CS 747: Programming Assignment 3

Total marks: 15

(Prepared by Thomas Jacob and Bhavini Jeloka)

This assignment tests your understanding of the concepts taught in class and its applications to real world problems. In particular, we examine one of the most popular domains: **autonomous driving**. You will have to come up with a controller to drive a car, and are free to use any approach: coding it up yourself and/or learning with value function-based approaches and/or using policy search.

There are two tasks in this assignment. In Task 1, you will construct a controller that can navgiate a car out of a parking lot on a winter night. The surroundings are icy and the parking lot is also close to freezing. It is your responsibility to get the car out of this situation and on the road safely. Task 2 is along similar lines but you find yourself in a stickier situation. The parking lot is now filled with pools of mud from a cold winter storm. You must dodge these obstacles and find a safe way out.

All the code you write for this assignment must be in Python 3.8.10. The libraries that you require come installed with the docker image that has been shared for the course, and are already imported in the files you need to complete.

Code Structure

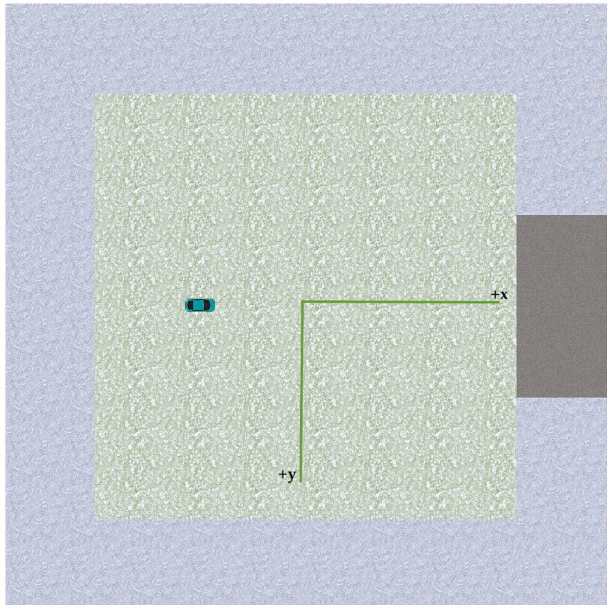
This compressed directory has all the files and folders that are required for the assignment (we acknowledge Wesley Hsieh for a base simulator that our assignment is built upon). For the most part, you do not need to worry about these files. It is, however, advised that you read through `driving_env.py` in the `envs` folder to understand how the `_step` function works. In short, it returns the reward, termination condition, the next state and a boolean flag to indicate the success of the episode (it also returns a dictionary with additional information that can be ignored). In addition to the above, the `_reset` function of the simulator resets the environment and returns the initial state of the car.

We have provided `run_simulator.py` to run simulations and visually see the working of your controller, which you'll have to submit as described later. This code has been provided with 10 random seeds that evaluate your controllers for a fixed few environments (different initial states of your car and different positions of the obstacles in Task 2) and outputs the execution time. The only file you need to edit is `run_simulator.py`. Do not edit any other files.

For evaluation, we will use another set of environments in the autograder, and use its score as is for the evaluation. See the exact details below.

Task 1: The Parking Lot Problem

As mentioned above, your task is to navigate your 50 x 25 car out of a 700 x 700 square grid. It must exit onto the road whose entrance has its centre located at (350, 0). This grid is centred at the origin with the coordinate axes shown below (screen size: 1000 x 1000). It is an episodic task that begins with your car initialised at a random position and orientation; the episode ends when you navigate out or bump into a wall (or pit in Task 2). We also cap episodes to a total of 1000 time steps.



Your car has 4 state features: the x and y position coordinates, the velocity and the heading angle. In order to control the state, you have two independent control inputs **steer** and **acceleration** that take the following discrete values.

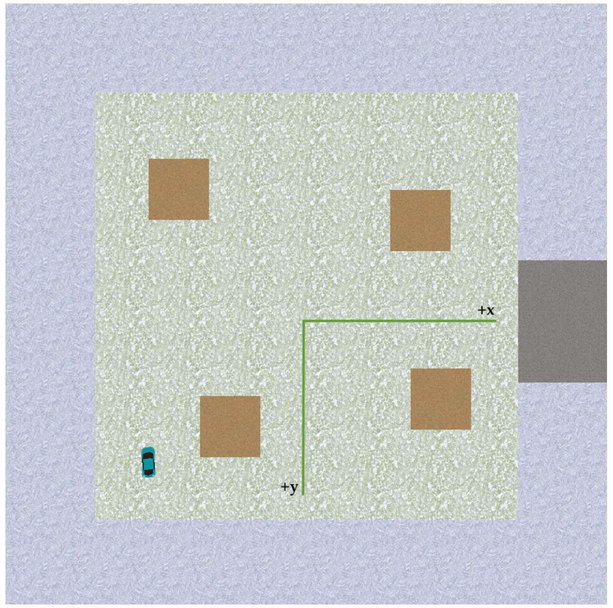
	Control Input: Steer
0	-3 degrees
1	0 degrees
2	3 degrees

	Control Input: Acceleration
0	-5 units
1	-3.95 units
2	0 units
3	3.95 units
4	5 units

The class `Task1` has been created for you inside the `run_simulator.py` file. The `next_action` function should determine the action to be taken in order to reach the next state while the function `controller_task1` integrates these sequences of actions to complete the entire navigation task. Please read the comments in the code in order to understand the components that can and cannot be modified under `controller_task1`. You can come up with any technique to reach the road and marks will be given as long as your method works for all random initialisations that we will test it for. Feel free to add helper functions and other initialisations to support your code.

Task 2: The Parking Lot Problem Intensifies

Building on the first task, your goal now is still to navigate your 50 x 25 car out of a 700 x 700 square grid. It must exit onto the road whose entrance has its centre located at (350, 0). However, this time, the road is narrower and the parking lot is filled with 4 randomly located pits of mud (with a size of 100 x 100 each). The parking lot is once again centred at the origin with the coordinate axes shown below (screen size: 1000 x 1000).



The class `Task2` has been created for you inside the `run_simulator.py` file. States and actions are as in Task 1. The `next_action` function should determine the action to be taken in order to reach the next state while the function `controller_task2` integrates these sequences of actions to complete the entire navigation task. Please read the comments in the code in order to understand the components that can and cannot be modified under `controller_task2`. You can come up with any technique to reach the road and marks will be given as long as your method works for all random initialisations that we will test it for. Feel free to add helper functions and other initialisations to support your code. You will need to brainstorm on obstacle avoidance manoeuvres.

Running the Code

Since it is important for you to be able to visualise your agent's behaviour as you develop it, we had provided a visual interface. However, this might not run as is on the docker, and you will have to run the code on your local system so that a new game window is allowed to open with the `render_mode` flag. Without the flag, however, since there will be no rendering, you can run your code in the docker container. If you want to run on your local system, you just have to run `pip install -e .` from the root directory `gym_driving_dir` so that the dependencies from `setup.py` get installed automatically before you run the scripts. To run on docker, however, you will need to follow additional steps. Firstly, run `pip install -e .` from the root directory `gym_driving_dir` as before (do not worry if some warnings are given). From within the docker, running OpenCV throws an error `ImportError: libGL.so.1: cannot open shared object file: No such file or directory.` To circumvent this, you have to install libgl1 by `apt-get update && apt-get install libgl1 .`

To run the code you have written run `python run_simulator.py --task T1/T2 --render_mode` (use this while running code in your local system as mentioned) to visualise the simulation and `python run_simulator.py --task T1/T2` (only this command will work inside docker) otherwise from the `simulator` directory.

Now to verify the code against the 10 random seeds run `python autograder.py --task T1/T2` from the `simulator` directory. To pass the autograder, you should not put any additional print statements in the code (comment them out if you used them for debugging).

Evaluation

5 marks are reserved for the correctness and performance of your code on each of the two tasks, and 5 marks for your report.

For each task, 2.5 marks are for your performance on the given (10) random seeds, and 2.5 marks will be for unseen (10) random seeds. The 2.5 marks are divided into 1 mark for successfully exiting the parking lot on all runs within 1000 time steps. The remaining 1.5 marks will depend on the number of random seeds for which you exit within a shorter span, which is coded into the autograder. Thus, for example, if your agent is always successful, and clear 8 of the 10 initialisations within the specified threshold, you will receive (1 + 8 * 0.15 = 2.2) marks out of 2.5.

Unlike the previous assignments, you have been given a free hand to come up with your controller. Hence, we would like to see a clear presentation of your approach. Include a file named `report.pdf` that spells out the ingredients of your solution, any intermediate experiments that may have guided your decisions, learning curves if you did use some form of learning/parameter tuning, and so on. If your report is not sufficiently clear and informative, you will stand to lose marks.

The TAs and instructor may look at your source code and notes to corroborate the results obtained by your program, and may also call you to a face-to-face session to explain your code.

Submission

You have to submit one tar.gz file with the name (roll_number).tar.gz. Upon extracting, it must produce a folder with your roll number as its name. It must contain a `report.pdf` - the report as explained above, and one code file: `run_simulator.py`. You must also include a `references.txt` file if you have referred to any resources while working on this assignment (see the section on Academic Honesty on the course web page).

Deadline and Rules

Your submission is due by 11.55 p.m., Sunday, November 6. Finish working on your submission well in advance, keeping enough time to test your code and upload to Moodle.

Your submission will not be evaluated (and will be given a score of zero) if it is not uploaded to Moodle by the deadline. Do not send your code to the instructor or TAs through any other channel. Requests to evaluate late submissions will not be entertained.

Your submission will receive a score of zero if your code does not execute on cs747 docker container. To make sure you have uploaded the right version, download it and check after submitting (but before the deadline, so you can handle any contingencies before the deadline lapses).

You are expected to comply with the rules laid out in the "Academic Honesty" section on the course web page, failing which you are liable to be reported for academic malpractice.