Assignment 2
AI and Robotics (AIR) - 236609

January 4, 2023

This is the second out of the three assignments of this course. Each assignment is divided into two distinct and complementary parts: theoretical and practical.

1 Theoretical Assignment

For the theoretical part you are required to submit one document, a pdf file generated using the latex template "Theoretical Assignment.tex" provided at `https://www.overleaf.com/read/mkphtcyjbjpc`. The maximal length of the submitted document is 3 pages.

In the git repository `https://github.com/sarah-keren/ai_dm`, you will find our ai_dm (AI decision making) library, which is aimed at demonstrating different AI approaches for sequential decision making, including planning, RL, learning, etc. It is still work-in-progress, but it already contains an implementation of Best-First-Search and a variety of its instantiations (such as A*, Breadth-First-Search etc) and a basic implementation of q-learning.

To see how the code works, have a look at the folder ai_dm/Tests, where you will find the notebook ai_dm_tutorial_with_rl.ipynb that demonstrates how to run our code on the openAI taxi domain. Even though you can solve the assignment without running the code, we suggest you do run the notebook to examine the performance of the different algorithms.

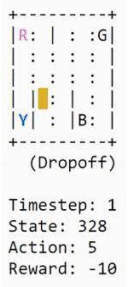

Figure 1

1.1 Modeling

Provide a formal definition of the Taxi domain using the three models from assignment 1: Classical planning, Markov-Decision Process (MDP) and Partially Observable Markov Decision Process (POMDP). For simplicity, assume there is no bound on the size of the grid (i.e., unless they are explicitly modeled as walls, there are no edges).

- For classical planning, use the original model as it is described for the taxi domain. `https://www.gymlibrary.dev/environments/toy_text/taxi/`
- For the definition of the MDP, consider the following stochastic effects. Pickup actions may fail to pick-up a passenger with some probability, in which case the passenger stays in place. In addition, a movement action may fail with some probability, in which case the agent moves in the opposite direction (it stays in place if it's near a wall).
- For the POMDP, support the case in which the agent does not know for sure where it is. Its initial belief is a uniform distribution over the actual cell, and the 4 adjacent cells. The agent has access to the map and can observe the walls around it, but when it observes a wall it does not know its direction for sure. Specifically, when it observes a wall from some direction X, the wall is in direction X with probability 0.8 and with 0.2 probability the wall is to the right of that direction (e.g., when the taxi observes a wall to its north, the wall is with 0.2 probability to it's east).

Make sure to provide a formal definition of all the components using the models we used in class.

1.2 Best First Search

In our current code, we run a* with the very basic goal_heuristic $h_G(n)$, which assigns a value of 1 to a node n that represents a terminal state, and 0 otherwise.

(a) Is $h_G(n)$ admissible for the taxi domain? If so, provide an intuitive explanation for why (no need for a formal proof). If not, provide a counter-example. As a reminder, a heuristic is admissible if it underestimates the cost to reach a goal, and over-estimates the total reward that is accumulated before a terminal state is reached.

(b) Suggest an admissible heuristic $h'_G(n)$ that dominates $h_G(n)$ for the taxi domain. As a consequence of the definition of an admissible heuristic, $h'(n)$ dominates $h(n)$ if for every node n

- $h(n) \leq h'(n)$ if the heuristic is used to estimate cost, and
- $h'(n) \leq h(n)$ if they represent accumulated reward

(c) Is your heuristic admissible for all deterministic and fully observable domains? If it is, provide an intuitive explanation why. Otherwise, provide a domain in which it is not.

(d) Suggest an admissible heuristic for the stochastic version of the taxi domain. An admissible heuristic for a stochastic domain under-estimates the excepted total cost/ over estimates the expected accumulated reward.

1.3 Q Learning

As we saw in class, the basic intuition behind most RL approaches: is to explore more when knowledge is weak and exploit more as you gain knowledge. Formally, this is the GLIE (Greedy in the Limit with Infinite Exploration) principle. An algorithm is GLIE if it satisfies the following two properties:

1. If a state is visited infinitely often, then each action in that state is chosen infinitely often (with probability 1).
2. In the limit (as $t \to \infty$), the learning policy is greedy with respect to its learned values (with probability 1).

The first condition requires exploration to proceed indefinitely, while the second requires that its magnitude decays over time.

In the standard version of Q-learning we saw in class (and which is the base of our current implementation of Q-learning `https://github.com/sarah-keren/ai_dm/blob/master/ai_dm/RL/q_learning.py`), we use the GLIE epsilon-greedy exploration principle.

1. Explain if and how the GLIE principles are supported in Q-learning.
2. Suggest an alternative implementation of Q-learning that is GLIE and explain how it satisfies the two conditions. Note that this can be an existing approach from the literature beyond the one we implemented in our code.

For both sections above, an informal explanation is sufficient (no need for formal proofs).

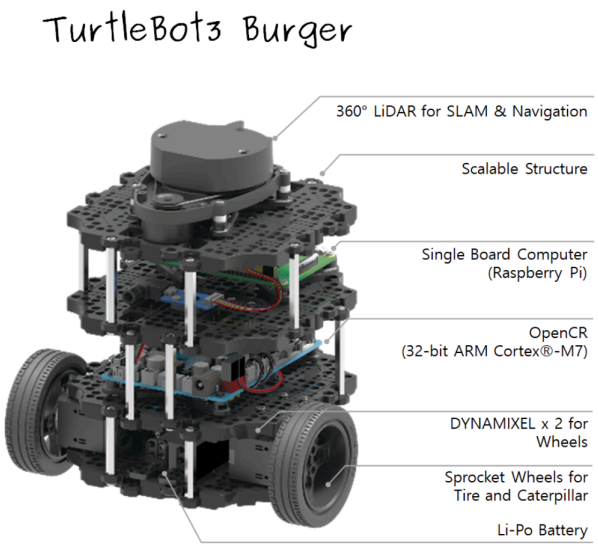

Figure 2

2 Practical Assignment

2.1 Logistics

For the practical part, you are required to submit 2 documents: A single python module containing your code named assignment2.py, and a pdf file generated using the latex template "Lab Report.tex" provided at `https://www.overleaf.com/read/mkphtcyjbjpc`.

You will be working with the turtlebots 3. To complete the practical part of the assignment you must run your solution both in simulation and on the real robots during reception hours.

2.2 Technical details

- We will be using Ubuntu18 with ROS melodic.
- The repo for the exercise can be downloaded from `https://github.com/sarah-keren/MRS_236609`
- You need to use the Turtlebot3.
- Our code should be installed under the src folder of the ROS project you created.
- You can use all the packages we have seen so far, including move_base.
- You will need to complete assignment2.py.
- You will be allowed to localize your robot, using the relevant ROS command.

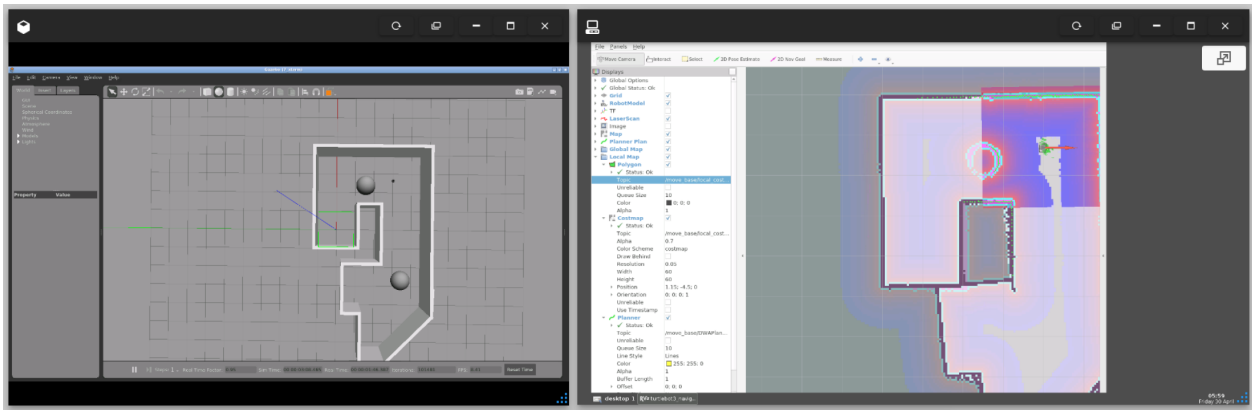

Figure 3

2.3 Task

You have an inspection task where a single robot with Lidar is placed in a room with its walls fully mapped (you will only receive the map at execution time). Apart from the walls, there will be spheres and blocks of unknown size in the room. The approximate location of the objects and the number of objects will be provided as input together with the map as well as the amount of time you will have for execution (between 2 to 10 minutes). Your task is to 'examine' all of the objects from all sides with the Lidar so that they are added to the map. You will use the allocated time to add as many objects to the map as you can.

2.4 Evaluation

For the development of your code and for the lab report, we will provide 5 simulated settings with maps and objects. We will evaluate your solutions based on the report, your code, and the physical experiments you will run in the lab.

2.5 Pay attention

- **Do not !** include ANY package that is not already installed by default. If you want a package that is not installed - you need our approval.
- You need to make sure your code is python2 compatible.
- The lab report must not exceed 5 pages.
- The maximal speed of the turtlebot is 0.22 m/s and 2.84 rad/sec.
- This may not be the final description of the assignment. As you start working on it, you will have questions, and we may need to slightly update the description accordingly.

Good luck and don't forget to enjoy the process!