Code a Car Camp	Summer 2022
Written by: ECE484 and Code A Car camp team	*MP2: Vehicle Model and Control*

Website 

1 Introduction

In this MP, you will use the techniques discussed during the lecture to tune a waypoint tracking controller, which can be an important part for the future MPs and projects. Your goal is to play with the controller gains to achieve better performance of the controller. ROS is used in this MP to connect the vehicle model and the controller to the simulator and acquire commands from the user.

Learning objectives

- Vehicle models
- Waypoint following controller design for vehicles
- Controller design with state feedback control

System requirements

- Ubuntu 20.04
- ROS Noetic

2 Vehicle Controller

In this part of the MP, you will need to tune a PD vehicle controller to drive the vehicle along the track shown in Figure 1. Your goal is to minimize the amount of time it takes for the vehicle to run one lap around the track while ensuring the vehicle stays on the track. In the following section, we will discuss how you will be tuning the controller.

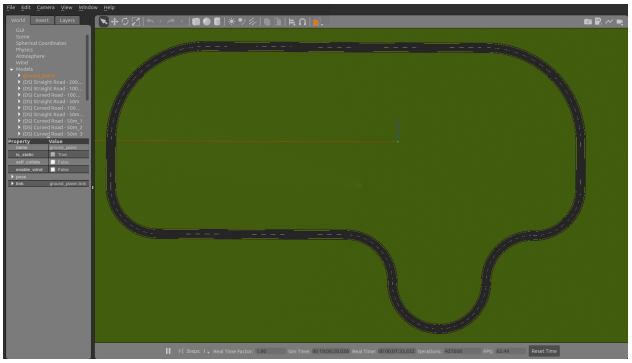

Figure 1: The race track that the vehicle is going to follow

2.1 Module Architecture

In this assignment, you will mainly need to modify the PD gains in *controller.py*.

The controller works as follows: The controller takes the current state of the vehicle (pose and twist) and the reference state of the vehicle (position, orientation, and velocity) and use them to compute the speed and steering angle necessary to reach the next waypoint.

2.1.1 set_pos.py

This is a utility function that allows you to set position of the vehicle without restarting the simulator. The vehicle can be set to any position with orientation 0. You can set the position of the vehicle using command

```
rosrun mp2 set_pos.py --x 0 --y -98
```

Note that the starting point of the vehicle is at $[x, y] = [0, -98]$.

2.1.2 main.py

As the name implies, this file contains the main function of this MP. You should run this file with python3 to drive the vehicle model.

run_model This is the main function for this MP. It will loop through the waypoint list and call the controller to drive the vehicle to follow all the waypoints. In addition, this function will use the current waypoint and the previous waypoint to compute the reference state of the vehicle. The reference position is computed by interpolating the position of the current waypoint and the previous waypoint. The reference orientation will be the orientation of vector starting from previous waypoint to current waypoint.

2.2 Tuning the controller

2.2.1 Background

In this part of the MP, you will implement the path following controller discussed during the lecture to drive the simulated vehicle along the given trajectory. Given a reference state $[x_{ref}, y_{ref}, \theta_{ref}, v_{ref}]$ and the current state $[x_B, y_B, \theta_B, v_B]$, the "error" vector $\delta = [\delta_x, \delta_y, \delta_\theta, \delta_v]^T$ can be defined as

$$\begin{aligned}
\delta_x &= cos(\theta_{ref}) * (x_{ref} - x_B) + sin(\theta_{ref}) * (y_{ref} - y_B) \\
\delta_y &= -sin(\theta_{ref}) * (x_{ref} - x_B) + cos(\theta_{ref}) * (y_{ref} - y_B) \\
\delta_\theta &= \theta_{ref} - \theta_B \\
\delta_v &= v_{ref} - v_B
\end{aligned} \tag{1}$$

With the definition of error, the control input to the vehicle $u = [v, \delta]$ can be obtained by

$$u = K * \delta, \tag{2}$$

where K can be defined as

$$K = \begin{bmatrix} k_x & 0 & 0 & k_v \\ 0 & k_y & k_\theta & 0 \end{bmatrix}, \tag{3}$$

where each k term should be non negative. The path following controller produced by this gain matrix performs a PD-control. It uses a PD-controller to correct along-track error for longitude control. The lateral control is also a PD-controller for cross-track error because δ_θ is related to the derivative of δ_y. More information about this controller implementation can be found here.

2.2.2 Tuning

Choosing the optimal parameters for a PID controller with a nonlinear plant is always a hard problem. Below, we provide a set of potential parameters for this waypoint following controller

$$\begin{aligned}
k_x &= \begin{bmatrix} 0.1 & 0.5 & 1.0 \end{bmatrix} \\
k_y &= \begin{bmatrix} 0.05 & 0.1 & 0.5 \end{bmatrix} \\
k_v &= \begin{bmatrix} 0.5 & 1.0 & 1.5 \end{bmatrix} \\
k_\theta &= \begin{bmatrix} 0.8 & 1.0 & 2.0 \end{bmatrix}
\end{aligned}$$

You may also try tuning the parameters by yourself. Below we provide a potential method to choose the gains for this controller.

1. Set all gains to zero.
2. Increase the P gain until the response to a disturbance shows steady oscillation.
3. Increase the D gain until the oscillation goes away (i.e. it's critically damped).
4. Repeat step 2 and 3 until increasing the D gain does not stop the oscillation.
5. Set P and D to the last stable values.

Some additional method about tuning controller gains can be found here.

2.3 Development instructions

The parameters you will have to modify are located in `controller.py` which is located in the following path:

```
cd ~/Desktop/CodeACar22/src/mp2/src
```

In this MP, you will work with the vehicle model in the Gazebo Simulator. To start the simulator, run the following commands:

```
source /opt/ros/noetic/setup.bash
cd ~/Desktop/CodeACar22/
catkin_make
source devel/setup.bash
```

in the folder. There should be no error during the execution of the command and when finished, you should see two additional folders devel and build.

After all the previous setup steps are finished, you can start the simulator by running the command:

```
roslaunch mp2 mp2.launch
```

You should be able to see the Gazebo simulator window and the vehicle in the simulator as shown in figure 2 (you may need to rotate the camera).

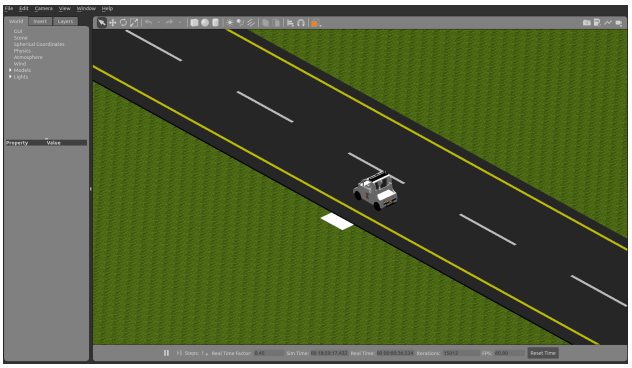

Figure 2: The initial state of the vehicle. Note that the starting point is marked by a small white rectangle.

Next, start the evaluator by opening a new terminal and typing following commands:

```
source /opt/ros/noetic/setup.bash
source ~/Desktop/CodeACar22/devel/setup.bash
rosrun mp2 evaluate.py
```

To run the controller, open a new terminal and run the following commands:

```
source /opt/ros/noetic/setup.bash
source ~/Desktop/CodeACar22/devel/setup.bash
rosrun mp2 main.py
```

2.4 Submission

For this mp, you are rewarded for completing the entire loop of the track and finishing it as fast as possible

When you are ready to submit to the leaderboard, run the *evaluate.py* command with the *upload* flag and then *main.py* in a separate terminal as the following:

```
rosrun  mp2 evaluate.py --upload
```