Final Project: Reinforcement Learning for Motion Planning and the Control of Dynamical Systems

Purposes

- Basic concepts of Reinforcement Learning (RL).
- Off-policy Temporal-Difference Learning (TD).
- Tabular methods for TD.
- Control of Dynamical Systems.

Implementation Tasks

Problem 1: Cruise Control (20 points)

Automatic cruise control is an excellent example of a cyber-physical system which is the model of a physical process regulated by a discrete-time controller. The purpose of the cruise control system is to maintain a constant vehicle speed. This is accomplished by measuring the vehicle speed, comparing it to the desired or reference speed, and automatically adjusting the throttle according to a control law.

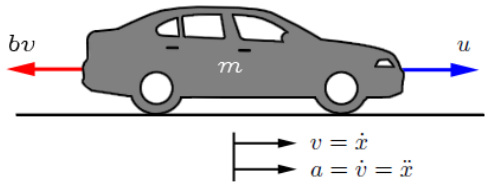


Figure 1: Cruise control

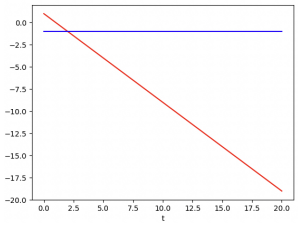


Figure 2: Trajectory when there is no control. Red: position x. Blue: velocity v.

We consider a variant of the cruise control problem, that is the *automatic parking* problem which requires to find a sequence of control inputs, i.e., accelerations, to stop the car at a given position. We use the following ODE to model the car's dynamics.

$$\dot{x} = \frac{dx}{dt} = v, \qquad \dot{v} = \frac{dv}{dt} = u \tag{1}$$

The system has two state variables: $x(t)$ and $v(t)$ which indicate the current position and velocity respectively at a time $t \geq 0$. The control input is denoted by u, i.e., the acceleration/deceleration, it can only be changed every 0.1 seconds to regulate the car's position and velocity. It is a deterministic system.

Range of the States. In order to make our RL training tasks easier, the system state is bounded as $x \in [-5, 5]$, $v \in [-5, 5]$ at all time. Any state outside of the range is not under our consideration for training.

Episode. An episode of the system is a trajectory $s_0, s_\delta, s_{2\delta}, \ldots, s_{k\delta}, \ldots$ wherein $\delta = 0.1, \ldots, s_{i\delta} = (x(i\delta), v(i\delta))$. Given an initial state s_0, the generation of the episode requires to solve the ODE (1), we may use the following Python function to compute the next state of a given state:

```
def model(s,t,u):
    dsdt = [s[1], u]
    return dsdt

delta = 0.1                    # should be defined only once
step = np.linspace(0, delta) # should be defined only once
y = odeint(model, s0, step, args=(u_i,))
```

The program uses the SciPy `odeint` function (link) to numerically compute the reachable state of the system under an control input `u_i`. By repeatedly run this program by N times, you are able to generate a trajectory of N time steps, i.e., $t \in [0, N\delta]$. Figure 2 illustrates the projections of the trajectory from the state $s_0 = (1, -1)$ over the time horizon $[0, 20]$ when the control input is 0 in all time steps.

Q-Table. We are going to design and implement an RL framework which produces a feedback law represented as a Q-table, Q. The Q-table is represented as a 3-D matrix, it is an approximation of the action-value function for the best policy which regulate the car's state to $(0, 0)$, i.e., parking the car in the position of $x = 0$. Given a state $s = (x, v)$, the Q-table entry $Q(x, v, u)$ tells the estimate return of an action u performed in s for the next 0.1 time.

State Space Discretization. Even our bounded state space range has infinitely many states, and it is impossible to use a finite size Q-table. Therefore, we consider to discretize the range of $[-5, 5] \times [-5, 5]$ to only consider finitely many references for (x, v) to find control inputs. As an example, you may discretize the state space by

```
X = np.linspace(-5, 5, 21)
V = np.linspace(-5, 5, 21)
```

i.e., uniformly choose 21 references for x and v respectively.

Reward. The reward should be carefully designed to make the RL process effectively learn the estimate returns and organize them in the Q-table.

You are required to complete the following tasks regarding to the above automatic parking problem.

1.1 **(4 points) Write a Python class for modeling of the environment, i.e., the ODE model of the car.** The class should at least have the following functions: (2 points) interacting with the agent and (2 points) generating a trajectory from a given initial state regarding to a given Q-table (feedback law to determine the control input in every time step).

1.2 **(2 points) Explain how you define and implement the rewards to maximize the RL performance.**

1.3 **(8 points) Write a Q-learning process in Python to train a Q-table which can regulate the car to the state** $(0, 0)$**.** You may consider the following value space for the control input:

```
control_inputs = np.array([-5, -1, -0.1, -0.01, -0.001, -0.0001, 0,
                           0.0001, 0.001, 0.01, 0.1, 1, 5])
```

You are also required to write an agent class which stores the ML model and interacts with the environment. You may find a proper value for ϵ by tuning and use it as the probability of exploration. *Please use a progress bar for training.*

1.4 **(2 points) Model evaluation.** Please explain how you would like to evaluate the quality of a Q-table which is obtained from your RL learning process. Your method is required to reflect the accuracy of the policy (Q-table) and comprehension on the episodes. Please keep the method simple and address the requirements based on the concepts of Q-learning.

1.5 **(4 points) Summary of experiments.** Comprehensively test your RL method by considering 3 different discretization schemes (e.g., size = 21, 51, 101), 2 different learning rates, 2 different discount factors, and 3 different (combined) settings for the number of episodes and their maximum length. You are also required to evaluate, compare and explain the qualities of the Q-tables you obtained using those settings.

As an example, Figure 3 illustrates the trajectory of the car generated by using the Q-table from the RL process with a 101×101 discretization, $10,000$ episodes whose length are bounded by 500.

Problem 2: Finding the Shortest Feasible Path (20 points)

We are going to implement a program of using reinforcement learning for motion planning in a 2-D map. It can be divided into the following main steps.

2.1 **(4 points) Map abstraction.** The task requires to transform/abstract a given map to a matrix data structure. The original maps are BMP files which can be loaded in Python to a NumPy array. We assume that the maps only have two colors (black and white). Such a map can be simply transformed to a matrix whose entries represent the pixel values. An obstacle in the map can be represented by black pixels, and other white pixels represent the enterable areas. The main issue of this simple method is that the number of entries in the resulting matrix is often tremendous. For example, the simple map given in Figure 4a is of the size 528×532

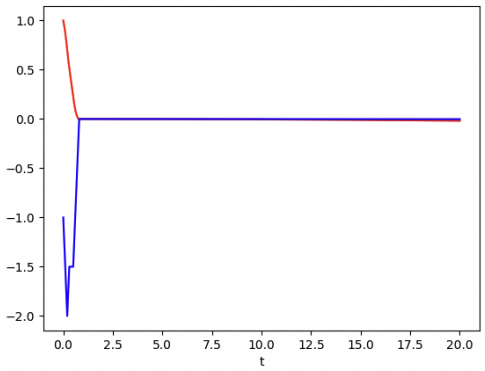
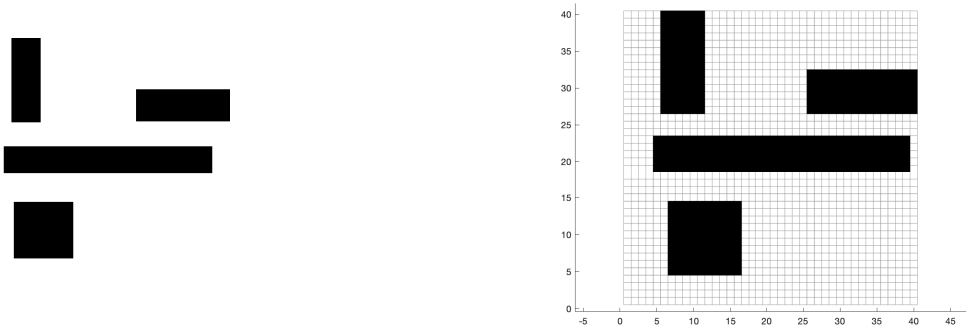


Figure 3: Trajectory of the car with a successful control strategy.



(a) Original Map (b) Abstracted Map

Figure 4: Example of map abstraction

and it is obvious that it is unnecessary to use so many entries to represent the obstacles since they are of simple shapes.

Therefore, we seek to compress the matrix data structure for maps. To do so, we may use an approach similar to pooling. For example, if the given image is of the size 600×600 and we want to compute an abstraction of the size 60×60, then we may uniformly subdivide the given map to 60×60 many 10×10 blocks, each of them is white if there is no black pixel included, otherwise black. Figure 4b shows the 40×40 abstraction of the map in Figure 4a. If the size of the original map is not divisible by the abstraction size, then the tail pieces are treated in the same way as the normal pieces.

2.2 **(3 points) Building environment.** The task requires to write a Python class that simulates the moves in a map abstraction. Four actions are considered: *left*, *right*, *up* and *down*. Given an agent state along with an action, an interaction with the environment object is required to return the next state along with a reward. You are required to design/define the reward to maximize the performance of RL. In the simulation, you are not allowed to enter an entry which is an obstacle or go out of the map.

2.3 **(3 points) Building agent.** You are required to write a Python class for the agent which keeps a Q-table. At least a function for the interaction with an environment should be implemented. It performs an interaction with an environment and updates the Q-table. As it is explained in the class, the Q-table should be a 3-D matrix.

2.4 **(4 points) Q-learning process.** The task requires to implement a Q-learning function which repeatedly performs agent-environment interactions for a specific number of episodes which are length-bounded. The function should have at least the following parameters: learning rate, discount factor, number of episodes, the maximum length of episodes. You may find a proper value for ϵ by tuning and use it as the probability of exploration. *Please use a progress bar for training.*

2.5 **(2 points) Model evaluation.** Please explain how you would like to evaluate the quality of a Q-table. Your method is required to reflect the accuracy of the policy (Q-table) and comprehension on the episodes. Please keep the method simple and address the requirements based on the concepts of Q-learning.

2.6 **(4 points) Summary of experiments.** Comprehensively test your RL method by considering 2 different learning rates, 2 different discount factors, and 4 different (combined) settings for the number of episodes and their maximum length. You are also required to evaluate, compare and explain the qualities of the Q-tables you obtained using those settings. Please perform your tests on the given two maps.

Report

You are required to write a report for this assignment. It should include the following parts:

- For Task 1.1, 1.2, 1.3, 2.2, 2.3, 2.4, you are required to explicitly answer the questions and explain the methods and algorithms implemented.
- For Task 1.4 and 2.5, you are required to explicitly explain your model evaluation method.
- For Task 1.5 and 2.6, the evaluations should be done by the methods you proposed for 1.4 and 2.5 respectively. The experimental results should organized in a way that is sufficiently clear for quality evaluation and comparison. Using tables is a good idea. For each problem, please discuss the following aspects, their relations and possible impacts on the learning performance:
  - Size of the state space;
  - Number of episodes explored in training;
  - Maximum length of episodes used in training;
  - Lengths of the test episodes;
  - Learning rate and discount factor.

You are encouraged to add more observations and thoughts.