ECE 498SMA: Principles of safe autonomy Fall 2020

Written by: Yangge Li, Sayan Mitra *MP 2: Vehicle Model and Control*

Website *Due Date: 2020/10/16*

1 Introduction

In this MP, you use the techniques discussed during the lecture to develop a waypoint tracking controller which can be important parts for the future MPs and projects. You will play around with the controller gains to achieve better performance of the controller. ROS is used in this MP to connect the vehicle model and controller to the simulator and acquire command from user.
Same a previous MPs, this MP is divided into two parts. In the first part of this MP (section 2), you will be working on some theoretical problems about PID controller. You will have understanding about the purpose of gain values of PID controller. In the second part of this MP (section 3), you will working with the Gazebo simulator. You will need to use the knowledge from previous part of this MP and lecture, to develop a waypoint following controller for vehicle. You will need to use the controller to drive the simulated vehicle on a race track.
For part one, you will individually submit the solution to Problems 1-3 in one file `MP2_1_netid.pdf`. For part two, your group will need to submit a single file `MP2_2_groupnumber.pdf` with the solution to Problems 4-6. Name all the group members and cite any external resources you may have used in your solutions. More details for submission are given in Section 4. All the regulations for academic integrity and plagiarism spelled out in the student code apply.

Learning objectives

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

System requirements

- Ubuntu 16
- ROS Kinetic
- ros-kinetic-ros-control
- ros-kinetic-effort-controllers
- ros-kinetic-joint-state-controller
- ros-kinetic-ackermann-msgs

2 Written Problems

Problem 1 (10 points)**.** Consider the two dimensional ODE system described by:

$$\begin{aligned}\dot{x} &= x^2 + y\\ \dot{y} &= x - y + a,\end{aligned}$$

where a is a parameter of the model. Find all the equilibrium points of this system.

Problem 2 (10 points)**.** Consider an n-dimensional ODE $\dot{x} = f(x)$ and suppose the initial state of the system comes from a ball $\Theta = \{x \in \mathbb{R}^n \mid |x| \leq \rho\}$ of radius ρ centered at the origin. The symbols here have their usual meanings: the state of the system is an n-dimensional real vector, the function $f : \mathbb{R}^n \to \mathbb{R}^n$ is Lipschitz continuous, and any solution of the system is a function $\xi : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$, that is $\xi(x_0, t)$ is the solution of the system at time t starting from the initial state $x_0 \in \Theta$.

If the system is Lyapunov stable, then does it have any invariants (as defined in the first 2 lectures)? That is, can you come up with a set I such that for all $x_0 \in \Theta$, and for all $t \geq 0$, $\xi(x_0, t) \in I$.

What does this observation tell you about the relationship between Lyapunov stability and invariance?

Problem 3 (15 points)**.** Consider the 2-dimensional linear time invariant system:

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}0 & v\\ 1 & 2\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}u_1\\ u_2\end{bmatrix} = Ax + Bu,$$

where v is a model parameter. We would like to design a state-feedback controller to make the system asymptotically stable. Let the feedback law be of the form:

$$\begin{bmatrix}u_1\\ u_2\end{bmatrix} = -\begin{bmatrix}k_{11} & k_{12}\\ 0 & k_{22}\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} = -Kx.$$

Write down the equations for the closed loop system. Suppose the gain k_{22} is set to be 0. Write down conditions on k_{11} and k_{12} or specific values that makes the closed loop system asymptotically stable. Show your work.

3 Implementing Vehicle Controller with Gazebo

In this part of the MP, you will need to develop a vehicle controller to drive the vehicle along the track shown in fig 1. You are allowed to use any control method you like to fulfill the task. You will need to record the amount of time it take for the vehicle to run one lap around the track. In the following section, we will discuss a waypoint following strategy as discussed during the lecture as a reference method to solve the problem.

3.1 Module architecture

The supporting code is available from this git repo. The provided code for MP2 is located in `./src/dev` folder. In this assignment, you will mainly need to implement function **execute** in controller.py. However, we strongly encourage you to read through the code and feel free to modify the provided code if necessary. You will learn ROS mechanics from this and some modules of this MP can be helpful for your future MPs and project.

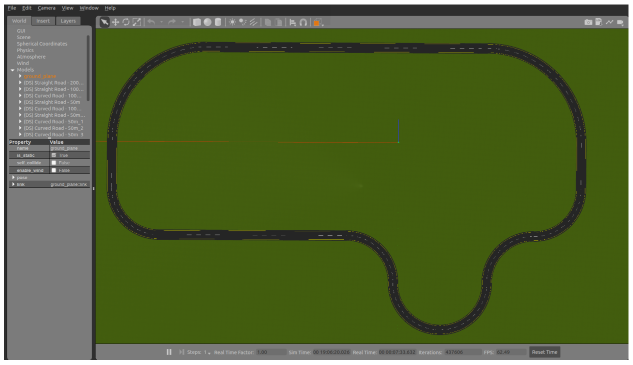

Figure 1: The race track that the vehicle is going to follow

3.1.1 controller.py

This file contains class vehilceController, that holds the controller for the vehicle. The class have following member functions.

getModelState This function will call the "/gazebo/get_model_state" service that will return a message that contains the position, orientation, linear velocity and angular velocity of the vehicle. The return value of this function is a Gazebo message ModelState. This message is widely used in ROS describe a Gazebo model's pose, which includes position and orientation and twist, which includes linear and angular velocity. Note the orientation information is in form of quaternion in the message. The content in ModelState message **msg** can be accessed by

```
msg.pose.position.x
```

More details about ModelState message can be found here.

execute This function contains the controller which will enable the vehicle to drive to a waypoint. The function will take 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 waypoint. The current state the vehicle is stored in a ModelState message format and the reference state of the vehicle is stored in a list with 4 elements [reference_x, reference_y, reference_orientation, reference_velocity].
The computed control input to the vehicle will be the steering angle and the velocity of the vehicle. These two values will be packed into an AckermannDrive message. The AckermannDrive message is commonly used in ROS to drive car-like vehicle using AckermannDrive steering. The content in AckermannDrive message **msg** can be accessed by

```
msg.speed
```

More details about AckermannDrive message can be found here. The AckermannDrive message containing the control input to the vehicle will then be published to the vehicle.

3.1.2 utilities.py

This files contains utility functions used for this MP.

euler_to_quaternion This function will convert euler angle to quaternion. The input to the function r is a list that contains the roll, pitch, yaw component of euler angle. The ouput of the function is a list that contains the x, y, z, w component of quaternion.

quaternion_to_euler As it's name implies, this function will convert quaternion representation of orientation to euler angle. The input to the function is the x, y, z, w component of the quaternion and the output is a list that contains the roll, pitch, yaw component of euler angle.

3.1.3 set_pos.py

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

```
python set_pos.py --x 0 --y 0
```

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

3.1.4 waypoint_list.py

This file contains a list of waypoint along the race track. Each waypoint contains three components, x position of the target waypoint, y position of the target waypoint and the reference velocity of the target waypoint. The provided waypoints is just a reference, so feel free to modify the centent of the list or add additional waypoints if needed.

3.1.5 main.py

As the name implies, this file contains the main function of this MP. You should run this file with python2 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 previous waypoint to compute the reference state of the vehicle. The reference position is computed by interpolating the position of current waypoint and previous waypoint. The reference orientation will be the orientation of vector starting from previous waypoint to curent waypoint.

3.2 Development instructions

3.2.1 Running Gazebo Simulator

In this MP, you will work with the vehicle models in the Gazebo Simulator. To run the simulator, you should first go to the root directory of the files you downloaded from git repository where you should see a src folder. The next step is to run command

```
catkin_make
```

in the folder. There should be no error during the execution of the command and when finished, you should see two folders devel and build been generated.
The next step is to run command

```
source ./devel/setup.bash
```

in the root directory of the files you downloaded. This command should be executed every time before you try to run the simulator from a new terminal.
After all the previous setup steps are finished, you can start the simulator by running command

```
roslaunch gem_gazebo gem_vehicle.launch
```

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

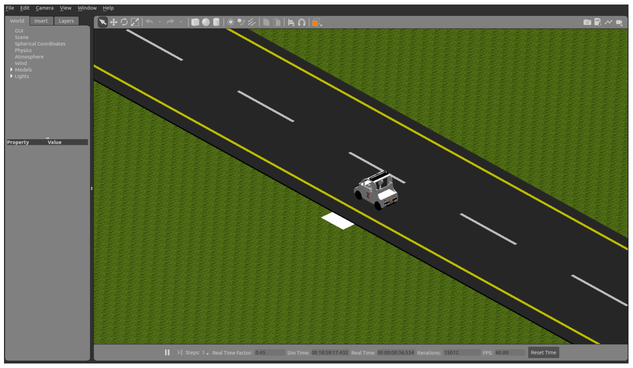

Figure 2: The initial state of vehicle. Note the starting point is marked by a small white rectangle.

To run the controller, you can go to the src/mp2 folder and use command

```
python main.py
```

3.3 Implementing the controller

In this part of the MP, you will implement the path following controller discussed during the lecture to drive the simulated vehicle along given trajectory. Given a reference state $[x_{ref}, y_{ref}, \theta_{ref}, v_{ref}]$ and 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

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

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 (position error and velocity error). The control on curvature is also a PD-controller for cross-track error because δ_y is related to the derivative of δ_θ. Some more information about this controller implementation can be found here.
Choosing the optimal values of gains for PID controller with nonlinear plant is always a hard problem. Below provides one 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 is steady oscillation.
3. Increase the D gain until the the oscillations go away (i.e. it's critically damped).
4. Repeat steps 2 and 3 until increasing the D gain does not stop the oscillations.
5. Set P and D to the last stable values.

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

3.4 Report

For the programming part of this MP, you will need to answer the following problems

Problem 4 (15 points)**.** Describe the controller you used to solve the problem. How do you choose the parameters for the controller? How long does it take for the vehicle to run one lap around the track?

Problem 5 (20 points)**.** Plot the trajectory of the the vehicle is travelling. You should draw a x-y plot with the default initial position when you start gazebo simulator. In addition, you should mark the waypoints in your plot.

Problem 6 (20 points)**.** Record a video for one example execution of this scenario. The video should include the GAZEBO window. Provide a link to the video and include it in the report.

4 Report and Submission

For Problems 1-3, each student should write a report that contains solutions for individually. You may. discuss the problems following the tenets of academic integrity and collaboration. This report should be submitted to gradescope **individually** to assignment with filename `MP2_1_netid.pdf`. This part will have 35% of the total grade of this MP.

For Problems 4-6, each group should write a report that contains the solutions, plots, and discussions. This report should be submitted to gradescope **per group** to assignment with filename `MP2_2_report_#.pdf`. In addition, each group should submit the code to gradescope. The content in folder — should be submitted to gradescope **pergroup** to assignment in a zip file with filename `MP2_2_code_#.zip`. This part will have 55% of the total grade of this MP.

For this MP, you will need to demo your code to one of the TA. You can do the demo during any of the lab sessions or office hours. The TA will primarily check if the vehicle can properly follow the provided track. All members should show up and be prepared to answer some questions. The TA might pick a random person and ask that person to answer the question. So all students should be familiar with the content. This part will have 10% of the total grade of this MP.