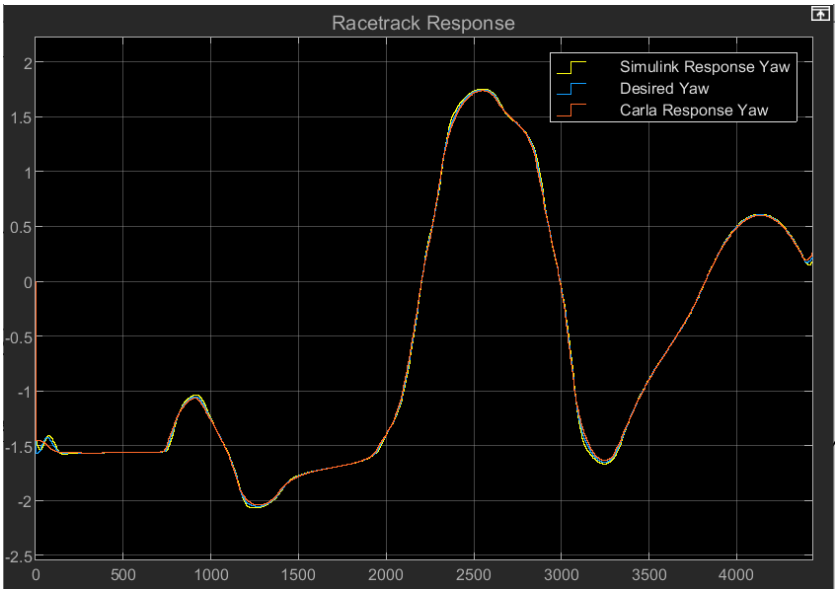
A) Using the auto-tuner:

- Iteration 1:

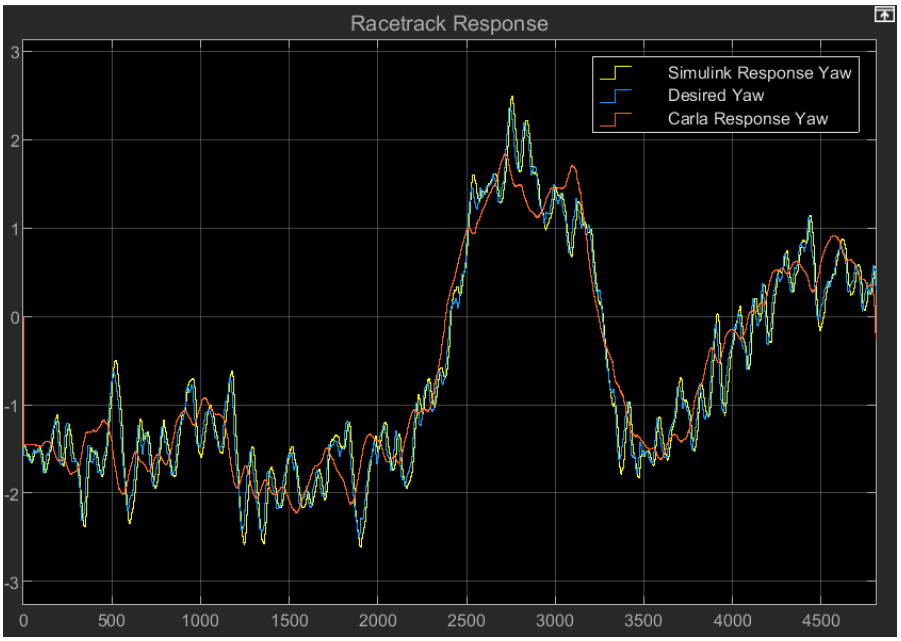
Kp = 0.8	Ki = 0.02	Kd = 0.03

Specification	**Value**
Rise time	20 time-steps
Settling time	254 time-steps
Maximum overshoot percentage:	19.46%
Steady state error	1.1119e-14
Gain Margin	37.7 dB
Phase Margin	19.4 deg
Maximum overshoot due to disturbances	15.89%
Settling time due to disturbances	160
Steady state error due to disturbances	0.0142
Controller effort	6.389

We simulated this controller on Carla and this is the plot of the Carla response against the Simulink response and the desired yaws:



With random disturbance in the range (-0.1, 0.1):

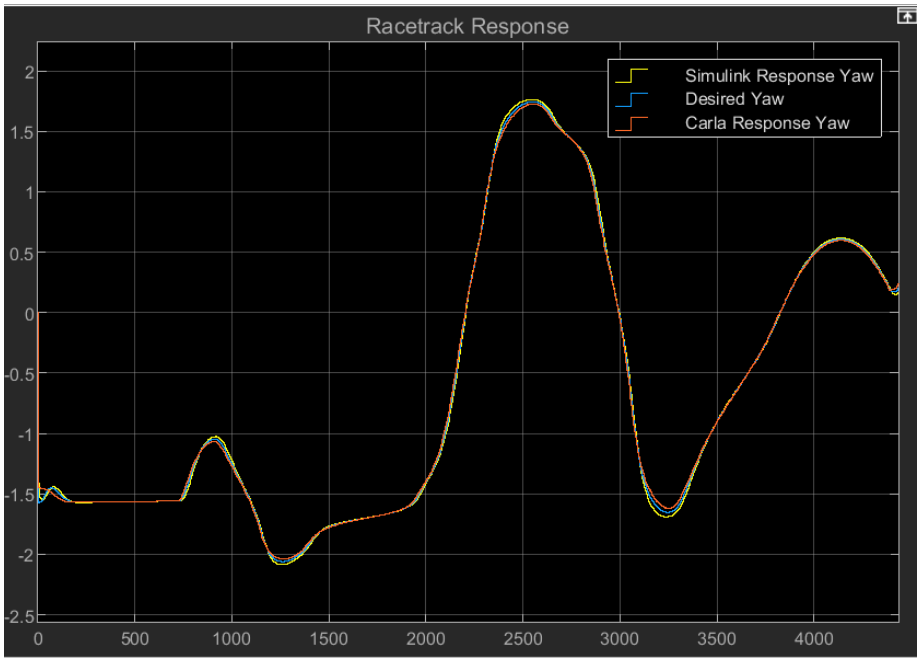


- Iteration 2:

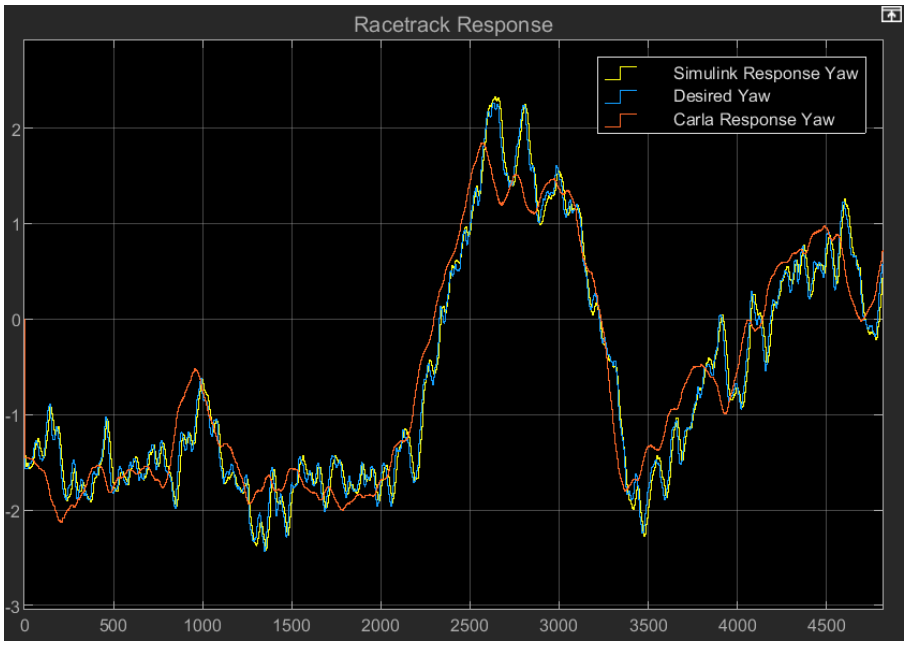
Kp = 0.85242	Ki = 0.010776	Kd = 1.1801

Specification	Value
Rise time	25 time-steps
Settling time	161 time-steps
Maximum overshoot percentage:	9.78%
Steady state error	6.3715e-15
Gain Margin	14 dB
Phase Margin	149 deg
Maximum overshoot due to disturbances	11.56%
Settling time due to disturbances	164
Steady state error due to disturbances	0.0136
Controller effort	6.9133

We simulated this controller on Carla and this is the plot of the Carla response against the Simulink response and the desired yaws:



With random disturbance in the range (-0.1, 0.1):

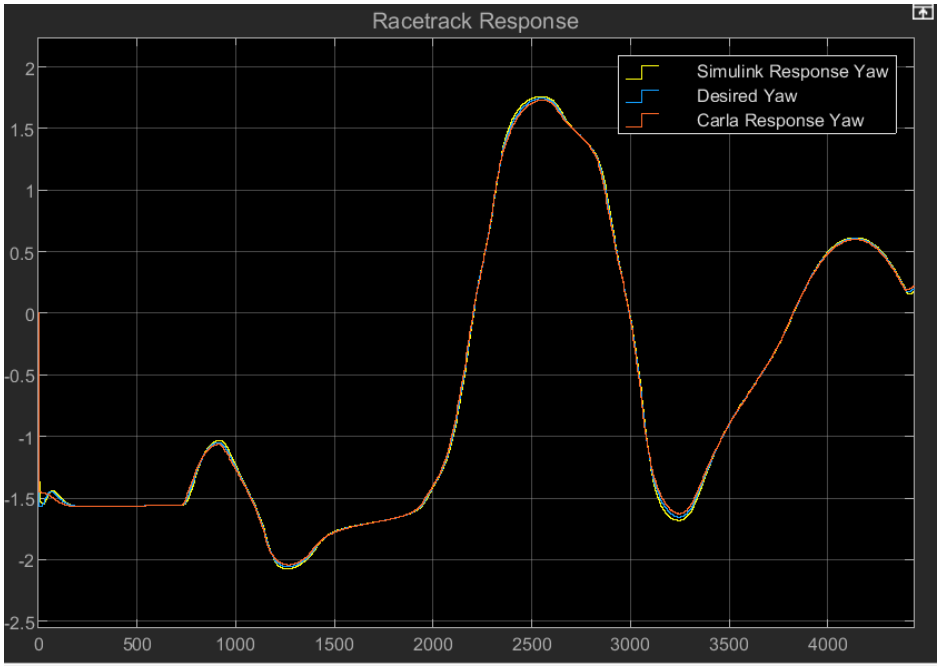


- Iteration 3:

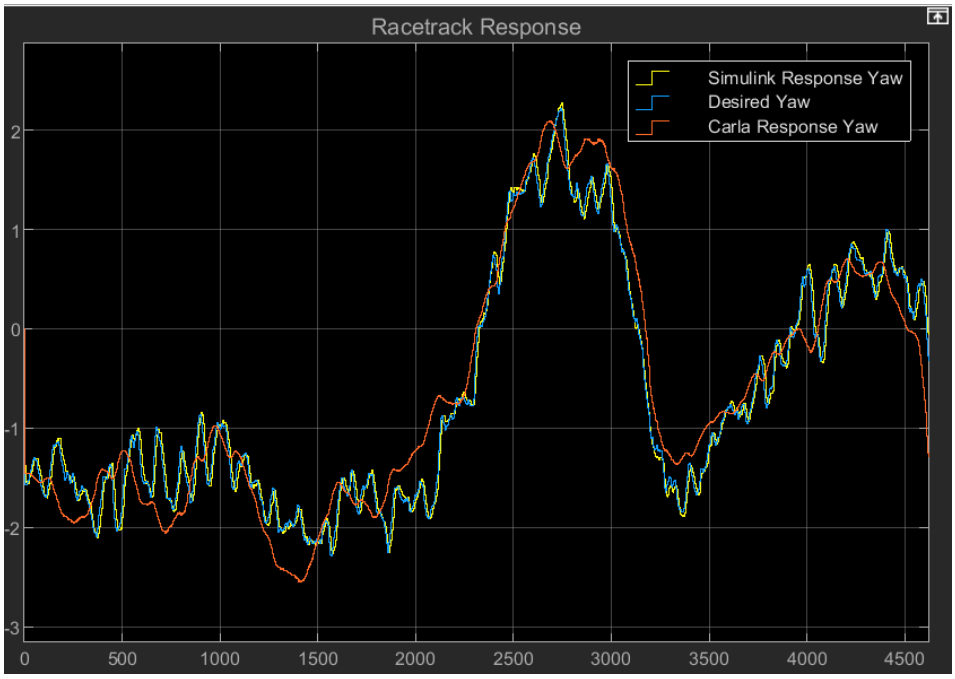
Kp = 1.0772	Ki = 0.013899	Kd = 1.3096

Specification	**Value**
Rise time	20 time-steps
Settling time	236 time-steps
Maximum overshoot percentage:	8.57%
Steady state error	6.7296e-15
Gain Margin	12.3 dB
Phase Margin	151 deg
Maximum overshoot due to disturbances	8.73%
Settling time due to disturbances	188
Steady state error due to disturbances	0.0105
Controller effort	9.2485

We simulated this controller on Carla and this is the plot of the Carla response against the Simulink response and the desired yaws:



With random disturbance in the range (-0.1, 0.1):



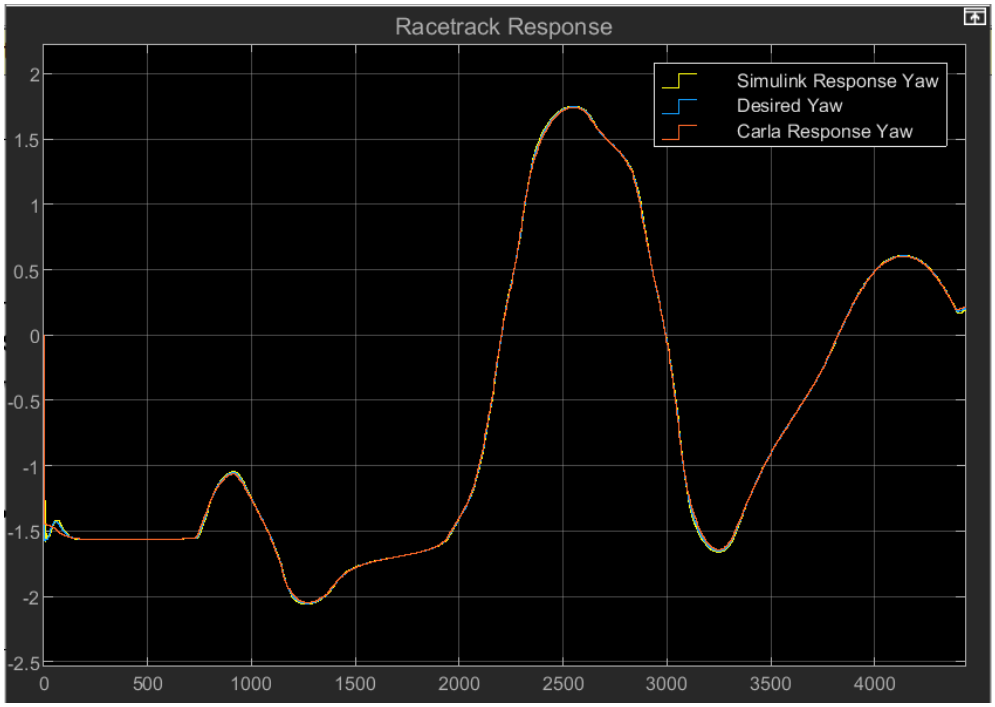
- Iteration 4:

Kp = 1.37411238	Ki = 0.03460598	Kd = 2.83167288

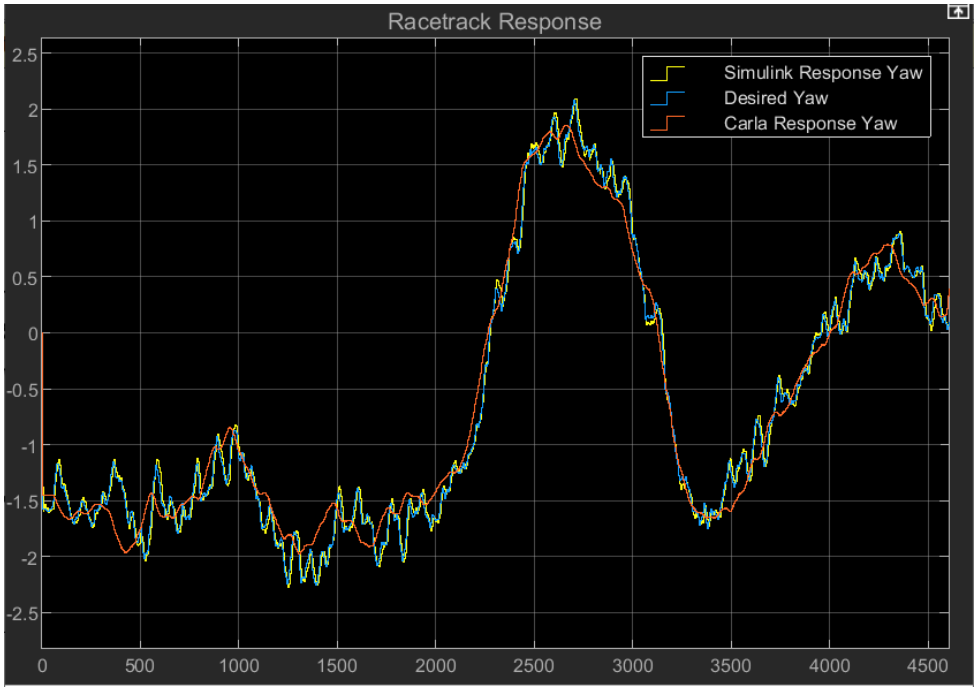
Specification	**Value**
Rise time	17 time-steps

Settling time	129 time-steps
Maximum overshoot percentage:	11.48%
Steady state error	4.6873e-15
Gain Margin	5.68 dB
Phase Margin	148 deg
Maximum overshoot due to disturbances	13.01%
Settling time due to disturbances	88
Steady state error due to disturbances	0.0112
Controller effort	17.3663

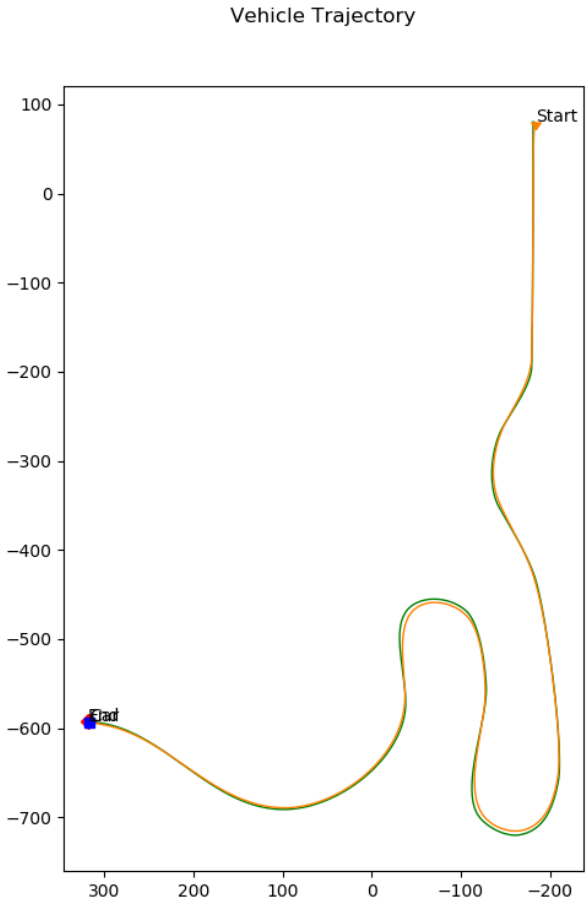
We simulated this controller on Carla and this is the plot of the Carla response against the Simulink response and the desired yaws:



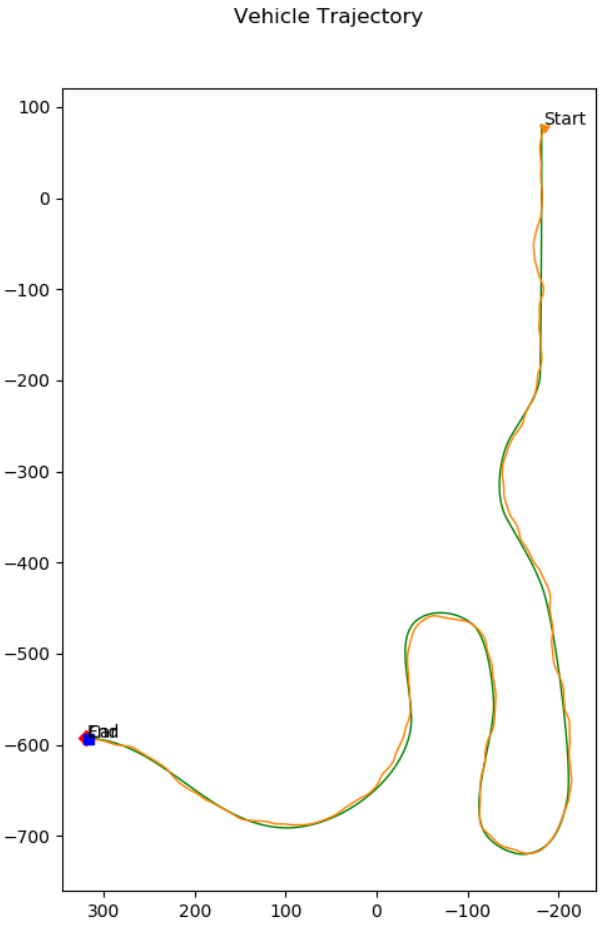
With random disturbance in the range (-0.1, 0.1):



- Final grid search vehicle trajectory:



- Final grid search vehicle trajectory with disturbance:



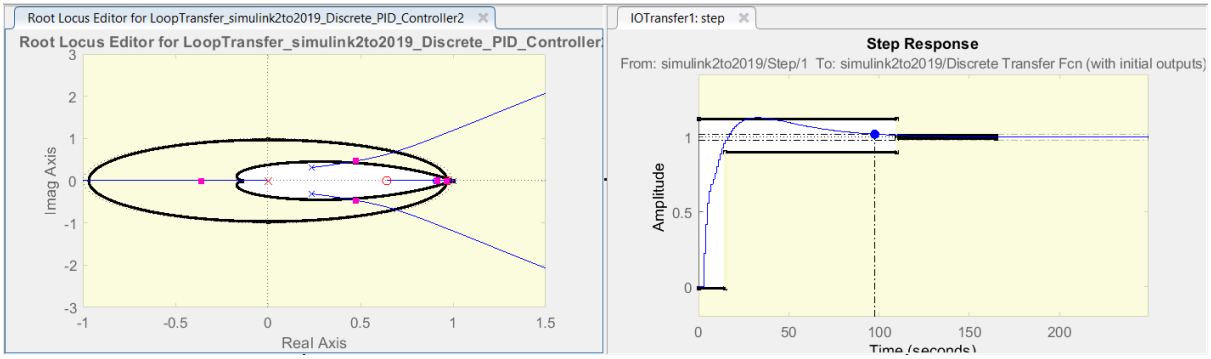
- After performing these iterations and surveying the changes in controller specifications as we changed its gains, we found that the specifications we have set earlier cannot be achieved all together. We hence loosened our required specifications to become:

Specification	**Value**
Rise time	20 time-steps
Settling time	130 time-steps
Maximum overshoot percentage:	12%
Steady state error	10e-14
Gain Margin	5.5 dB
Phase Margin	70 deg

Maximum overshoot due to disturbances	14%
Settling time due to disturbances	130
Steady state error due to disturbances	0.015
Controller effort	25

B) Using the Root Locus Method:

This time we again started with the 0.8, 0.02, 0.03 controller, then we tuned it using the root locus method. On the root locus, we specified the settling time and maximum overshoot percentage criteria. Then, we moved the poles and zeroes to match these criteria while monitoring these changes on the step response graph. The bode plot and the step response selected are shown below.



The first thing to notice about the bode plot is the number of PID poles, which is two instead of one as expected. This is because the PID Designer requires the presence of a filter and so we added an initial pole at z = 0 so that it will have minimal effect on the step response.

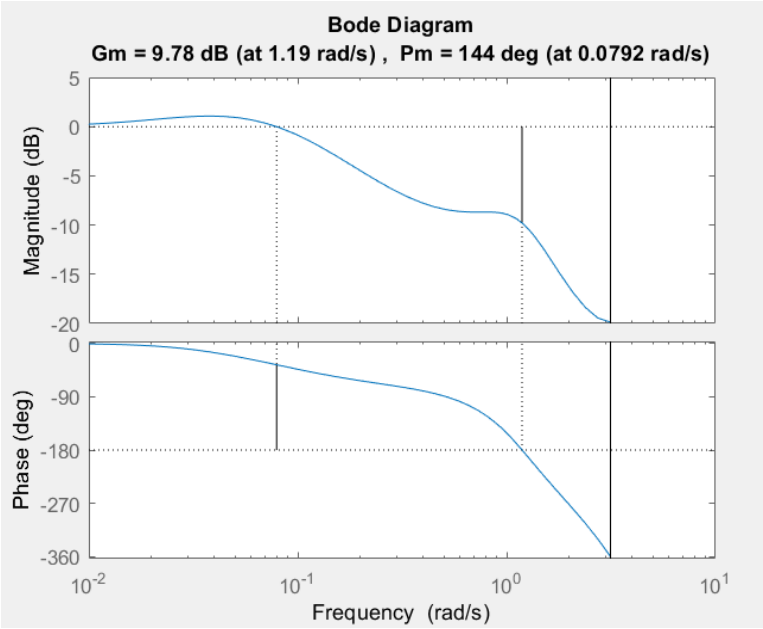
After tinkering with the closed loop poles, we reached the conclusion that the poles shown above were the most suitable for our specifications. It is noticeable that they do not exactly lie within the bounds set by the settling time and the maximum overshoot. That is because other factors were taken into consideration as well, such as controller effort and steady state error. This will make the settling time slightly exceed our previous specification.

Updating the PID block, we reached the following PID constants:

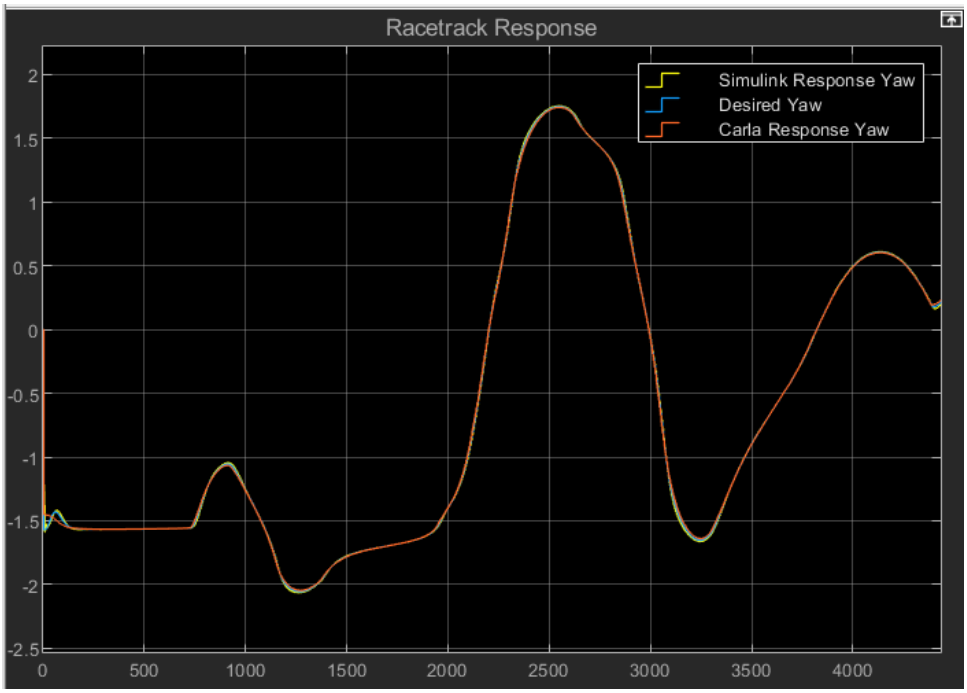
Kp = 1.1292	Ki = 0.0270	Kd = 1.8830

Specification	Value
Rise time	17 time-steps
Settling time	150 time-steps
Maximum overshoot percentage:	13.5%
Steady state error	7.7933e-15
Gain Margin	9.78 dB
Phase Margin	144 deg
Maximum overshoot due to disturbances	14.1%
Settling time due to disturbances	165 time-steps
Steady state error due to disturbances	0.011
Controller effort (sum of absolute value of errors)	13.1602

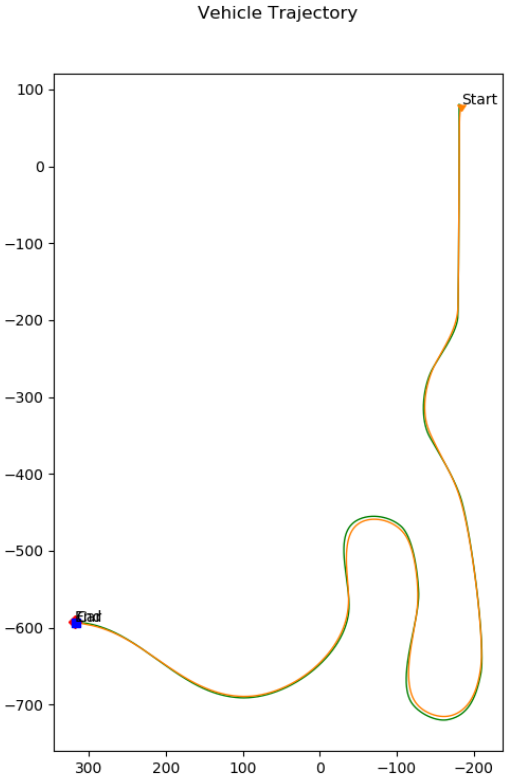
As expected, the settling time is 10 timesteps above our specifications. However, this slight increase in settling time is a reasonable cost for the robustness of the rest of the specifications, of which we specifically note the robustness of the gain and phase margins:



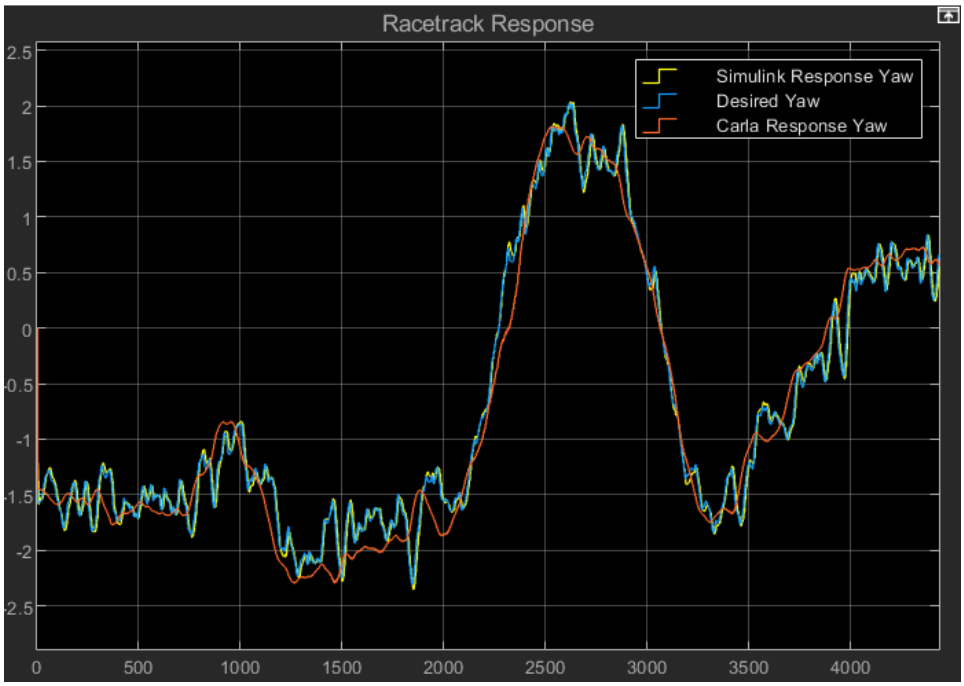
We simulated this controller on CARLA and this is the plot of the CARLA response against the Simulink response and the desired yaws:



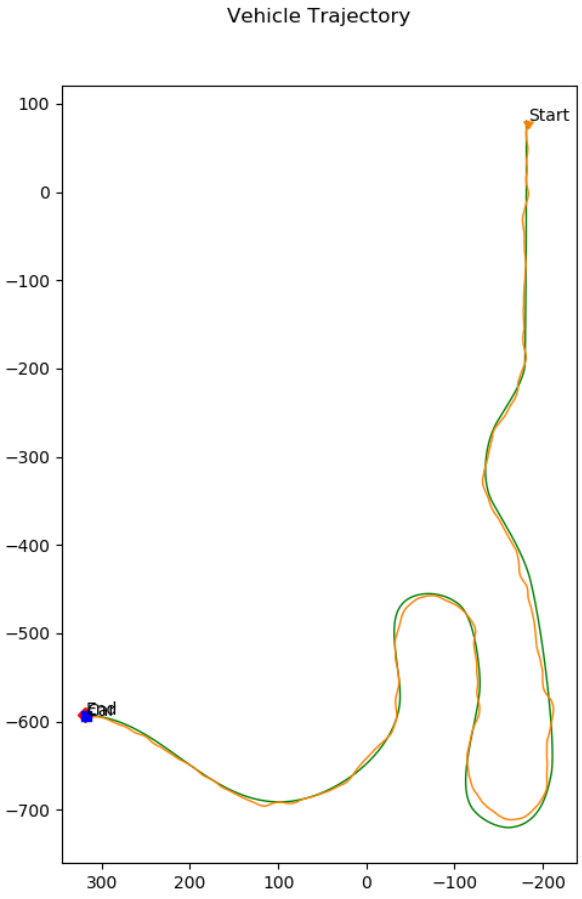
Vehicle Trajectory:



With random disturbance in the range (-0.1, 0.1):



Vehicle trajectory:



Conclusions and discussion:

The previous process of designing a PID controller for our simulated car system required several iterations to specify the most suitable PID constants. Along the way, we have redefined our specifications more than once based on the outputs. Our model of the CARLA-provided vehicle is highly accurate and this can be observed from all the previous CARLA vs. Simulink graphs in the absence of disturbances. However, in the case of disturbances, the model loses its accuracy and cannot quite predict the behavior of the simulations. This might be due to the fact that the car was identified using smooth input steers, and so our model fails to describe its behavior under rapidly-varying inputs.

Our model and PID controllers are very suitable for controlling the CARLA-provided and for minimizing the error to the track waypoints. They are simple models with a small number of zeroes and poles and thus can be easily altered to meet any set of specifications and provide simple time-domain equations.

Extra Notes:

- Python line to start Carla server:

CarlaUE4.exe /Game/Maps/RaceTrack -windowed -carla-server -benchmark -fps=30

- Python code for “Controller” steer input:

```
if yaw < desired_yaw - 0.05:
  steer_output  = 0.5
elif yaw > desired_yaw + 0.05:
  steer_output  = -0.5
else:
  steer_output  = 0
```

- Python code for “Controller2” steer input:

```
if yaw < desired_yaw - 0.1:
   steer_output  = 0.2
elif yaw < desired_yaw - 0.08:
   steer_output  = 0.15
elif yaw < desired_yaw - 0.05:
   steer_output  = 0.1
elif yaw < desired_yaw - 0.03:
   steer_output  = 0.05
elif yaw > desired_yaw + 0.1:
   steer_output  = -0.2
elif yaw > desired_yaw + 0.08:
   steer_output  = -0.15
elif yaw > desired_yaw + 0.05:
   steer_output  = -0.1
elif yaw > desired_yaw + 0.03:
   steer_output  = -0.05
```

```
else:
    steer_output  = 0
```

- Python code for “Controller3” steer input:

```
if yaw < desired_yaw - 0.1:
    steer_output  = 0.3
elif yaw < desired_yaw - 0.08:
    steer_output  = 0.2
elif yaw < desired_yaw - 0.05:
    steer_output  = 0.1
elif yaw < desired_yaw - 0.03:
    steer_output  = 0.05
elif yaw > desired_yaw + 0.1:
    steer_output  = -0.3
elif yaw > desired_yaw + 0.08:
    steer_output  = -0.2
elif yaw > desired_yaw + 0.05:
    steer_output  = -0.1
elif yaw > desired_yaw + 0.03:
    steer_output  = -0.05
else:
    steer_output  = 0
```

- Python code for “Step” steer input:

```
if t>15:
  steer_output = 0.1
else:
  steer_output = 0
```

- Python code for “Dance” steer input:

```
if t > 15:
  if t - np.floor(t) < 0.5:
   steer_output  = 0.1
  else:
   steer_output  = -0.1
  else:
   steer_output  = 0
```

- Python code for “Impulse” steer input:

```
if t > 15 && t <= 15.5:
 steer_output  = 0.1
else:
 steer_output  = 0
```