COMP0037 2025 / 2026 Robotic Systems

Lab 03: Policy Evaluation

COMP0037 Teaching Team

February 13, 2026

Overview

In this lab, we will explore some of the ideas associated with FMDPs and policy evaluation. This is needed to understand how the policy and value iteration algorithms work and covers some material in Lectures in 05 and 06. It will also introduce you to the rest of the software which is used for CW1.

The scenario is that some designers have bought a robot from the McCheap Robot Company. The robot operates in an environment with goals and holes. Both are terminal sites — if the robot falls down them, it's game over. Goals give a big positive reward, holes a big negative one.

The robot is only capable of moving in four directions (`MOVE_LEFT`, `MOVE_RIGHT`, `MOVE_UP`, `MOVE_DOWN`) and waiting (`WAIT`). These first five actions are move actions. In addition, to these actions, we add two artificial ones. The `TERMINATE` action is only called at a terminal state (goal or hole) and `NONE` which is the only task allowed if the robot is in a state it shouldn't be able to get (e.g., inside a wall).

Installation Instructions

The code does not require you to install any additional packages beyond those required to support Lab 02.

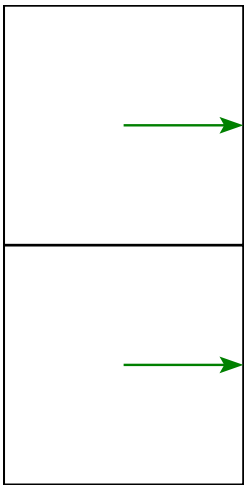

Figure 1: The basic policy.

Activities

1. The script `setup_basic_map_and_policy.py` is used to create a simple map and a simple policy. All the activities in this question ask you to use it.

   a. Run the script `setup_basic_map_and_policy.py` and confirm you get an output which looks like the one in Figure 1

   This basic scenario is rather boring and the policy is a bit useless.

   b. Modify `setup_basic_map_and_policy.py` to add a goal state at (1,2).

   Hint: Check the API for `EnvironmentMap`.

   Hint: Recall that the indexing into the map starts at (0, 0), not (1, 1).

   c. Modify the policy in `setup_basic_map_and_policy.py` to work in this case. The policy should do the following:

      1. Introduce a move up action in cell (1, 1).
      2. Introduce a terminate action in cell (1,2).

   Hint: Check the API for `Policy` and the action types specified in `ActionTypes` enum.

-103	-102	-101	-100	-101	98	99	100	99	98

Figure 2: The converged value function $v_\pi(s)$.

2. This question asks you to reverse engineer an environment and a policy. You will need to edit the script `evaluate_simple_policy.py`.

   a. The environment and policy is unknown. However, the output of the policy evaluator is shown in Figure 2. Work out the location of the hole and goal, together with the policy, which causes this value function to arise.

The McCheap Robot lives up to its name. It has an "ϵ-greedy" style of behaviour. Let a_t be the action requested at time t. When a_t is a move action — in other words $a_t \in \{$`MOVE_LEFT`, `MOVE_RIGHT`, `MOVE_UP`, `MOVE_DOWN`, `WAIT`$\}$ — then there is a probability $1 - \epsilon$ that it will execute the desired command. There is a probability ϵ that it will execute a move command (including the specified one) at random. (If a is one of the artifical commands — `TERMINATE` or wait `NONE` — it is guaranteed to be executed with probability 1.)

This model will be implemented as an FMDP.

3. You will need to modify the scripts `check_fmdp.py` and `environment.py`.

   a. Modify the code so that at each cell, you run different actions. Examine the distribution of the next state, reward and probability. What do you notice about these?

   b. Modify the environment to make it more interesting and check other actions. The types of things you could investigate include making the grid 2D, adding more goals, adding more holes, and adding obstacles.

   c. Modify the method `next_state_and_reward_distribution` to implement the "ϵ-greedy" model for the robot. Explore how the FMDP behaves.

      Hint: Recall that in call cases the probability must sum to 1.

The robot now operates in an environment where there is a cliff the robot can fall down. The cliff is modelled as a series of hole cells.

4. In this question you will modify the script `cliff_of_doom.py`.

    a. Complete a policy which will get you from the start to the goal from any point in the map, avoiding the cliff.

    b. Try different values of ϵ by setting different values in `environment` using the `set_epsilon` method. How do these influence the state value functions?