ECE 484: Principles of safe autonomy	Fall 2025
Written by: ECE484 team	*HW2, MP2: Vehicle Model and Control*
Website 	*Due Date: 10/03/2025*

1 Introduction

In this assignment, you will develop both a lateral and a longititudal controller for a car. These can be useful for future MPs and your project. ROS is used to connect the vehicle model and controller to the simulator.

As usual, this assignment has two parts. In Section 2, you will work on theoretical problems individually. Submit a file `hw2_<netid>.pdf` with the solution to Problems 1-3. In Section 3, you will work with the Gazebo simulator to create a vehicle controller with your MP team. You will use the controller to drive the simulated vehicle on a race track. Your MP group will need to submit a file `mp2_<groupname>.pdf` with the solution to Problems 4-8. Name all the group members and cite any external resources you may have used in your solutions. Please include the links to your code and video in the report. All the regulations for academic integrity and plagiarism spelled out in the student code apply.

Learning objectives

- Vehicle models
- Longitudinal and Lateral controller design for vehicles

System requirements

- Ubuntu 22.04
- ROS Humble
- Gazebo 11
- ros-humble-ackermann-msgs
- ros-humble-ros2-control
- ros-humble-ros2-controllers

2 Written Problems

Problem 1 (30 points)**.** Consider an adapted version of the Dubins car model, which is a nonholonomic model of a car moving in a plane. The state of the car is described by its position (x, y) and its heading θ. The equations of motion for this simplified model are given by:

$$\begin{aligned}\dot{x} &= 2v\cos(\theta),\\ \dot{y} &= -3v\sin(\theta),\\ \dot{\theta} &= u,\end{aligned}$$

where v is the constant speed of the car and u is the control input representing the turning rate, rather than the steering angle, making this a simplified model.

For this problem, you are to:

1. Derive the car's trajectory over time by solving the differential equations given the initial conditions $x(0) = 0$, $y(0) = 0$, $\theta(0) = 0$, a constant speed $v = 1$, and a constant turning rate $u = 0.5$.
2. Plot the trajectory of the car from $t = 0$ to $t = 8$ seconds.
3. Discuss the path taken by the car and how changes in u (the turning rate) might affect this path.

This task requires integrating the given system of differential equations to find $x(t)$, $y(t)$, and $\theta(t)$, and then using these expressions to plot the trajectory of the car over the specified time horizon.

Problem 2 (30 points)**.** Consider the two-dimensional linear time invariant system $\dot{x} = Ax$, where

$$A = \begin{bmatrix} 1 & 2 \\ -4 & b \end{bmatrix}$$

Is the system asymptotically stable for $b = -2$? Is the system asymptotically stable for $b = 6$? Explain why, considering the eigenvalues of A.

Problem 3 (40 points)**.** Consider the 2-dimensional linear time invariant system:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 3 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = Ax + Bu,$$

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

$$\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = - \begin{bmatrix} 3 & k_{12} \\ 2 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = -Kx.$$

Write down the equations for the closed loop system. Also write down conditions on gain k_{12} 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 Figure 1. In addition, you need to do some analysis on certain metrics that measure the performance of your controller. Section 3.1 explains the module architectures and utility functions. Section 3.2 shows how to run the infrastructure when you finish the implementation. Section 3.3 describes the tasks you need to code for this assignments.

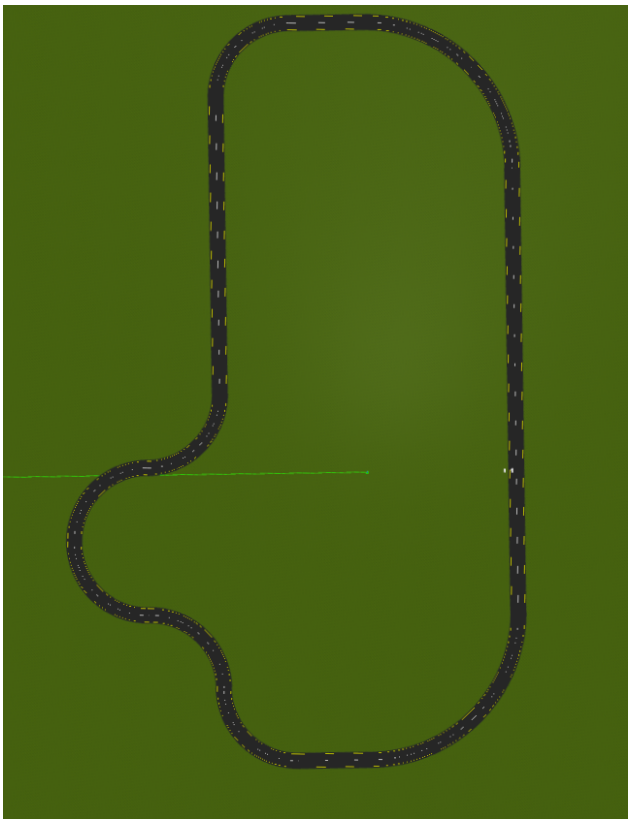

Figure 1: The race track that the vehicle is going to follow

3.1 Module architecture

The supporting code is available from this git repo. The provided code for MP2 is located in `src/mp2/src` folder. In this assignment, you only need to implement **three tasks** in controller.py. However, we strongly encourage you to read through all code so that you will have a better understanding of the MP. You will learn ROS mechanics from this and some modules of this MP can be helpful for your future MPs and project.

3.1.1 controller.py

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

execute This function contains the controller which will enable the vehicle to drive to the target waypoint. The function will take the current state of the vehicle and and all future waypoints(you may not need all) and use them to compute the speed and steering angle necessary to reach the waypoint. We will talk about the implementation details in Section 3.3. The computed control inputs to the vehicle are 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 ROS2 to drive car-like vehicle using AckermannDrive steering. The content in AckermannDrive message **msg** can be accessed by msg.*parameter*, for example:

```
msg.speed
```

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

Additionally, this function will contain the response from calling "/get_entity_state" service that will return a message that contains the position, orientation, linear velocity and angular velocity of the vehicle. This message is widely used in ROS2 to describe a Gazebo model's pose, which consists of position and orientation, and twist, which consists of linear and angular velocity. Note that the orientation is in the form of quaternion in the message. The content in the response of a "/gazebo/get_entity_state" message **msg** can be accessed by

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

More details about GetEntityState can be found here, and EntityState can be found here

3.1.2 util.py

This file contains some utility functions for this MP.

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

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

3.1.3 set_pos.py

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

```
python3 set_pos.py --x 0 --y -98 --yaw 0
```

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

3.1.4 waypoint_list.py

This file contains a list of waypoints along the race track. Each waypoint contains two components: x position of the target waypoint, and y position of the target waypoint.

3.1.5 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_loop 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.

get_entity_state This function executes the service calls to to the "/get_entity_state" service.

entity_state_callback This function is the callback function for "/get_entity_state" function and stores the entity state when the service call is complete.

3.2 Running instructions

3.2.1 Running Gazebo Simulator

In this MP, you will work with the vehicle model 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 the below command in the folder.

```
colcon build --symlink-install
```

There should be no error during the execution of the command and when finished, you should see three additional folders: log, build, and install.

The next step is to run the below command in the root directory of the files you downloaded.

```
source install/setup.bash
```

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 the command

```
ros2 launch mp2 mp2_simple.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).

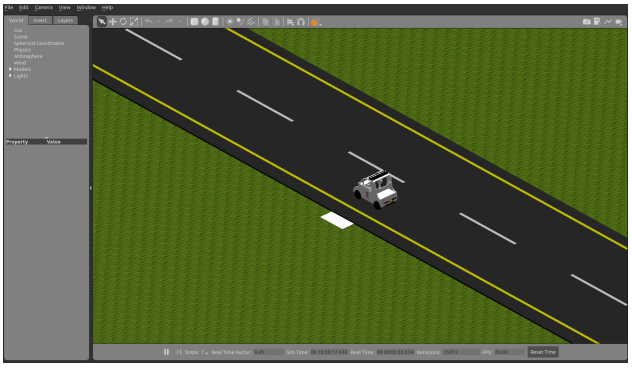

Figure 2: The initial state of the vehicle. Note that the starting point is marked by a small white rectangle.

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

```
python3 main.py
```

3.3 Implementing the controller

All you need to do is to finish the 3 functions inside **controller.py**, and do not modify the function name, input, output.

3.3.1 Task 1: Extract Vehicle Info from ROS

In this task, you are required to familiarize yourselves with ROS2 message and how to extract necessary information from the ros messages. Please read the documentation GetEntityState and EntityState to extract the vehicle's current states (pos_x, pos_y, vel, yaw). Hint: you may use the the utility function quaternion_to_euler() we provide to you.

```
def extract_vehicle_info(self, currentPose):
        # TODO
    return pos_x, pos_y, vel, yaw
```

3.3.2 Task 2: Implement Longitudinal Control

Having assessed the current status of the vehicle and the upcoming waypoints that have yet to be reached, your next task is to develop a longitudinal controller to determine the vehicle's target speed. Essentially, your goal is to establish a relationship between (the track's curvature, the vehicle's current state) and an appropriate target speed. For instance, if the upcoming waypoints suggest that your vehicle will be navigating a straight path, it would be advisable to aim for higher speeds. Conversely, if the future waypoints indicate that a sharp turn is imminent, it would be prudent to reduce speed. We suggest a baseline target speed of 12 m/s for straight sections and 8 m/s for turns. The method you use to derive this mapping is open-ended, and any sensible approach will be considered acceptable.

```
def longititudal_controller(x, y, vel, yaw, future_unreached_waypoints):
    # TODO
    return target_vel
```

3.3.3 Task 3: Implement Lateral Control

In this section, you are tasked with implementing a lateral controller for your vehicle, utilizing the Pure Pursuit algorithm to determine the appropriate steering angle. The Pure Pursuit algorithm employs a "look-ahead point," situated at certain distance ahead on the vehicle's reference path. Your objective is to guide the vehicle toward this point by calculating the necessary steering angle.

```
def lateral_controller(x, y, yaw, target_waypoint, future_unreached_waypoints):
    # TODO
    return target_steering
```

For an in-depth explanation of the Pure Pursuit Controller, please refer to this Link. While the derivation of the underlying mathematics is not a requirement for this MP, acquiring a fundamental understanding of the Pure Pursuit Controller concept will be advantageous for future exams and projects.

In summary, to determine the steering angle for your vehicle, you'll need several key parameters: the lookahead distance ld (the distance between your vehicle's current position and the lookahead point), the angle α between the vehicle's heading and the look-ahead line (the line connecting the vehicle and the

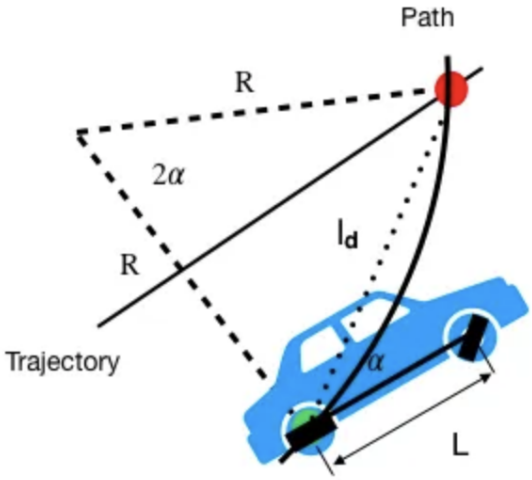

Figure 3: Pure Pursuit Geometric relationship

lookahead point), and the vehicle's wheelbase L (a constant that has been provided in the code). The steering angle α can be calculated using the following formula:

$$\delta = \text{arctg}\{\frac{2L * sin(\alpha)}{ld}\}$$

This formula allows you to translate the given variables into a specific steering angle, guiding your vehicle toward the lookahead point. Note that target waypoints are static points we provide to you, that you have to reach; and lookahead point is an imaginary points on the reference trajectory that's up to you to choose.

Here we propose 3 methods on how to choose the lookahead point to follow, but you can come up with your ideas as well.

1. Directly choose the closest target_waypoint
2. Set a constant lookahead distance and interpolate between future waypoints
3. Set a dynamic lookahead distance (relative to vehicle state) and interpolate between future waypoints

3.4 Metric

In this MP, we would like you to measure your designed controller using the below 2 metrics. These two metrics will be checked by the autograder.

1. **Safety**: Number of waypoints your controller can successfully follow.
   You have 4 simulation seconds to reach each waypoint, and you have to reach each waypoint in sequential order, elsewise we will treat you as deviating from the track.
2. **Efficiency**: Time to finish(TTF) the whole track.
   While maintaining safety, we also want you to be efficient in reaching all waypoints, thus your total simulation TTF should not exceed 130 seconds.

3.5 Tips for developing

If you would like to test out specific part of the track (E.g. start at the i-th waypoint just to test the sharp turn), you could comment out the first i waypoints in the waypoints.py and set the start position at the (i+1)th waypoint using method we specified at Section 3.1.3

3.6 Autograding (20 pts)

Your controller should be robust enough to handle some small uncertainty around the starting point. Therefore, during the autograding, we will randomly choose a starting condition (x, y, yaw) from this set $x \in [0, 3], y \in [-97.5, -98.5], yaw \in [-0.1, 0.1]$. (Note that it might differ from your local test, and you can set different initial conditions using set_pos.py)

In this assignment, we prioritize safety first, so if you fail to reach all waypoints, the maximum you can get is 90% * 20pts. In addition to safety, we also want you to be efficient in reaching the destination, thus we apply a penalty if your simulation TTF is above 130 seconds. We do not set hard limitation on acceleration values, but you will need to analysis it for Problem 5.

```
wp = % of successfully followed waypoints
penalty = 0.1 TTF >= 130(threshold) else 0
autograded_part_score = 20 * (1 - penalty) if wp == 100% else 20 * min(0.9, wp)
```

Important Notes on the MP2 Autograder:

1. All functions, except for the three task-specific functions, will be overwritten by default functions during the autograding process.
2. Note that there is a distinction between simulation time and real-world calendar time. For example, while 10 seconds may pass in real time, only 1 second may elapse in the simulation. We utilize simulation time to evaluate your efficiency metrics. Refer to `main.py` for more details.
3. When submitting your code to the autograder, please remove all print, plot, and log functions, as these can adversely affect the simulation's time-to-finish (TTF). However, you are free to modify the code when running it locally.
4. The 3 groups with the lowest TTFs on the autograder will receive +2 points on this MP

3.7 Report

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

Problem 4 (15 points)**. Longitudinal Controller**: How do you establish the relationship between the track's curvature, the vehicle's current state, and its target velocity? Could you elaborate on the methodology employed to derive this mapping? Specifically, how have you optimized the process for efficiency while ensuring safety?

Problem 5 (15 points)**. Comfort Metric Analysis**: After your controller successfully completes one lap within the specified time, please generate an acceleration-time plot. Take note that accelerations exceeding 0.5G (or 5 m/s^2) may result in discomfort for passengers. Did your controller cross this threshold? If it did, how frequently did this occur, and what were the contributing factors? We have provided a flag in controller.py to enable/disable acceleration logging, provided you implement Task 1 correctly.

Problem 6 (15 points)**. Lateral Controller**: Could you elaborate on the criteria used to select the lookahead target waypoint? How many of the suggested methods did you explore for this purpose? Among those, which method is the most effective, and why do you think so?

Problem 7 (15 points)**.** Draw an x-y plot recording the trajectory of the vehicle around the track. In addition, you should mark the default initial position and the waypoints in your plot.

Problem 8 (10 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.

Problem 9 (10 points)**. Demo.** For this MP, you will need to demo your code to the TAs in lab sessions on **October 3**, and be prepared to be asked MP related questions.

3.8 Submissions

The maximum one can get from this HW is 100 pts (Problem 1-3). The maximum one can get from this MP (Problem 4-9) is 100 pts.

1. (100pts) Everyone submits individual `<netid>_ECE484_HW2.pdf` on Canvas
2. (70pts) Only one group member needs to submit `MP2_<groupname>.pdf` on Canvas
3. (20pts) Only one group member needs to submit `controller.py` on auto-grading site
4. (10pts) Everyone needs to attend demo in their sections, and this part will be graded individually

1-3 are due 11:59pm CST 02/28. 4 is due by the time of your discussion section.