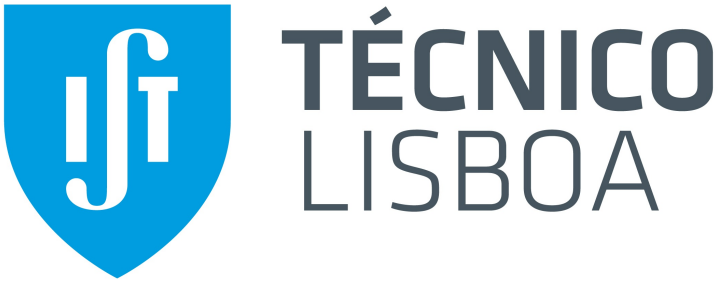


Introdução à Robótica

Mini Project 3

By:

Tiago Prata
Tiago Cabecinha

Professor:

Pedro Urbano Lima

Contents

1 Introduction **2**

2 Background **2**

2.1 Markov Decision Processes (MDP) 2

2.2 Policy 2

2.3 Value Iteration 3

2.4 Q Function 3

2.5 Reinforcement Learning 3

3 Implementation **3**

3.1 Defining the states 3

3.2 MDP Model 5

4 Results **7**

5 Conclusions **9**

1 Introduction

In our previous works, we managed to localize our robot in a certain map and create a path given that same map, using cell decomposition and then the robot was able to follow that same path using the Dynamic Window Approach algorithm. Being able to create its own path having a goal and being able to follow it gave some autonomy to the robot. Despite being able to even dodge unmapped obstacles, its autonomy is extremely low. It is true the robot is able to accomplish path planning and path following but it is not able to make its own decisions. In this project we had the task to give more autonomy to the robot making it able to take decisions knowing there are prohibited zones. In order to perform this project we were introduced to new concepts of decision making using model-based learning. It is also given an introduction to Reinforcement Learning but unfortunately it was not possible to explore it in this work.

2 Background

2.1 Markov Decision Processes (MDP)

A Markov Decision Process is a mathematical framework for modeling decision making. These are used for studying optimization problems solved via dynamic programming and reinforcement learning. A MDP has various components, states, actions, probability transition matrices, cost functions and goal states. The solution of these MDPs are policies which indicate which action is better in each state in order to reach the goal state. The states represent the information that the decision depends on representing relevant information. The actions represent the movements the robot can perform. The probability matrices describe the probability of a certain action be taken from one state to another. We will have the same number of matrices as we have of actions since we need a matrix for each action and each of this matrices has a states by states size. The cost function describes the cost of each action when in each state. Having this information it is easy to understand it is a states by actions matrix. The MDPs follow the Markov Property which means the system only needs to have information about the previous event to make a decision.

2.2 Policy

A policy is a solution of a MDP. It describes which action the agent should take in each state according to the data the MDP has. The optimal policy describes the best action to take for each state in the model. A policy is represented by a vector with size equal to the number states and in each position it will have the best action to make in that state in order to minimize the cost. In order to compare the actions the agent can take, there is a need to have a way to compare them. For this we use the cost-to-go function shown next.

$$J^{\pi} = (I - \gamma P_{\pi})^{-1} c_{\pi} \tag{1}$$

2.3 Value Iteration

It is a Dynamic Programming method for computing the optimal policy. It recursively refines an estimate of Q. There is not a proper end to this recursive so it uses normally a threshold that is compared with the difference between the Q from the last iteration with the Q from the previous iteration. If the value is lower than the threshold than the recursive ends.

2.4 Q Function

Represent the average cost-to-go of a fixed policy given an initial state and action. The Q function maps each state-action pair and is defined by the following equation.

$$Q^{\pi}(x,a) = c(x,a) + \gamma \sum_{y \in \chi} P_a(y|x) J^{\pi}(y) \tag{2}$$

After having the optimal Q function computed, we iterate through each state and check the action with less cost. Knowing these it is easy to compute the optimal policy.

2.5 Reinforcement Learning

In contrast with what was talked before, Reinforcement Learning is a model-free approach that attempts to figure out which action the agent should perform in each state in order to have the lowest cost possible. The RL algorithms do this without having any knowledge of the environment the agent lives in. Not having this knowledge, the agents needs to learn what happens while it interacts with the environment. When working with RL there are two main components, an agent and the environment he lives in. The agent is an entity that performs actions in the environment it lives on and receives a cost or reward. The environment is where the agent lives and interacts. To interact with the environment, the agent needs to perform actions. An action is a move the agent can perform. Inside the environment there are different stages making it easier for the user to understand the agent situation. When the agent performs an action in the environment, it receives a rewards resulting from that action.

There are three main approaches when implementing a RL algorithm, Value based that has the purpose to maximize a Value Function and the goal is to find the optimal value, Policy based that generates a policy that gives the best action in each state in order to minimize that cost to the fullest having the goal of finding the optimal policy and Model Based, that creates a virtual environment and the agent runs several iterations across that environment learning with each iteration.

3 Implementation

In this section, it is explained the main steps for the resolution of all proposed tasks.

3.1 Defining the states

The first thing to do, was to discretize the map used in the previous works. To accomplish that, we defined the cell size in *Rviz* for the size of the robot, having now squares of *30cm X 30cm*. This discretized map is represented in the figure 1.

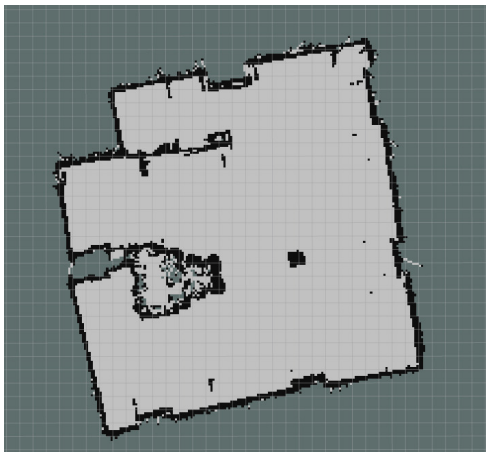
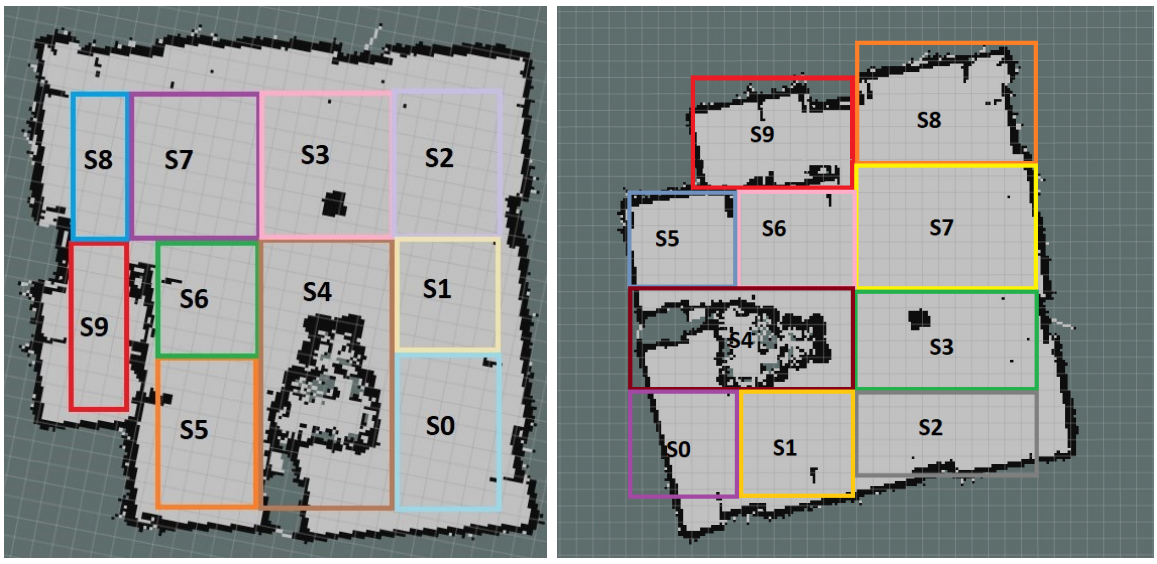


Figure 1: Discretized map

Our interpretation of this task, got us to a different approach. We tried to define our own states where each state is a set of cells from the map. This way, it is possible to define our set of states.

The ideal representation of states is shown in figure 2a, where each state is composed by cells where each cell is completely used or not in that state. This ideal situation would be accomplish if we were able to change the orientation of the map in relation to the cells but we were not able to do that. The map is strictly connected to the squared cells of the *Rviz*.

Although we had this setback, we proceed to the states definition in the map. The true representation of the states is shown in figure 2b. It is now more enlightening the problem we were having. In this situation, a state is represented by a certain number of cells, where the cells need to belong completely to a state.



(a) Ideal Situation (b) Real Situation

Figure 2: Defining States.

3.2 MDP Model

Our main purpose of this project was to make the robot reach a certain destination without passing through a certain region of the room. Our personal choice was to make the center of the class, section S4, our forbidden region and we defined two goals, the states S0 and S9. In order to discover what were the best actions to perform in each state we created two different MDPs with the same transition probabilities matrices however, they had different reward matrices since their goals were different. Our environment was divided into 10 different spaces as shown in the previous images in the previous section. This states were chosen this way so that we could easily create our MDPs without having undesirable problems related with the transition matrices. The actions we defined were UP, DOWN, LEFT, RIGHT and STAY and we based this action according to the image 2 a) in the previous section. For example, if the robot is in the state S0 and performs the actions UP, it will most likely go to state S1. We had five different transition matrices, one for each action where the shape of each one was 10 x 10 being the lines the starting state and the columns the state after performing that action. To help calculate our policies, we also defined two reward matrices with the shape 10 x 5 being the lines the states and the columns the actions the robot can perform. The pair (state, action) of any of this matrices would give us the reward of performing that action in that state.

$$P_{UP} = \begin{pmatrix} 0.20 & 0.80 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.20 & 0.80 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.80 & 0.20 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.20 & 0.80 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.20 & 0.80 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.80 & 0.20 \end{pmatrix}$$

$$P_{DOWN} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.80 & 0.20 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.80 & 0.20 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.20 & 0.80 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.80 & 0.20 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.60 & 0.40 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.20 & 0.80 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$P_{LEFT} = \begin{pmatrix} 0.20 & 0 & 0 & 0 & 0.80 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.20 & 0 & 0 & 0.80 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.20 & 0.70 & 0.10 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.20 & 0 & 0 & 0.10 & 0.70 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.20 & 0.40 & 0.40 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.20 & 0.80 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$P_{RIGHT} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.80 & 0.20 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.35 & 0.35 & 0.10 & 0 & 0.20 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.80 & 0.20 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.10 & 0.70 & 0 & 0.20 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.80 & 0 & 0 & 0 & 0.20 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.80 & 0.20 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$P_{STAY} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_0 = \begin{pmatrix} 0 & 0 & -10 & 0 & 1 \\ 0 & 1 & -10 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & -10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -10 \\ 0 & 0 & 0 & -10 & 0 \\ 0 & 0 & 0 & -10 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad R_9 = \begin{pmatrix} 0 & 0 & -10 & 0 & 0 \\ 0 & 0 & -10 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & -10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -10 & 0 \\ 0 & 0 & 0 & -10 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

4 Results

It was developed a a script in Python that, given the matrixes with the actions and the reward matrixes for the goal states it solves the two MDPs calculating the optimal policies for each one. The tool that makes this process is the *mdptoolbox.mdp.ValueIteration* [9], that receives as arguments the transition matrix, the reward matrix, the discount factor and the maximum iteration value.

The discount factor - γ - represents the preference of short-term solutions over long-term solutions although it takes more time for it to converge. This value is bounded between 0 and 1 and our choice was to use $\gamma = 0.9$. Asymptotically, the closer γ is to 1, the closer the policy will be to one that optimizes the gains over infinite time. The two runs would yield two policies for the robot to make its decisions to get to each goal.

The calculated optimal policies with $\gamma = 0.9$, can be seen bellow.

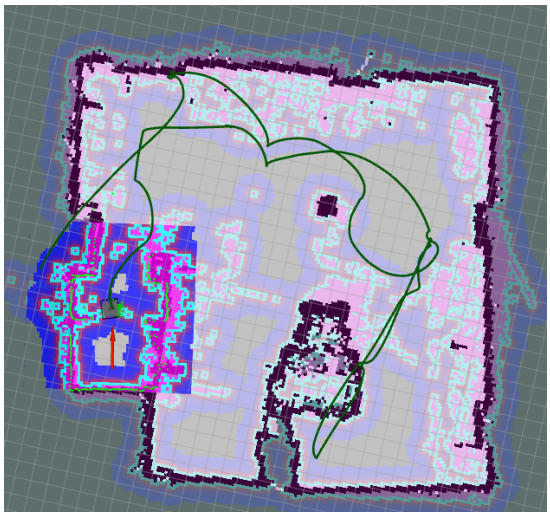
$$policy1 = [4, 1, 1, 3, 3, 0, 0, 3, 3, 0]$$

$$policy2 = [0, 0, 2, 2, 0, 0, 0, 2, 1, 4]$$

After getting the optimal policies, it is sent each goal state to the robot, basing on the previous work with the Navigation Stack. The value that is sent for the robot corresponds to the center of the goal state.

For each policy, it is generated a path that a robot should follow. This path is composed by all the states the robot should pass.

It was performed to experiments: One starting in the state S0 with goal states S9, S0 abd the other starting in the state S5 with the same goal states. This experiments can be visualized in the videos in [3]

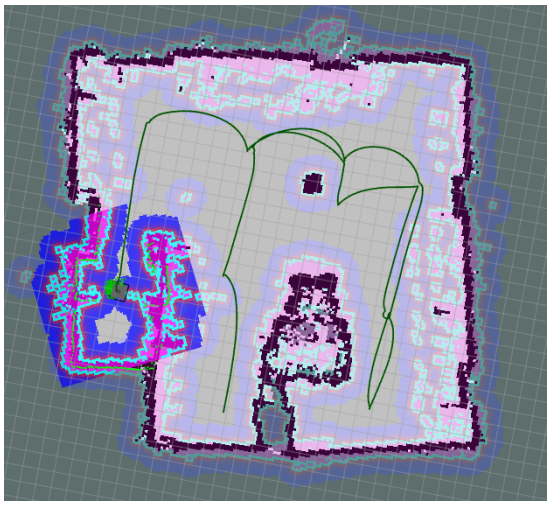


(a) First Trajectory

```
p-X571GT-X571GT:catkin_ws $ rosrun turtlebot3_na
vigation mdp_try.py
State 9
[8, 7, 3, 2, 2, 1, 0]
[INFO] [/turtle_tf_listener]: Goal execution don
e!
State 0
State Calculated
[1, 2, 3, 7, 7, 8, 8, 9]
[INFO] [/turtle_tf_listener]: Goal execution don
e!
tiagocabecinha@tiagocabecinha-VivoBook-ASUSLapto
```

(b) Results from First Trajectory.

Figure 3: First Experiment.



(a) Second Trajectory

```
tiagocabecinha@tiagocabecinha-VivoBook-ASUSLapto
p-X571GT-X571GT:catkin_ws $ rosrun turtlebot3_na
vigation mdp_try.py
State 5
[6, 7, 7, 3, 2, 1, 0]
[INFO] [/turtle_tf_listener]: Goal execution don
e!
State 0
State Calculated
[0, 1, 2, 2, 2, 2, 3, 7, 8, 9]
[INFO] [/turtle_tf_listener]: Goal execution don
e!
tiagocabecinha@tiagocabecinha-VivoBook-ASUSLaptoah
```

(b) Results from Second Trajectory.

Figure 4: Second Experiment.

When saving the trajectory, the path of the robot oscillates as the time goes by because of the accumalted errors and deviations related to the robot pose. It is recomended to visualize the videos to understand better each experiment.

We can see in the images 3 and 4 the result of both experiments. At right, in both images, it is shown the calculated paths for each goal state. At the left of both images, it can be seen the path performed by the robot.

5 Conclusions

At the end of this work, we can say that the main tasks were accomplished with success, although we have not explored as we wanted the Reinforcement Learning approach.

About the developed work, we did not follow the most common approach regarding the discretization of the map into states. We have created our own states, formed by a set of cells from Rviz. Following this approach, took us to a smaller state space. Even though, we were able to accomplish the requested tasks, getting the optimal policy with few iterations and turn it into a path for the robot to follow.

Using the algorithms from the previous works, the robot was able to follow that path while localizing himself in the map, avoiding the forbidden area and respecting the solved policy from the MDP model.

In the dropbox [3], it can be found all the videos and files regarding to this project.

References

[1] https://github.com/ros-planning/navigation

[2] https://github.com/udacity/odom_to_trajectory

[3] https://www.dropbox.com/sh/zl597u3wkonm0rc/AADcYACq5w5ld4_GasQQUIaZa?dl=0

[4] http://kaiyuzheng.me/documents/navguide.pdf

[5] https://emanual.robotis.com/docs/en/platform/turtlebot3/navigation/navigation

[6] https://en.wikipedia.org/wiki/Markov_decision_process

[7] https://artint.info/html/ArtInt_227.html

[8] https://en.wikipedia.org/wiki/Reinforcement_learning

[9] “Markov decision process (mdp) toolbox.”