COMP0037 2024/25 Coursework 2

COMP0037 Teaching Team

14th March 2025

Overview

- Coursework 02 Part 01 Release Date: Friday, 14th March 20245
- **Assignment Due Date: Wednesday, 23rd April (16:00 UK time) 2025**
- Weighting: 60% of the module total
- **Final Submission Format**. Each group must submit *two* things:

    1. A zip file (which contains the source code implemented to tackle this coursework). The name of the zip file must be of the form `COMP0037_CW2_GROUP_g.zip`, where `g` is the letter code of your group.
    2. A report in PDF format. It will be named `COMP0037_CW2_GROUP_g.pdf`.

    Note that a penalty of 2% will be applied if the files are submitted with the wrong name.

The total marks for each question are written in the form [X marks].

Approximately 67% of the total coursework marks are from your analysis and 33% from your coding of the algorithms. Therefore, please take time and care with your writing and analysis of solutions.

For the submitted code, you must use the notation variable names and code provided in the lectures. If you do not use these, we will consider the work a potential plagiarism case and investigate further.

You will need to implement additional routines to support the functionality required in these questions. The code to implement this is available on Moodle. It is based upon but extends and refactors the code provided in the labs. The general areas requiring adjustment will be shown using comments.

As discussed in Lecture 00 (Overview), we expect you to take care in writing your report. For example, figures and graphs must have captions and be numbered and labelled. Equations captured by a screenshot, from the slides or elsewhere, and pasted into the document are of low quality and are likely to cause a penalty for your marks. When we ask you to write some code, provide an explanation in your report of what you wrote.

The final mark will be computed by summing all the marks awarded on individual questions and dividing by the mark total.

The code will not be independently marked but must be provided before the submission deadline. It might be checked, and if it does not run or produce the results presented in the report, it can impact your marks.

Use Case

The use case continue the scenario you encountered in Coursework 01. A robot has been tasked with automatically cleaning various terminals in an airport. Airports are generally very busy places and require constant cleaning.

In this coursework, we will extend this in two ways:

1. Part 1: Use model-free methods to implement a low-level grid-based planner. This will use discrete tabular methods to plan the overall trajectory of the robot.

2. Part 2: Use model-free methods to implement the low-level control of the robot. This will use continuous-valued variables, and so function approximation must be used.

Note that we have had problems getting Part 2 to work properly over the past couple of years. If we are still unable to get it to work, the coursework will only consist of Part 1 and marks will be scaled accordingly. The decision about whether to release Part 2 will be made by Friday, 21st March 2025.

Part 1

The airport still hasn't learned its lesson and is still using the noisy robot from Coursework 1. As a reminder, when commanded to move in a given direction, it is not guaranteed to move in that direction. Rather, as illustrated in Figure 1, the robot moves in the nominal direction with a probability p or one square on either side of the nominal direction with a probability $q = 0.5(1 - p)$.

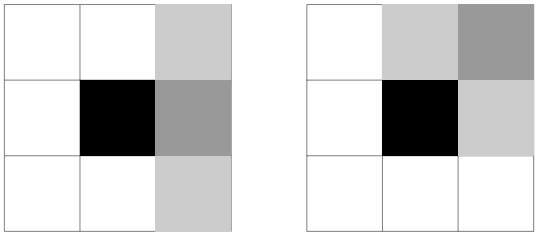

Figure 1: Examples of the errors in the process model. In both cases, the original robot position is the black square. The nominal direction of motion is the dark grey square. The errors in the robot mean that the robot will either end up in the dark grey square or the light grey square.

The engineers propose to use a model-free approach to learn a policy using a grid-based controller to plan the movement of the robot. This uses a simplified version of the robot state and the environment. The robot's state consists of its position expressed as the grid coordinates (x, y).

Two methods for model-free evaluation and control have been proposed: Monte Carlo methods, and Temporal Difference (TD) Learning.

1. a. What are the on-policy and off-policy Monte Carlo-based approaches for estimating $v_\pi(s)$? What are the main differences between them and the model-based approaches encountered in the first part of the module? What are the potential issues with using MC-style learning?

[3 marks]

Both the on-policy and off-policy MC algorithms are to be used to estimate $v_\pi(s)$ for a simple scenario and a single policy. In this scenario, the policy evaluator from Lecture 07 can be used to provide a ground truth solution.

b. Run the script `q1_b.py` and compare the effects of both first visit and every visit strategies on the MC solutions. By discussing the results of each, which algorithm appears to perform better? How does changing the number of episodes change the results?

   Hint: you should consider a way to *quantitatively* evaluate the difference between the ground truth and different predictor algorithms.

   [6 marks]

c. Off policy MC policy prediction algorithms use a target policy and a behavior policy. Explain what these policies are. Using the script `q1_c.py`, test several different values of ϵ_b to explore the behaviour. Are there any choices of ϵ_b where the algorithm will fail (crash or throw an exception)? Are there any choices of ϵ_b where the system converges extremely slowly? Explain your reasoning for both answers.

   [4 marks]

d. Temporal Difference Learning (TDL) is another sample-based approach for evaluating policies and learning controllers. Describe a TD learning algorithm to estimate $v_\pi(s)$. What are the advantages or disadvantages of TD compared to MC approaches. Are TD learning algorithms on or off policy? Is that good or bad?

   [4 marks]

e. Modify `TDPolicyPredictor` in `td/td_policy_predictor.py` to finish implementing the Tabular TD(0) algorithm for estimating $v_{\pi(s)}$. Explain your implementation.

   [6 marks]

f. Using the script `q1_f.py`, try several values of α in the TD algorithm. How do these values affect the rate of convergence? Do they influence the final values the solutions converge to? Why do you think you get this result?

[5 marks]

g. Being able to figure out the policy directly from observing a set of episodes is very useful for behaviour cloning and learning by example. One way to do this is to use the Monte Carlo / TD learned prediction of $v_\pi(s)$ to reverse engineer the policy π that the robot is running. What algorithm would be used to work out the policy? What do you need to know to be able to apply this?

[4 marks]

[Total 33 marks]

Some policies now need to be designed to guide the movement of the robot through the airport. In this case, the policy needs to take the robot from an arbitrary start location and bring it to one of a set of terminal states. TD algorithms will be used to determine the policies.

2. a. Q-Learning is described as an off-policy control algorithm. Briefly describe Q-learning algorithm for TD(0) and explain why it is off-policy.

[2 marks]

b. Modify the `QLearner` class to complete the implementation of the Q-learning algorithm. Explain your code.

Hint: You might find the small scenarios provided in `q2_f.py` helpful for debugging your code and illustrating that it functions correctly.

[6 marks]

The next few subquestions will use the `corridor_scenario`.

c. Using the script `q2_c.py`, find and show the policy and associated value function for the `corridor_scenario`. Discuss the policy and value functions. Do you think the converged policy appears to be sensible or not? Explain your reasoning.

[4 marks]

d. One way to change the performance of the algorithm is to vary the number of iterations required. By examining the policy and value function for several different choices, do you see evidence that changing the number of iterations affects the overall behaviour? Discuss and present the evidence for your conclusions.

[4 marks]

e. Instrument `q2_e.py` to store the time required for each episode. Present the timings. Are the timings constant or do they vary? If they vary, what do you think causes this variation? Provide evidence to support your answer.

[5 marks]

f. The Sarsa algorithm is an on-policy controller. Briefly describe the Sarsa algorithm. What are its potential advantages and disadvantages compared to Q-learning?

[2 marks]

g. Modify the `SARSA` class to complete the implementation of the SARSA algorithm. Explain your code.

Hint: You might find the small scenarios provided in `q2_f.py` helpful for debugging your code and illustrating that it functions correctly.

[6 marks]

h. Compare the performance of Q-learning and SARSA on the corridor scenario using `q2_h.py`. What do you notice about the state values and the extracted policies for each algorithm? What do you think might be the reason for this?

[4 marks]

The following is purely optional and *no* marks are awarded for it. It's provided just in case you might find it interesting.

The floor of the airport can be very smooth. As a result, the robot can slip with a probability $p_{slip}(s)$. When the robot slips, it stays in the same state and has a penalty -1. With probability $1 - p_{slip}(s)$ robot does not slip. In this case it uses the noisy model above.

i. Run the script `q2_i.py` on this example and compare how the algorithm behaves compared with the earlier examples where the floor was not slippery. You might want to look at things such as the computed policy, the value function, and the time required.

[Total 33 marks]

[Coursework part 1 total 66 marks]