ECE 484: Principles of Safe Autonomy	Spring 2022
Written by: ECE 484 Team	*MP4: Autonomous Pipeline*
Website 	*Due Date: Apr 8, 2022*

1 Introduction

In this MP, you will need to combine the knowledge you gained from the previous MPs to implement an autonomous vehicle to run on the track and avoid obstacles.

Same as the previous MPs, this MP is divided into two different parts. In the first part of this MP (section 2), you will be working on some theoretical problems about planning and reinforcement learning. In the second part of this MP (section 3), you will be provided with a basic autonomous agent pipeline that contains perception, decision and planning and low-level control modules and you will need to implement the modules so that your Autonomous agent can perform lane tracking on a race track while avoiding obstacles (other stationary vehicles). Note that we will also do a friendly competition between groups: the top three groups whose car completes a lap around the track the fastest will receive bonus points.

For the written problems, you will individually submit the solution to Problems 1-3 in one file called `MP4_netid.pdf`. For the report, your group will need to submit a single file `MP4_group_name.pdf`. 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

- Perception
- Control
- Decision

System requirements

- Ubuntu 16
- ROS Kinetic

2 Written Problems

Problem 1. A* search (10 points)

(a) Construct an example of a graph G with at least 6 nodes and 10 edges where A^* search fails to find the optimal path. Clearly mark the values of the heuristic function h at each vertex, the *start* and the *goal* vertices, the optimal path. Then show the step-by-step execution of the algorithm in a table to illustrate the sub-optimal path that the algorithm finds. *Do not use the graph from the lecture slides or search for an example online.*

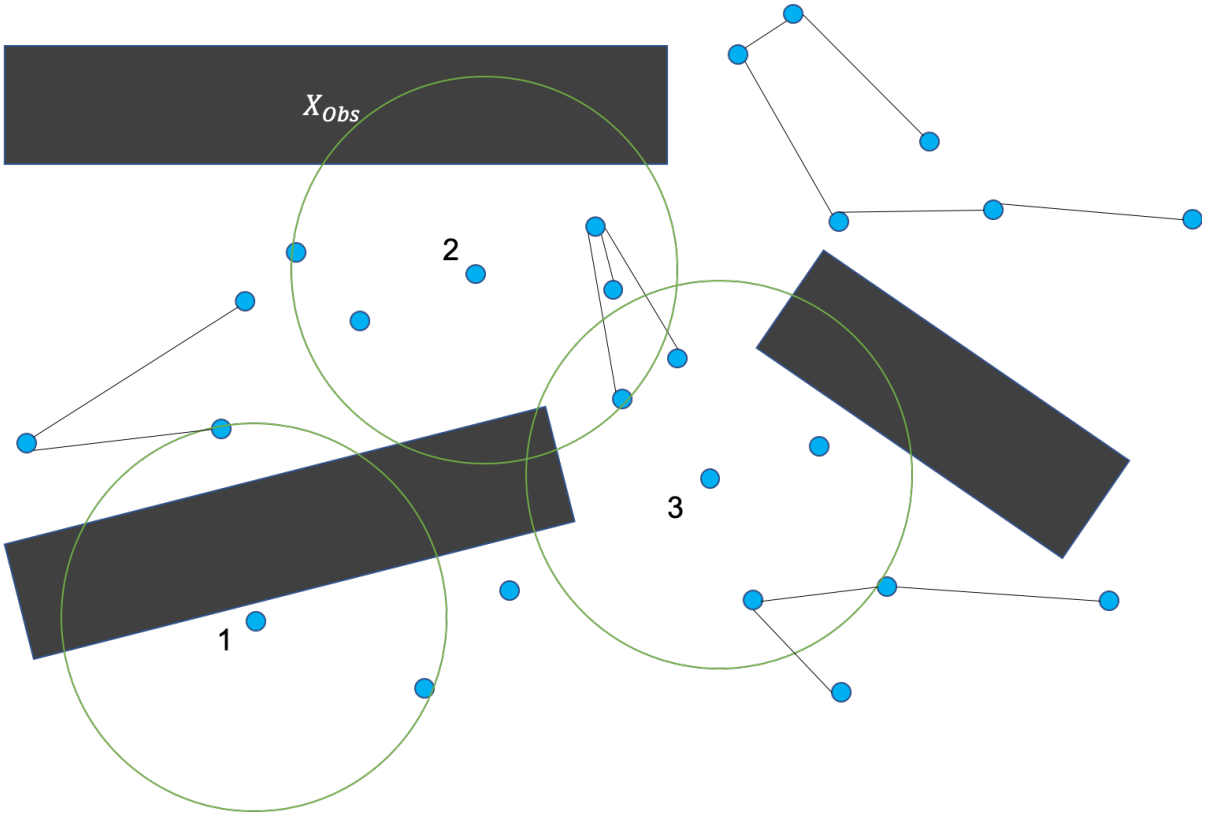
(b) Change the heuristic function h on the above graph G to an admissible heuristic. Illustrate the execution of A^* on this changed graph.

Problem 2. Probabilistic Road Maps (10 points)

(a) Recall the Probabilistic Road Maps algorithms from lecture. In particular, recall that the function $Near(G = (V, E), v, r)$ returns the vertices in V that are within r distance of v.

Consider the partially constructed PRM shown in the figure below. Suppose the next three vertices chosen in the outer for loop are marked by 1, 2, and 3, and the corresponding r-radius circles is shown. Suppose also that *CollisionFree* paths are found by drawing straight lines. Then, draw the edges that are added to he PRM graph after these vertices are processed.

(b) What are some of the advantages of PRMs?



3 Vision-based Navigation with Collision Avoidance

In this part of the MP, you will need to implement an autonomous agent to run in the Gazebo environment. This environment, as shown in Fig. 1, is very similar to the race track you see in MP1 and MP2; the difference is that this race track contains various obstacles. Similar to previous MPs, you will continue to

work on the GEM car. Your goal this time is to apply the knowledge and concepts learned in this course to build an autonomous-driving pipeline to the GEM car. In this MP, no predefined waypoints will be given for tracking. Instead, the vehicle should be able to navigate safely merely based on the perception signals, e.g., images and LiDAR point clouds. You may find the lane detection in MP1 and the controller design in MP2 useful to complete this MP.

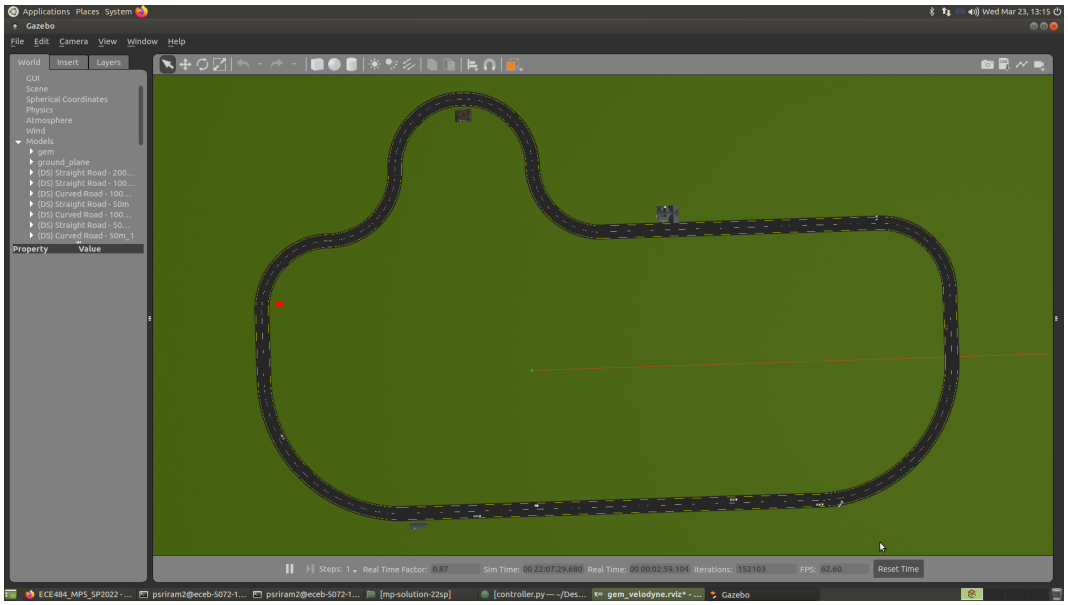


Figure 1: An example environment for this MP.

3.1 Module architecture

This section describes components that are important for this MP. The provided code constructs a basic pipeline including perception, decision and planning and low-level control to drive the GEM vehicle. **Feel free to modify any parts listed below** to increase the performance of your autonomous driving pipeline. Here is a demo video for this MP.

All the files of basic pipeline are located at ./src/mp4/src.

3.1.1 Perception

This module is located in ./perception/perception.py. As its name implies, this is the perception module for the GEM vehicle. The perception module collects and processes the data from sensors on the GEM vehicle. Currently, the available sensors are a front facing camera, LiDAR, IMU, and GPS. You will use your lane detection module from MP1, as well as the LiDAR and GPS sensors as the main components of your perception module. Specifically, you will use the bird's eye view image from the lane detection module to calculate two errors: the lateral error between the vehicle position and the center line of the lane and the error in the vehicle heading with respect to the lane heading. In addition, you will use the LiDAR reading in the box $x \in [0, 20]$, $y \in [-2, 2]$, $z \in [-1.6, 0.5]$ with respect to vehicle frame in front of the vehicle and calculate the average position of the LiDAR point clouds in that box. The position is then projected to the x-y plane by dropping the z component and the distance between vehicle and the position is computed. This position of the LiDAR point clouds will be used to detect any obstacles in front of the vehicle. Finally, the GPS readings can provide the current state (pose and twist) of the vehicle.

ROS topics for sensors on GEM car:

```
LiDAR: /velodyne/points
Camera: /gem/front_single_camera/front_single_camera/image_raw
GPS*: /gazebo/model_states
IMU: /gem/imu
```

3.1.2 Decision

This module is located in ./decision/decision.py. This module will contain the decision module of the GEM vehicle. It should take results (e.g. the LiDAR readings or the GPS readings) from the perception module and decide what the car should do next. Currently, the decision module will receive the LiDAR reading discussed in Section 3.1.1, the GPS reading describing the state of the vehicle, and the lateral and heading errors extracted from the lane detection module. These parameters can be used to inform the decision module e.g. if the LiDAR reading falls below some threshold (indicating the presence of an obstacle in front), initiate a lane change.

This module provides core functionality to your autonomous driving pipeline. You may apply knowledge learned in lectures and previous MPs here or explore other techniques.

3.1.3 Control

This module is located in ./controller/controller.py. This module contains a low level controller to control the vehicle motion. This controller should be very similar to the controller you implemented in MP2. Typically, it takes the reference from the decision module and the current state of the vehicle and compute the required values to run the vehicle. In the current implementation, it accepts the target velocity, the lateral error, and the lane heading error. As previously done in MP2, these three values can be used in combination with PD control to calculate the desired velocity and steering angle.

3.2 Development instructions

For this MP, there's no gold solution. You will need to develop your own algorithm to solve the problem of running on the track safely. While we have provided an initial framework, feel free to modify any of the code to try your own algorithms. In the next few subsections, we will describe one potential approach to build an autonomous pipeline that you may follow. Note that it is useful to look at `main.py` to observe how all the components are connected.

3.2.1 Perception

For the perception module, you will need to first paste your MP1 implementation into the lanenet_detector class in `perception.py`. The functions used from your implementation will be `gradient_thresh`, `color_thresh`, `combinedBinaryImage`, and `perspective_transform`. You will also need to add your implementation of `line_fit.py` to the same directory as `perception.py`.

The provided VehiclePerception class has three functions:

- **cameraReading** returns the lateral error and the lane heading error.
- **lidarReading** returns the distance between the vehicle and object in front, calculated using the LiDAR point cloud data.
- **gpsReading** returns the state of the vehicle (position, orientation, linear velocity, angular velocity).

You will need to modify the `detection` function to calculate the lateral error and the lane heading error based on the bird's eye view image.

Let us speak in terms of the image coordinate frame (not vehicle coordinate frame). To calculate the lateral error, assume the x-position of the car to be in the middle of the image (roughly indicated by the purple rectangle in Fig. 2. The lateral error refers to the difference between the x-position of the car and x-position of the center of the lane at the bottom of the bird's eye view image. The x-position of the center of the lane can be calculated using the two lines fitted to each lane - if you remember, these two lines are calculated in MP1 as well. It is important to then convert from the pixel space to actual distance, which can be done using a provided constant `meter_per_pixel` in the lanenet_detector class. To calculate the lane heading error, you must take the derivatives of the two fitted lines and evaluate them at some y-position. To dampen any oscillating behavior, pick a y-value well ahead of the vehicle. The two derivatives at this specific y-value can then be used to calculate the lane heading errors and then averaged to find the mean heading error. The lane heading error is referred to in Fig. 2 by θ.

Hint: You need to use a proper perspective transform so that a valid bird's eye view image is returned, i.e., the two lanes should be parallel to each other in the bird's eye view image.

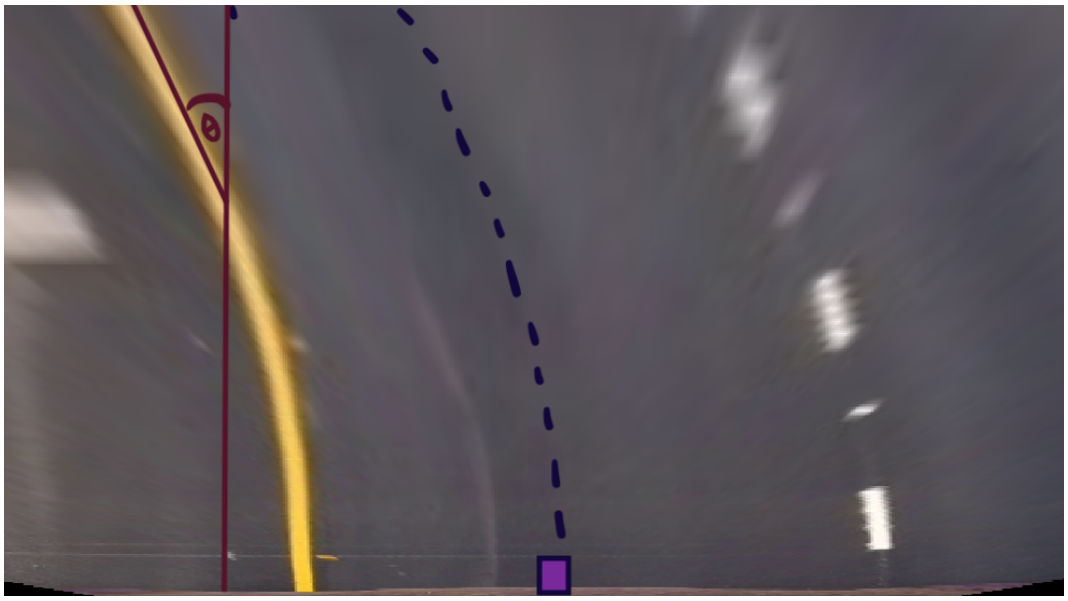


Figure 2: A rough sketch for lateral error and lane heading error

3.2.2 Decision

For the provided decision module, you will need to complete the `get_ref_state` function in `decision.py`. This function accepts the current state of the vehicle (GPS reading), the current distance between the vehicle and the obstacle in front (LiDAR reading), and the two error values - lateral error and lane heading error - computed in the perception module. This function will then return three values: the reference velocity, a modified lateral error, and the lane heading error.

In this decision module, we define four possible "vehicle states": lane keeping, start lane changing, stabilizing, and emergency stop. You will initialize the vehicle to one of these states, and transition between these states based on certain conditions. Moreover, you will also set the reference velocity and modify the lateral error based on the current state and miscellaneous conditions. The description of the states are as follows:

- Lane keeping: This state corresponds to the vehicle simply following its current lane and is a good

initial state for the vehicle. You may still want to control the reference velocity in some way, for example decrease the velocity if the road is curvy or an obstacle is detected.

- Start lane changing: This state corresponds to the vehicle changing lanes after an obstacle has been detected. The conditions for this state will be the most difficult to write. It is crucial to keep in mind that **lateral error** corresponds to the difference between the current vehicle position and the center line of the current lane. In order to get the vehicle to change lanes correctly, you may need to modify this lateral error based on the provided lateral error and the lane width (provided as a member variable of the VehicleDecision class).
- Stabilizing: This state corresponds to the stage where the vehicle "stabilizes" to the current lane after changing lanes. This essentially refers to ensuring that the car is driving straight once again after changing lanes.
- Emergency stop: This state corresponds to the vehicle coming to an emergency stop in the case it will not be able to avoid an obstacle. Ideally, this state should never be reached but it is still worthwhile to implement.

Based on the aforementioned description of the states, you will need to complete the implementation of the decision module.

3.2.3 Control

For the provided controller, you will need to complete the `execute` function in `controller.py`. The inputs to the function are the target velocity, lateral error between the vehicle and the center line of the lane, and the lane heading error. You will need to perform PD control similar to MP2 for the lateral error and lane heading error in order to compute the desired steering angle. The provided target velocity may be used directly as the desired velocity depending on your implementation of the decision module. These two control inputs should then be published to the vehicle model.

3.2.4 Editing world in Gazebo

You can edit the world using Gazebo GUI to generate different scenarios to test the algorithm you implemented. You can use the following buttons as shown in Fig. 3 to move or rotate a selected model. You can also copy and paste models to add more obstecles or create more complicated trajectories. You can save the model in ./src/gem_simulator/gem_gazebo/worlds for future use and you need to modify the launch file to point to the proper world. Note that when saving the model, you may need to consider deleting the vehicle model.

Figure 3: The buttons in the red box from left to right are selection mode, translation mode and rotation mode.

3.2.5 Running the MP

To run the MP, run the following commands:

```
roslaunch mp4 mp4.launch
```

```
python main.py
```

3.3 Report

Problem 1 (25 points)**.** Describe the algorithm you are using for each module. You need to describe each component of your design specifically potentially with pseudo code or diagrams.

Problem 2 (25 points)**.** Record a video of your GEM car driving on the race track and include the link to the video in the report.

3.4 Race

During the demo for this MP, there will also be a friendly competition between groups to see whose vehicle is the fastest while evading all obstacles. Based on how quickly the race car finishes a lap in the track, the rankings for the groups will be created and extra credit will be assigned.

- The Champion will receive 25 points of extra credit
- 2nd place will have 20
- 3rd place will have 15
- 4th place will have 10
- 5th place will have 5

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 Canvas **individually** with filename `MP4_netid.pdf`.

For the report in Section 3.3, each group should write a report that contains the solutions, plots, and discussions. This report should be submitted to Canvas with filename `MP4_group_name.pdf`. In addition, each group should submit the code to Canvas.