1 2 9 0

UNIVERSIDADE D
COIMBRA

Diogo António Ferreira Temporão Alves

Reinforcement Learning in the Navigation of Mobile Robots

Dissertation submitted to the Department of Electrical and Computer Engineering
of the Faculty of Science and Technology of the University of Coimbra
in partial fulfillment of the requirements for the Degree of Master of Science.

September 2019

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA

Diogo António Ferreira Temporão Alves

Reinforcement Learning in the Navigation of Mobile Robots

Supervisor:

Prof. Dr. Urbano José Carreira Nunes

Co-Supervisor:

Master Luís Carlos Artur da Silva Garrote

Jury:

Prof. Dr. Urbano José Carreira Nunes

Prof. Dr. Rui Alexandre de Matos Araújo

Prof. Dr. Rui Paulo Pinto da Rocha

A dissertation submitted in partial satisfaction of the requirements for the degree of Master of Science in Electrical and Computer Engineering

Coimbra, 2019

Acknowledgements

This dissertation could not have been completed without the supervision and support of many people. People to which I would like to express my admiration and thanks. I am very grateful to my advisors, Professor Doutor Urbano Nunes, and Master Luís Garrote, for allowing me to work on a theme that I enjoy, for offering me support in many and varied ways, for the orientations and suggestions provided during the realization of this work.

I would also like to thank my laboratory colleagues for their good disposition, which resulted in a relaxed, fun and healthy work environment. For Ricardo Cruz, for his availability in helping with the technical aspects of the InterBot platform.

A special thanks to Luis Garrote, for his patience and total availability to give great advice, for allowing pertinent discussions, always keeping his sense of humor, being an extra motivation for the development of the work.

I thank Institute of Systems and Robotics - University of Coimbra (ISR-UC) for providing me with the necessary conditions and resources for the realization of this final stage. This work has been supported by Fundação para a Ciência e Tecnologia (FCT) through project "UID/EEA/00048/2019", and "MATIS (CENTRO-01-0145-FEDER-000014)", project with FEDER funding, and programs PT2020 and CENTRO2020.

I am very grateful to all my friends, with a special mention to those who were part of my academic journey, always keeping the memories and experiences lived.

Finally, I am deeply grateful to my family, especially to my parents, brother, and girlfriend who provided me with all the support I needed to achieve my master's degree.

To all of you mentioned here and to others who are not, thank you very much!

Abstract

Over time, the idea that robots only carry out roles related to the industrial sector has been disappearing. Today, in society, there is a strong integration of robots in order to help/improve the execution of certain tasks. As a result, robots can be seen as essential tools in our daily lives, in many areas such as medicine, education or service robotics.

The main objective of this Master's dissertation is to develop and implement a new local navigation method for mobile robots based on Reinforcement Learning. This method enables virtual or real mobile platforms such as InterBot-Social Robot, developed at the Institute of Systems and Robotics (ISR-UC), to follow a path to navigate from location A to B. The method consists of two stages: training stage and online stage. The training stage consists in the robot learning to follow a previously defined path. This stage is performed in a simulation environment, providing total freedom in the development and improvement of the method. Through the training, a model is obtained and is used in the online stage enabling a mobile platform, in a simulation or real environment, to move along a path avoiding obstacles.

A set of tests and experiments were performed in different scenarios: tests such as limiting the number of available actions, changing the type of path representation (defined by line segments or cubic splines) and introducing obstacles near the path. The method developed presents promising results for paths with and without obstacles. When there is a limitation in the number of actions, the robot's behavior is unstable, although it can accomplish the desired objective.

Keywords: Navigation, Planning, Reinforcement Learning, Rewards, Actions, States

Resumo

Com o passar do tempo, a ideia de que os robôs desempenham unicamente papeis ligados ao sector industrial tem vindo a desaparecer. Atualmente, na sociedade, existe uma forte integração de robôs com o objetivo de auxiliar/melhorar a execução de determinadas tarefas. Desta forma, os robôs podem ser vistos como ferramentas essenciais no nosso quotidiano, em diversas áreas como medicina, educação, ou robótica de serviços.

Esta dissertação de Mestrado tem como objetivo principal desenvolver e implementar um novo método de navegação local para robôs móveis tendo por base aprendizagem por reforço (*Reinforcement Learning*). Este método permite que plataformas móveis virtuais ou reais como InterBot-Social Robot, desenvolvida no Instituto de Sistemas e Robótica (ISR-UC), siga um caminho de forma a navegar de um local A para B. O método consiste em dois estágios: estágio de treino e estágio online. O estágio de treino consiste em o robô aprender a seguir um caminho previamente definido. Este estágio é realizado num ambiente de simulação, permitindo uma total liberdade no desenvolvimento e aperfeiçoamento do método. Através do treino é obtido um modelo que é utilizado no estágio online permitindo que uma plataforma móvel, num ambiente de simulação ou real, se mova ao longo de um caminho evitando obstáculos.

Foi realizado um conjunto de testes experimentais abrangendo diferentes cenários: limitação no número de ações disponivéis, alteração do tipo de representação do caminho (definido por segmentos de reta ou splines cúbicos) e introdução obstáculos em pontos na vizinhança do caminho. O método desenvolvido apresenta resultados promissores para caminhos com e sem obstáculos. Quando há limitação no número das ações o comportamento do robô é instável embora consiga cumprir o ojetivo pretendido.

Palavras chave: Navegação, Planeamento, Reinforcement Learning, Recompensas, Ações, Estados

"Failure is simply the opportunity to begin again, this time more intelligently."

Henry Ford

Contents

List of Acronyms

BSRA Biased Sample-based Robot-Action

DWA Dynamic Window Approach

InterBot Interactive Mobile Robot

ISR Instituto de Sistemas e Robótica

ISRsea ISR Shared Experimental Area

RL Reinforcement Learning

ROS Robot Operating System

RRT Rapidly-exploring Random Tree

VREP Virtual Robot Experimentation Platform

List of Figures

List of Tables

Chapter 1

Introduction

This chapter presents the context and motivation of the developed work, as well as the main goals and key contributions.

1.1 Context and motivation

Robotics is the sector of technology who is responsible for the development and study of robots. Initially, the study of robotics was based on the need to automate the industrial sector allowing the machines to perform certain tasks automatically. As time passed, the robots with and without mobile capacity emerged, allowing activities to be carried out in place of the human being. The progress of technology over the years has contributed to great discoveries, not only on a technological but also on a scientific level. NASA announced in 2012 the mission to be accomplished in the year 2020, Mars 2020 [1], with the main objective of finding signs of habitable conditions on the planet Mars. For this mission, they will use a vehicle denominated rover, which has the required characteristics to navigate and get relevant information about the planet in question.

A mobile robot is characterized by its ability to move in a given environment. The robot has to have 3 basic modules: mapping (environment perception), localization and path planning. Navigation is not only about the robot knowing the representation of the environment, but also interpreting that representation, to be able to locate itself, allowing the planning of a path. One of the main objectives in the research of mobile robots is to increase their autonomy, social and human capabilities.

This dissertation work plan consists of developing and implementing a new local navigation method for the "InterBot-Social Robot" [2]. This robot was developed in the ISR-UC and has been the target of development and application as a working tool in several works,

such as "InterBot Mobile Robot: Human-Robot Interaction Modules" [3] and "InterBot mobile robot: Navigation modules" [4]. The work "A new hybrid motion planner applied in a brain-actuated robotic wheelchair" [5], used another platform, Robchair, also developed in the ISR-UC, being its navigation based on an algorithm called D-Dynamic Window Approach (D-DWA).

1.2 Main objectives

The main purpose of this work was to develop and test a local navigation method for a mobile robot based on Reinforcement Learning (RL) that allows following a path avoiding obstacles.

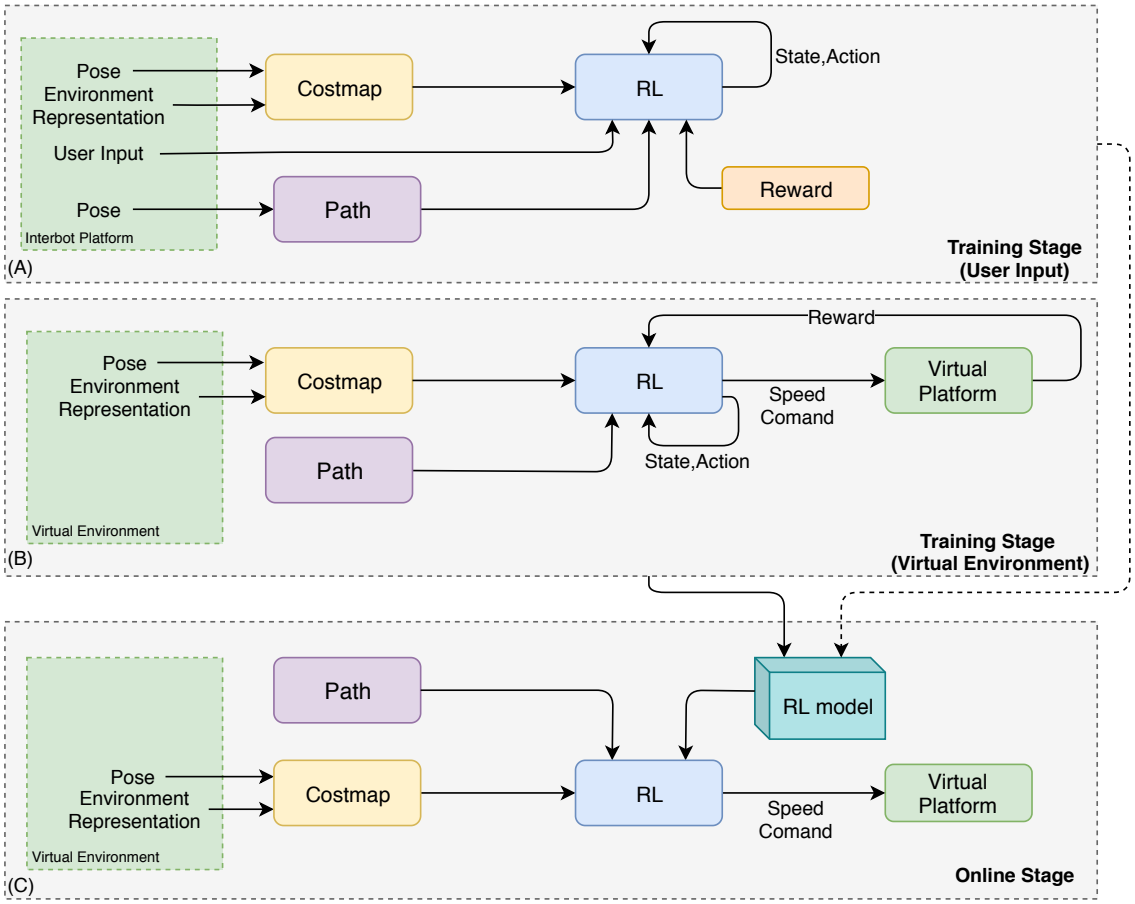


Figure 1.1: Block diagram of the proposed RL-based local navigation system.

The proposed framework (Fig. 1.1) consists of two stages as follows:

- **Training stage:** This stage consists of generating a model based on: (B) the training of a robot to follow a path in a simulation environment or (A) user's driving behavior. The user uses a gamepad to drive the InterBot platform.

- **Online stage:** The robot uses the model generated in the previous stage to follow a path avoiding obstacles (C).

When developing the method based in RL, a number of components/parameters need to be defined. In the process of developing the method, there were questions such as:

- How can states be defined?
- What actions can the agent perform?
- How can we reward the agent after performing an action?

1.3 Implementations and key contributions

The following main implementations and contributions are described in this dissertation:
Developed work (Chapter 4)

- Description of the proposed method as well as the reasoning for the proposed solutions to the questions described above since some relevant problems have appeared. Problems such as:
  - Small variations of the robot's pose generate multiple different states.
  - Reward propagation (delayed reward).
  - The robot gets stuck in sub-optimal solutions.

  The solutions found consisted in implementing:
  - a similarity state module.
  - a reward penalty (reward propagation) module.
  - an algorithm denoted as Biased Sample-Based Robot-Action algorithm (BSRA) Policy Search.

Software Implementation (Chapter 5)

- Description of the developed ROS package "RL_nav" that contains the implemented method and enables the communication/interaction with a real or a virtual platform.

Chapter 2

State of the art

2.1 Robot path planning

Generically path planning consists of finding a route to a goal without colliding with the environment. The path should be determined in the environment in which the robot moves. The environments can be static (position of objects does not change with time) or dynamic (position of objects/persons, changes with time). In this way the robot must be able to find the path while navigates. Two main planning perspectives are considered to exist: global and local. These perspectives use two types of map representation since a map can follow metric or topological approximations [6]. A topological map is a simplified map (graph) composed by nodes and arcs. Nodes are used to define specific locations and arcs are used to define paths between these locations. A metric map, considering the work developed, is a two-dimensional grid map, represented based on the discretization of the environment. Each cell on the map contains information on the presence or not of an obstacle. If an obstacle exists, the cell has a value of 1, otherwise 0. Figure 2.1 shows a diagram of the path planning perspectives and the types of maps used. Table 2.1 lists a few representative examples of path planning algorithms categorized by type (global, intermediate and local) and map representation (topological and metric).

2.1.1 Global Planning

Global planning can be divided into two approaches: when applied to mission planning and when the behavior of the robot in the environment is planned (path that the robot must follow). Global planning consists in a long-term plan that use a costmap considered static and global.

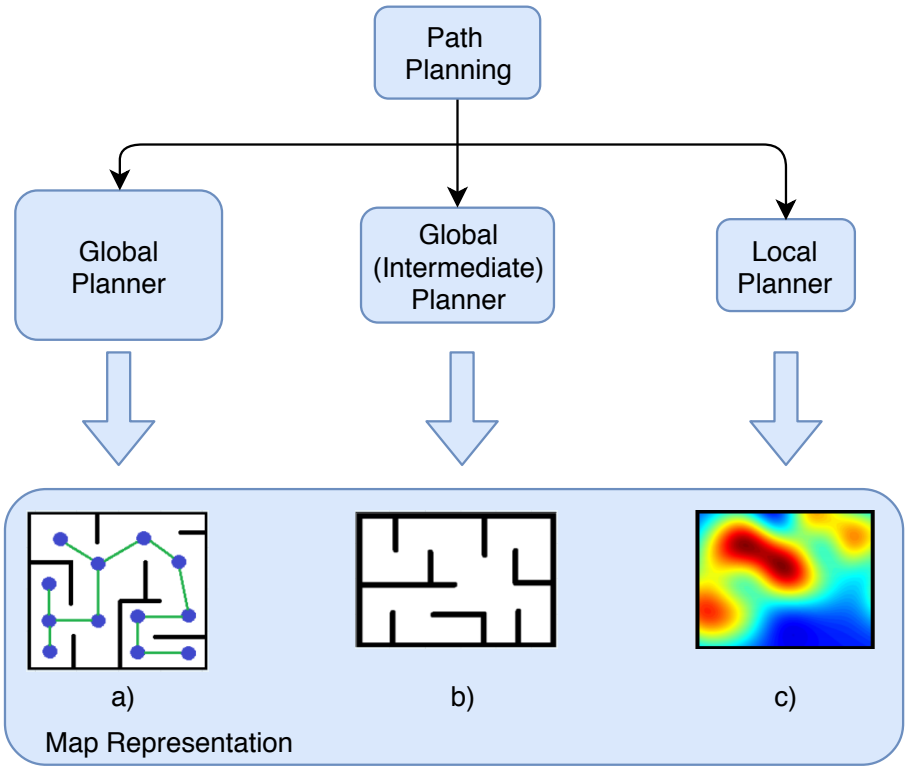


Figure 2.1: Three planning perspectives used by mobile robots: Global, Global Intermediate, Local and the respective maps they use: (a) Topological - green lines represent paths and blue points locations. (b) and (c) Metric - black/red cells are occupied and white/blue cells are empty.

Table 2.1: Examples of path planning algorithms with its main aspects (perspective and map representation).

Algorithm	**Path Planning**			**Map representation**	
	Global	**Global (Intermediate)**	**Local**	**Topological**	**Metric**
Dijkstra's	*			*	
Floyd-Warshall	*			*	
A*	*	*		*	*
RRT		*			*
Theta*		*			*
DWA			*		*
RL			*		*
Deep-RL			*		*

There are several algorithms that are based on the use of graphs, to choose the shortest path, such as: Dijkstra's algorithm [7] that determines the shortest path from one node to all the other nodes that compose the graph. Floyd-Warshall algorithm [8], compute the length of all shortest paths between any two vertices in a graph and A* search algorithm [9] differs from the Dijkstra algorithm since finding the lowest cost path between two nodes of a graph (combination of heuristic approaches). A variant of A* algorithm was proposed to solve the optimal path planning for mobile robots [10].

Algorithms such as Rapidly-exploring random tree (RRT) algorithm or Theta* algorithm are used in the second approach based on a metric map. RRT algorithm [11] is an algorithm based on creating a tree: a viable path is determined in each iteration, between a point (node) in the tree and a randomly generated point (node). A link is established between the tree node and the nearest generated node, which gives origin to the tree that fills the space. In [12] a variant of RRT was used to explore narrow passages or difficult areas more effectively. Theta* (Any-Angle) algorithm [13] is a variant of the A* made by Alex Nash et al., in 2007. This algorithm is similar to A* algorithm, where the main difference is the link between two vertices. In algorithm A* a node only recognizes the neighboring nodes, and in Theta* a node recognizes all the nodes. So a node can be connected to any node present in graph.

2.1.2 Local Planning

In local path planning, the planner uses sensory information captured from the environment to set a path. A metric map (e.g local costmap) is used to generate a short-term plan. Examples of local planning algorithms are Dynamic Window Approach (DWA), Reinforcement Learning (RL), and Deep Reinforcement Learning (Deep-RL). The DWA algorithm was proposed by Fox et al., 1997. This algorithm results directly from the dynamics of the robot, i.e., the controls responsible for the speeds (linear and angular) of the robot are searched. This is done using the dynamic window, which contains the speeds considered acceptable (speeds that guarantee the robot to stop safely before reaching the obstacle). This method was used in [14] to collision avoidance. A Reinforcement Learning algorithm consists in an agent that learns by interacting with a environment, receiving a value of reward in function of the actions that performed. The Deep Reinforcement Learning algorithm [15] can be seen as an extension of Reinforcement Learning. It uses a deep learning architecture, a neural network, combined with RL algorithms. In this type of algorithm the agent is the

neural network. Learning consists of adjusting weights using coefficients that approximate a nonlinear function between inputs and outputs.

2.2 Reinforcement Learning in robotic navigation

In autonomous navigation, the concept of learning is directly related to autonomy. Learning comes from adapting to unknown environments. In cases where the environment has dynamic characteristics, continuous learning is required to comply with the environment changing. A mobile robot whose navigation is based on RL has a higher learning capacity than another navigation method due to its self-adaptive capability.

In RL there are some definitions such as Agent (takes actions and interacts with the environment), State (current state of the agent), Reward (the value received after executing a certain action at a given state) and Environment (space where the agent moves).

The study and implementation of RL algorithms have been going on for some time. Table 2.2 presents an overview of the three algorithms (Q-Learning, R-Learning and H-Learning) based on the analysis and study of the information presented in [16, 17, 18].

In general this area of machine learning has been used in computer games [19], assistance robotics [20, 21], and in particular, Q-Learning has been used in the context of exploring labyrinths [22], move a robot to a goal avoiding obstacles [23, 24] or adapting a robot in a dynamic environment [25].

One of the adversities of implementing an RL algorithm is to define its components such as states, rewards, and actions. Table 2.3 presents a list of documents where RL is applied in the navigation, describing the main purpose and how the components of a RL-model were defined. In works such as [23] and [22], the states are defined as a function of position/distance between the robot and/or obstacles/objective. In [24] and [25], they are defined, respectively, based on force fields and a neural network that processes the information around the agent and returns the states. As regards actions, in most works, actions are based on moving forward, backward and turning right or left. As for the reward, in [25] and [23] the agent receives the reward depending on reaching the goal or avoiding collisions. In [22] and [24], the reward is defined, respectively, for each action in a specific state and as a function of the robot doing a certain movement.

Currently, there is a strong study and application of Deep Reinforcement learning for navigation in complex environments. The complexity increases with the number of objects/agents that the environment has. It happens that RL algorithms may not have the

computational capacity to generate an optimal navigation policy because it is not possible to treat the necessary states to obtain the maximum reward. Recent work [26] presents deep reinforcement learning for navigation in pedestrian-rich environments, the agent learns to navigate efficiently and safely. In [27], the agent learns how to navigate based on human behaviour following social rules such as passing on the right.

Table 2.2: Some aspects of the algorithms of Reinforcement Learning.

Q-Learning	**R-Learning**	**H-Learning**
- Model-independent - Optimizes reward with discounts - The actions do not allow the agent to predict the future - Requires low computational resources - Good performance in real time	- Model-independent - Optimizes the average undiscounted reward - It makes the agent "lazy"	- Model-dependent - Optimizes the average undiscounted reward

Table 2.3: Examples of the definition of states, actions and rewards in robots that use Reinforcement Learning for navigation.

Authors	**Description**	**States**	**Actions**	**Rewards**
G. Yen and T. Hickey [25]	Enables RL to be used in the direct control of a robot navigating in a dynamic environment.	3x3 square area surrounding the agent's. Neural network that returns/sets the states.	Not specified.	A small penalty for each step taken, a large penalty for collisions and a large reward for reaching the goal.
Nihal Altuntas, Erkan Imal, Nahit Emanet and Ceyda Nur Ozturk [23]	Guiding the mobile robot to a goal by avoiding obstacles.	Defined by the position of the agent according to the target and the nearest obstacle on the way.	3 actions: move forward with linear speed v or turn right or left with angular speed, ω.	Different rewards such as: for achieving the goal and for obstacle collision.
Gedson Faria and Roseli A. Francelin Romero [24]	Perform tasks avoiding obstacles.	Defined by force field concepts.	4 actions such as: move forward, turn left, turn right, and move backward.	In function of events such collision or move forward or turn right or left or backward.
B. Zuo, J. Chen, L. Wang, and Y. Wang [22]	Navigate in an unknown maze and move out.	Distances between the robot and the closest obstacles in three specific directions.	3 actions such as: move forward and left or move forward and right.	The value is defined for each action in a specific state.

Chapter 3

Background material

3.1 Reinforcement Learning

Reinforcement learning [28] can be defined as a learning method based on rewards when actions are performed in an environment. In this type of learning, who learns and make decisions is defined by agent, the one that interacts with the environment. The agent should learn what to do in order to maximize the value of these rewards, that is, a set of actions is performed and must learn which of these actions produce the biggest reward in order to, once placed in an environment, it should learn to be successful in it. Anytime an action is executed, the agent moves from one state to another, receiving its reward. In this way, a state can be characterized by the reward that the agent receives.

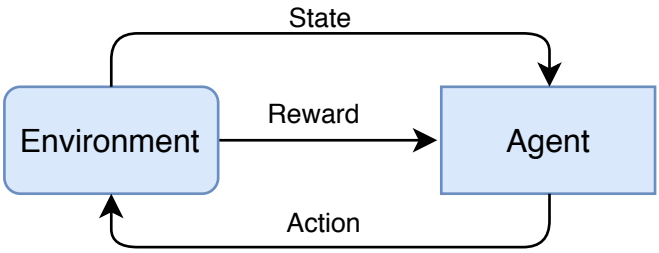


Figure 3.1: The basic architecture of a RL-model.

In addition to Environment and Agent, a Reinforcement Learning system also contains four sub-elements:

- Policy - defines how the agent behaves at a given moment.
- Reward signal - defines the goal of a reinforcement learning problem. This is connected to the policy, since if an action is associated with a negative reward, in a future situation

the policy might be changed and another action is chosen.

- Value function - specifies what is good in the long term. The value of a state is given by the sum of the rewards that an agent can expect to collect in the future derived from its actions, from that exact state.

- Environment model - allows to deduce the behavior that the environment will have.

3.1.1 Reinforcement Learning Algorithms

RL algorithms can be categorized in two types: model-based method or model-independent method. The first one is related to learning of a model of the environment which is based on the observations that the agent does. The model learns the probability of transition, $P(s'|(s,a))$, of the pair, current state s and action a, to the next state s'. This model is used to choose the best policy. The second method does not need to learn an environment model because it depends on trial-and-error to improve its knowledge. Examples of reinforcement learning algorithms are Q-Learning [16], SARSA [29], R-Learning [17] and H-Learning [18]. The Q-Learning, SARSA and R-Learning algorithms are model-independent methods, while H-Learning is a model-based method.

The performance of each of these algorithms is sensitive to their respective parameters, and it is necessary to adjust them in order to reach the best performance [18].

3.2 Q-Learning

The Q-Learning algorithm developed by Watkins [16], is an iterative method used to calculate the cost, Q(s,a), of a given action a, in the state s. The selection policy of a particular action is chosen using a function called Q-function given by:

$$Q(s,a) \leftarrow (1-\alpha) \cdot Q(s,a) + \alpha[r + \gamma \cdot \max_{a'} Q(s',a')] \tag{3.1}$$

This equation made use of the Bellman equation, getting two input parameters, the state s and the action a. γ is the discount factor, α is the learning rate, r is the immediate reward received when the agent makes a transition from a state s to a state s' after performing an action, and $max_{a'}Q(s',a')$ is the expected reward of the next state.

The purpose is to maximize the value of the Q-function (Qvalue). Q-Learning consists of 5 steps: it is created and initialized to zero what is called Q-Table or Q-Matrix, whose definition is such as Q-Matrix[s,a]=Qvalue. An action is selected and executed. To this

action will be associated with a reward value, which when determined, the Update of Q-Matrix is done using (3.1). After the update, the loop repeats itself by executing an action again. This matrix saves maximum expected reward, Qvalue, for an action in each state. Thus, the agent only needs to search in the Q-Matrix, what action to perform.

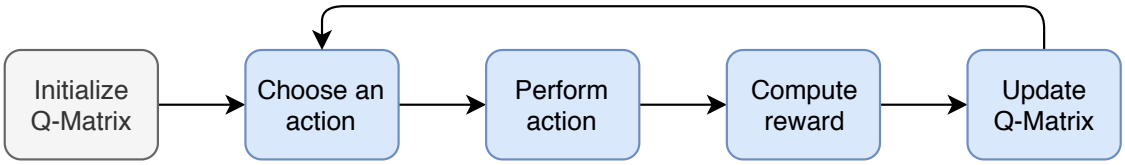


Figure 3.2: Steps of the Q-learning algorithm.

3.3 SARSA Algorithm

SARSA algorithm [24, 29] is seen as a variant of Q-Learning. Both algorithms are designed to evaluate the optimal Qvalue. The function of updating the Qvalues is given by:

$$Q(s,a) \leftarrow (1-\alpha) \cdot Q(s,a) + \alpha[r + \gamma \cdot Q(s',a')] \tag{3.2}$$

The difference exists in the selection of action a'. In SARSA the action is chosen based on the current policy (actual action) and in Q-Learning the action chosen has the highest reward, since the *max* operator is used.

3.4 Costmap

The local planning of a path estimated by a robot is supported by a map called costmap [30]. This type of map can be used in global navigation to find a path to reach a goal or in local navigation to avoid obstacles for example. It is a map that contains relevant information about the environment, i.e., it is a representation of the environment basing on occupancy grid (each cell can be defined as a empty or occupied).

There is also a variant of costmap, called layered costmap [31] consists of a master map that is updated according to the processing of different types of data in each of the layers that constitute it, being characterized as a hierarchical map resulting from the joining of sub costmaps.

Chapter 4

Developed work

This chapter will present how the components for the developed method were designed and implemented, as well as the reasoning for the proposed solutions. Different types of diagrams are used to model the system, explain the architecture, and the RL model's behavior.

4.1 Proposed RL approach

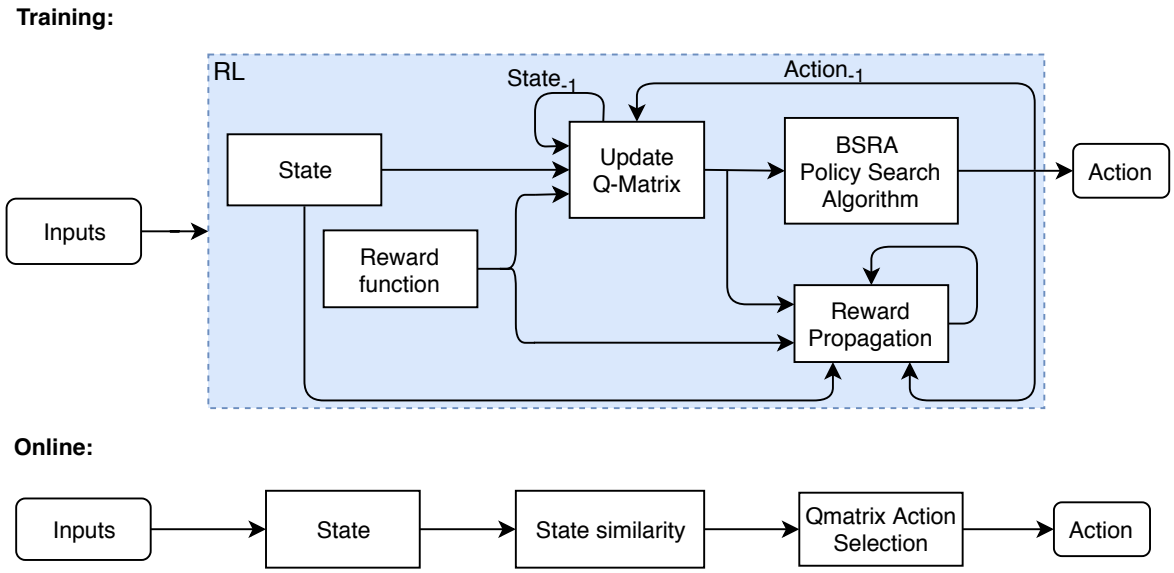


Figure 4.1: Proposed RL approach architecture.

The proposed system consists of two stages: training stage and online stage. Both stages have the same inputs (environment representation and robot's pose) and the output is an action that is converted in speed commands. As the training is performed, the Q-Matrix values are updated based on the current state, the previous state, the action previously executed and the reward obtained by the reward function. Two algorithms denoted as Biased

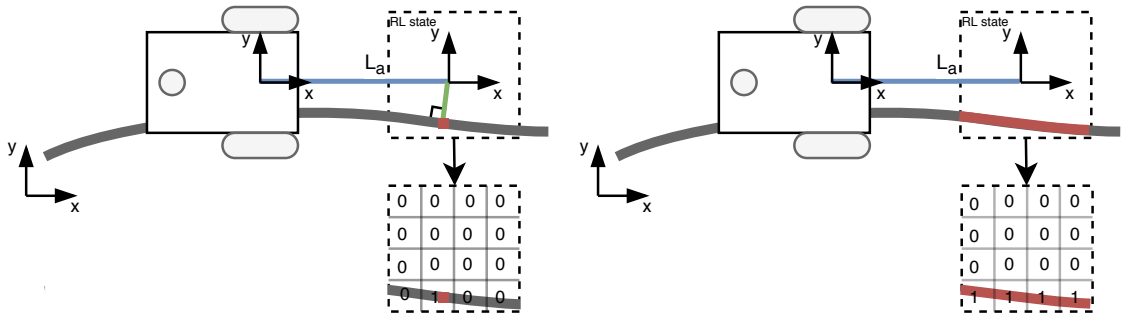
Sample-based Robot-Action (BSRA) Policy Search and Reward Propagation, respectively, were developed to help with the selection of the actions. When the training stage is over, the online stage follows. In this stage, the inputs are used to obtain the state where the agent is. The state is subjected to a comparison test (State similarity) to evaluate its similarity to the already explored states. After that, the action, of the Q-Matrix that presents the highest Qvalue for that state, is selected.

4.2 State set

A state in the RL model describes the current situation. In this dissertation a state is defined as the addition of two strings. One is extracted from the environment representation and the other is extracted from the objective path representation. The string related to environment is obtained using a costmap-like representation, which has a value of 0 when there are no obstacles and 1 in the presence of obstacles. The string related to the objective path use a similar representation. A map is obtained, which the center is a point obtained from a projection of the center of the platform. The projection technique is based on a technique named **Lookahead** [32].

Figure 4.2 presents two representations to determining the second part of the string. In Fig. 4.2a, a classic path following approach is presented where the error is calculated in front to ensure that the robot can control and minimize the error to the target trajectory. In this approach, one point, perpendicular to the path, was calculated (closer to the control point) and three errors were calculated: longitudinal, lateral and angular. Using these errors, it was possible to quantify each one and define them as a state of RL. However, quantifying these variables could complicate the learning process since could generate multiple states with small variations of the robot's pose. On the other hand, the map projection point does not need to be quantified and can be represented on a grid map, simplifying the process. This approach however does not represent the curvature of the path ahead. For this purpose, the representation in Fig. 4.2b was also proposed. It is important to mention that to guarantee invariance of the state to the robot position, the state is calculated in the control referential, i.e., the projected point on the path and the path patch are represented at the control point of the robot.

The strings are then created based on the values of the map cells, in the case of the environment as a function of the presence of obstacles or not and in the case of the path, the existence of path or not.



(a) (b)

Figure 4.2: Two different representations to defining the second part of the string that constitutes the state.

Figure 4.3 shows how a state can be defined using the representation in Fig. 4.2b . In both maps, the cells that have a obstacle/path, have value 1 and value 0 otherwise.

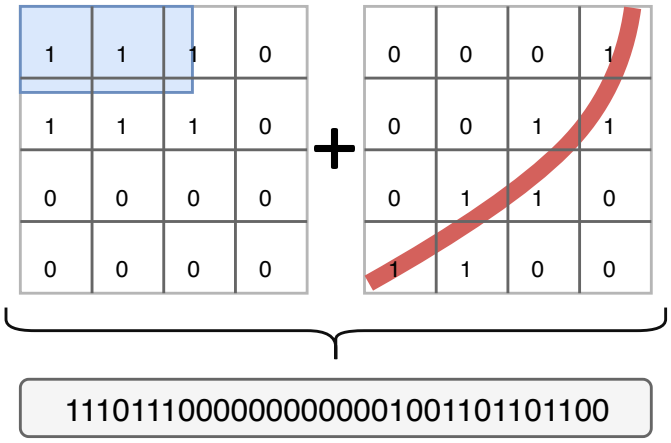


Figure 4.3: Possible state obtained given the configuration of obstacles and path.

4.2.1 State similarity

As mentioned above, one of the problems identified in relation to states is the fact that small variations of the robot's pose generate multiple states. So a small difference in pose generates different states (should be the same state) leading to the execution of different actions. To solve this, a common similarity measure called Cosine similarity [33] was used. In this way, states (A and B) with similar representation were compared, with the result of the measurement being between 0 and 1. The closer to the value 1, more similar the states are and the closer to 0, more different they are. The Cosine similarity is given by:

$$Similarity = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{4.1}$$

This is applied to all known states, i.e., considering the current state, if it has not yet been explored means that it does not exist in Q-Matrix, so it is compared to all the Q-Matrix states. If there is no state that is similar to the current state, this state is added to Q-matrix.

4.3 Actions set

An action in the RL model is what an agent can do in each state. In this dissertation a set of actions were defined, consists of speed pairs. Each pair contains a linear speed, v and an angular speed, ω. The total number of actions comes from all possible combinations of values given *a priori*. So, if the agent selects an action, a linear and angular speed is associated with it.

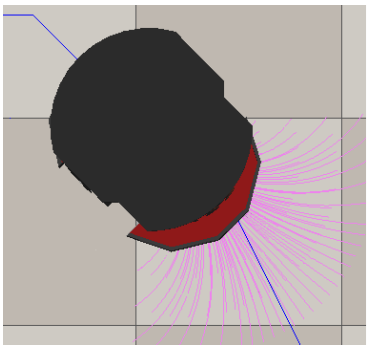


Figure 4.4: Possible actions (pink) that the agent can perform while navigating the environment.

4.4 Path

A path P, is defined, that connects an initial position and a final position (goal), consisting of intermediate points that are connected to each other, creating line segments. The generated path can be smoothed using cubic splines.

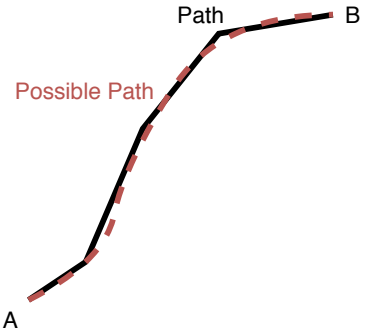


Figure 4.5: Path (black) defined *a priori* and a path that the agent could take (red).

In order to compute the distance from the robot to the line it is necessary to calculate the nearest line segment and then calculate the minimum distance to that segment. The distance from a point $P(x_0, y_0)$ to a line s: $ax + by + c = 0$ belonging to the same plane is given by:

$$d = \frac{|a \cdot x_0 + b \cdot y_0 + c|}{\sqrt{a^2 + b^2}} \tag{4.2}$$

4.5 Reward function

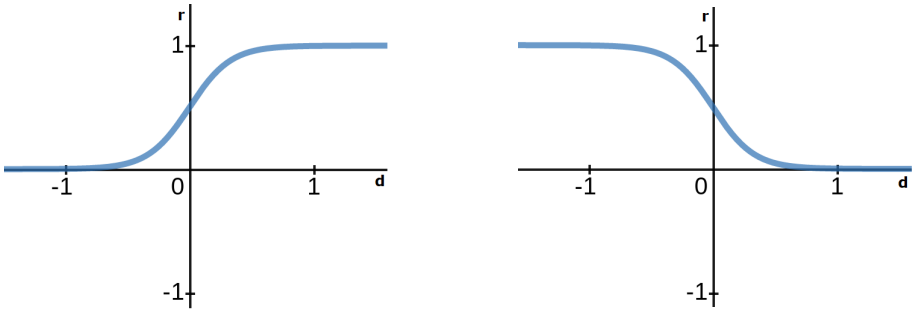
The reward function allows the agent to receive the reward from the environment after an action has been performed. This function returns a value defined by the lowest value of multiple sub-rewards. The sub-rewards are obtained using distances to:

1. **Goal:** the Euclidean distance is computed between the position of the robot and the end point of the path.

2. **Path:** the distance is computed between the position of the robot and the path, that is, each segment of the path is considered as a line and the distance to each segment is determined by (4.2). The shortest distance will correspond to the segment that is closest to the platform and so will be the value of the distance to the path.

3. **Obstacles:** the distance to the nearest point to the robot that belongs to an obstacle is determined.

Sigmoidal functions, represented in Fig.4.6, are defined to obtain the reward value as a function of the distance.



(a) (b)

Figure 4.6: Sigmoidal functions used to determine the reward as a function of a distance: (a) Function used to determine the value of the reward as a function of the distance to an obstacle. (b) Function used to determine the value of the reward as a function of the distance to a goal or path.

The equation to define a sigmoidal function is given by (4.3). This function contains constants that can be changed to set the reward function to the user's preference. To obtain sigmoidal functions as shown in Fig. 4.6, it is only necessary to change the values of the constants, the constant a allows to define the direction of growth and the inclination of the function, Kx allows to obtain or not negative rewards and c establishes the location of the center of the function.

$$f(dist) = Kx + \frac{1.0}{1.0 + e^{(-a*(dist-c))}} \tag{4.3}$$

This reward function could have been defined using one of the approaches listed in the works that are presented in Table 2.3. It was defined in function of distances because it was simple to evaluate.

4.6 Q-Matrix

In a common RL model the states, actions and Qvalues are stored in a matrix called Q-Matrix. An example of a Q-Matrix is shown in Fig. 4.7. Its rows consist of states and their columns of actions. The Qvalues are independent of each other because there is only one Qvalue corresponding to one pair (*state*, *action*). The Qvalue for each pair is obtained by (3.1), presented in Chapter 3.

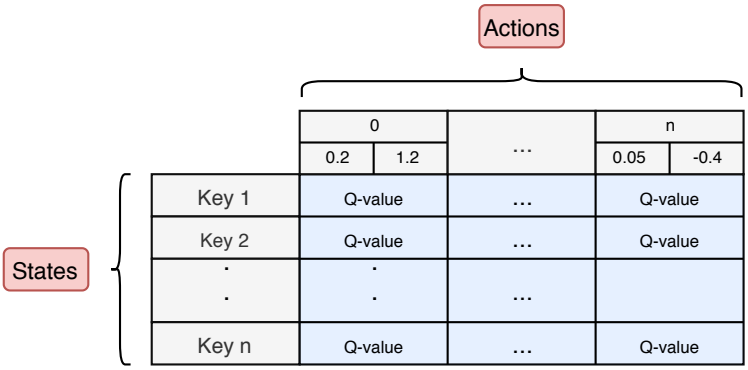


Figure 4.7: Example of a Q-Matrix, displaying the main components such as states, actions, and Qvalues.

In the implementation was defined a tree to store the states, because if the complexity of the environment increases, with a matrix representation there are memory problems since there are no resources needed to save all the states and, using a tree, it is also possible to increase the search efficiency of the states.

It is important to mention that this Q-Matrix is essential to determine the action to

execute. In the online stage the action a for a given state s is chosen by the following equation:

$$a \leftarrow \arg\max_{a \in Q(s,:)}(Q(s,a)) \tag{4.4}$$

4.7 Reward propagation

Figure 4.8 demonstrates a common problem in RL applications. When the agent receives a negative reward, it can also be a factor of previous actions and not of a single action. A temporal window is considered, the agent, from state S_1 to S_7, executed a set of actions until the reward became negative. To prevent the agent from executing with more regularity the actions that caused that reward, it is important to establish a penalty in the previous states. So the update of the Qvalues is given in the same way by (3.1) but to the reward r is applied the penalty as below:

$$\bar{r} = r * \frac{(N - L_d)}{N}$$

where N is the size of the temporal window and L_d is the distance, in iterations, to the event. States that are closer to the event have a higher penalty than states that are more distant.

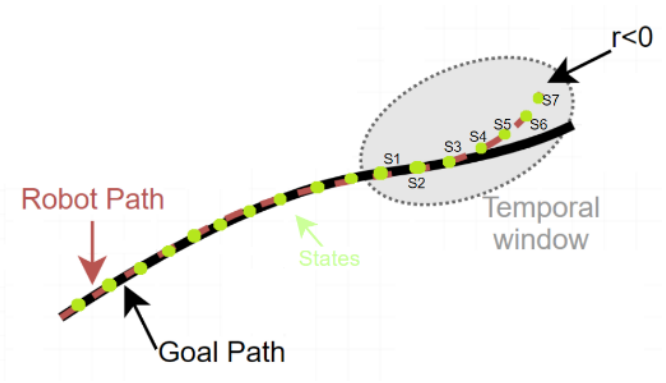


Figure 4.8: Reward propagation problem. Red path is the path that robot made, black path is goal path and green points are the states where robot been.

4.8 Biased Sample-based Robot-Action Policy Search Algorithm

The Biased Sample-based Robot-Action (BSRA) Policy Search Algorithm was developed to ensure that the agent does not simply follow the current policy since it can get stuck in

a suboptimal solution, i.e, the agent may think that an optimal policy was already found and not search for new solutions. In ϵ-greedy with random exploration, for problems with a high number of actions it is difficult to converge, also considering the problem of delayed reward (e.g., the robot is moving away from the path but still receives "positive" rewards). Contrary to the BSRA algorithm, the ϵ-greedy does not consider the knowledge of how to drive the robot. When the agent receives a positive reward, this is not always the biggest existing reward for that particular state transition. The action is chosen according to that reward, although it might not be the one that has the highest reward value.

The algorithm has an action as output and four inputs: the map (a local map), the path, the Qvalues, and all possible actions. For each action contained in the set of possible actions, is verified the Qvalue (line 3) and possible collisions when performing that action (line 12). If the Qvalue associated with this action is negative, it means that the action has already been chosen before and is an inappropriate action so it is discarded. If the action causes the robot to collide with an obstacle, the action is also discarded. It is computed a score using the kinematic model (line 16). The score and the factor of choosing that action are combined (line 18). Done this for each action, the action is chosen using a multinomial function (line 20 and 21).

Softmax function is a function that can convert a pair $(state, action)$ into a factor. At the output, the values are between [0,1]. The factor of an action being chosen given the Qvalues of a state is determined as follows:

$$Pr(a_i) = \frac{exp(Q(s_t, a_i)/T)}{\sum_{k=1}^{n} exp(Q(s_t, a_k)/T)} \tag{4.5}$$

where T is the temperature factor (control randomness of predictions).

Using the softmax function, after some time, the factor of choosing a certain action tends to get closer and closer to the value 1 or 0 because the Q-Matrix values are being updated. If an action starts to be considered good, the Qvalue increases and the respective factor value also increases. This allows to choose or reject more actions because of the dispersion of factors.

Multinomial Resampling function consists in: A score value is associated with each action. The score can be seen as a weight (the term weight is usually used when using a particle filter). Actions are sorted in a ascending order as function of the scores and a threshold (m_t) value is established. Once this value is set, the action is randomly chosen from the actions whose scores are in the interval $[m_t, 1]$. Using this function it is guaranteed that the chosen action is in the scope of the best actions.

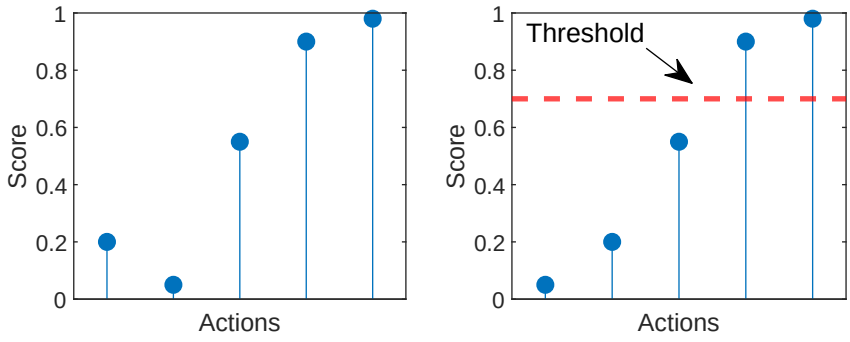


Figure 4.9: Graphics showing a example of the multinomial resampling step.

Pseudo-code for the BSRA Policy Search Algorithm, together with softmax function and multinomial resampling, is shown in Algorithm 1.

Algorithm 1: Action selection algorithm for the training stage of the RL model

Input: Map, Path, Qvalues, Actions

Output: Action

$S_c \leftarrow 0$
foreach $a \in Actions$ **do**
  if *Qvalue(Qvalues,a)<0* **then**
    $S_c \leftarrow S_c \cup 0$
    continue
  end
  c $\leftarrow$ 0
  localpose $\leftarrow$ (0,0,0)
  for *i=0 to N* **do**
    localpose,displacement $\leftarrow$ prediction(localpose,a) ; `// Kinematics equation`
    d $\leftarrow$ distanceToPath(localpose,Path)
    if *Map(localpose) is occupied* **then**
      c $\leftarrow$ 0
      break
    end
    c $\leftarrow c + (K_d/(K_e + d) + K_A * displacement$; `// Cost function`
  end
  $S_c \leftarrow S_c \cup$ c*P(a) ; `// P(a) is softmax function`
end
$N_s \leftarrow$ MR(S_c) ; `// MR is multinomial resampling function`
Action $\leftarrow \underset{k>m_t}{\sim} (N_s)$
return *Action*

Chapter 5

Validation Platforms and Software Implementation

This chapter presents the different platforms used to validate the work:

- InterBot Platform [2].
- Virtual platform present in the V-REP simulator.

All software implementation necessary for the correct operation of the local navigation method is presented.

5.1 InterBot Platform

InterBot (see Fig. 5.1) is a service robot, developed in the ISR-UC particularly for indoor navigation, that allow collaborative human-robot interaction (HRI) and perform given tasks. This section provides a brief description about the hardware and software architectures.

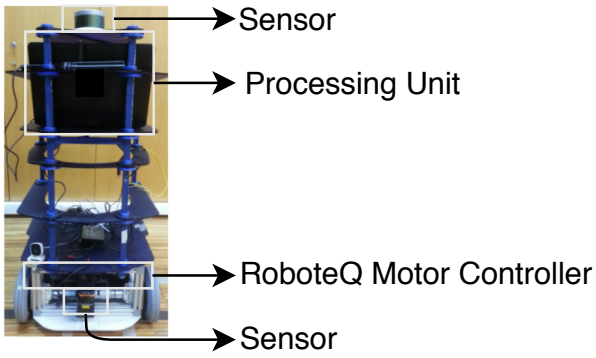


Figure 5.1: InterBot platform with its main components: sensors (Velodyne VLP16 and 2D laser Hokuyo's UTM-30LX), Processing Unit (Laptop), and RoboteQ Motor Controller.

5.1.1 Hardware architecture

Hardware architecture of InterBot platform can be divided in three main components as shown in Fig.5.2.

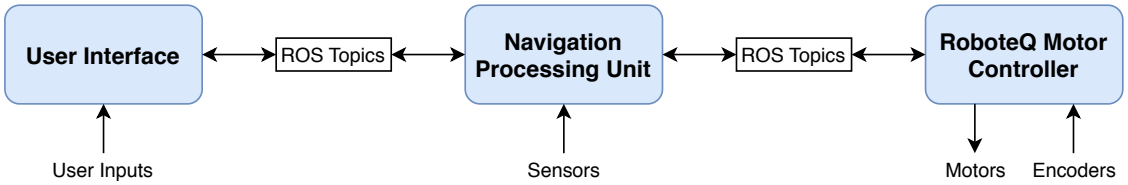


Figure 5.2: InterBot hardware architecture displaying the inputs and outputs of their main components.

5.1.1.1 User Interface

The InterBot platform has several human-robot interfaces to controlling it. A user can control via an on-board portable device, a remote station or a joystick, allowing a user to interact with the platform.

5.1.1.2 Navigation Processing Unit

Unit responsible for all the processing, in this case, a laptop, which manages all the information captured by sensors, encoders and all the software architecture present in the InterBot required for the localization, mapping and navigation.

5.1.1.3 RoboteQ Motor Controller

The platform is equipped with three wheels, two are driven by their own DC motors while the third is a small stabilizer wheel. Each wheel contains an encoder on its axle that returns pulses to a RoboteQ controller. This controller is also responsible for receiving speed commands from the previous unit.

5.1.2 Software architecture

Figure 5.3 presents the software architecture of the InterBot platform. The high-level component is the Navigation Processing Unit, where its modules are programmed in ROS environment. There are files designated as launch files responsible for setting configuration parameters of the different components that compose the platform, allowing them to serve

their purpose. More information about the whole architecture of the InterBot platform can be found in [3].

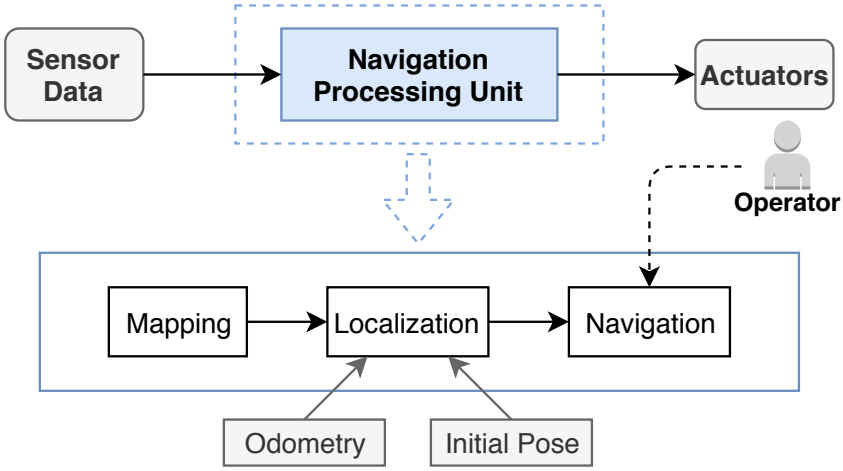


Figure 5.3: InterBot software architecture displaying the main modules.

5.2 Virtual Platform

5.2.1 V-REP

The V-REP [34] framework is a realistic simulator that allows users to create any desired scenario and simulate any situation in it. The program includes a large range of options in terms of components (e.g. sensors, actuators) which can be plugged into the different existing robots and tools. Each object of the scene can be operated/controlled from the V-REP framework. It can set a communication with the outside world using more than one interface such as Remote API, ROS interface, BlueZero interface and others.

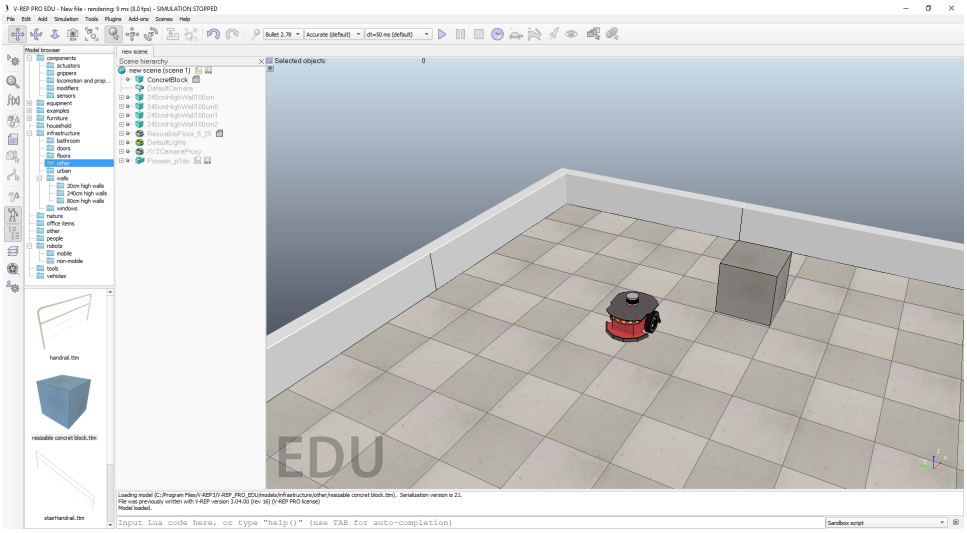


Figure 5.4: Example of a scene from simulator V-REP.

The ROS interface was used because the InterBot software is based on the ROS framework. Because the developed method uses the same framework, it can be tested and used in a real or a virtual platform. The interaction between the simulator and the outside world, in this case, the ROS package that contains the method developed, is established by messages between nodes.

In order to test the proposed method, it was necessary to modify the simulation environment previously since the objects present in the simulator are not, by default, prepared to be used with the ROS framework. The modification does not only involve obtaining specific scene information/parameters through a list of V-REP functions that allow, for example, to obtain the position of a certain object to an arbitrary reference, but is also necessary to insert code lines responsible for initializing, publishing, and subscribing topics.

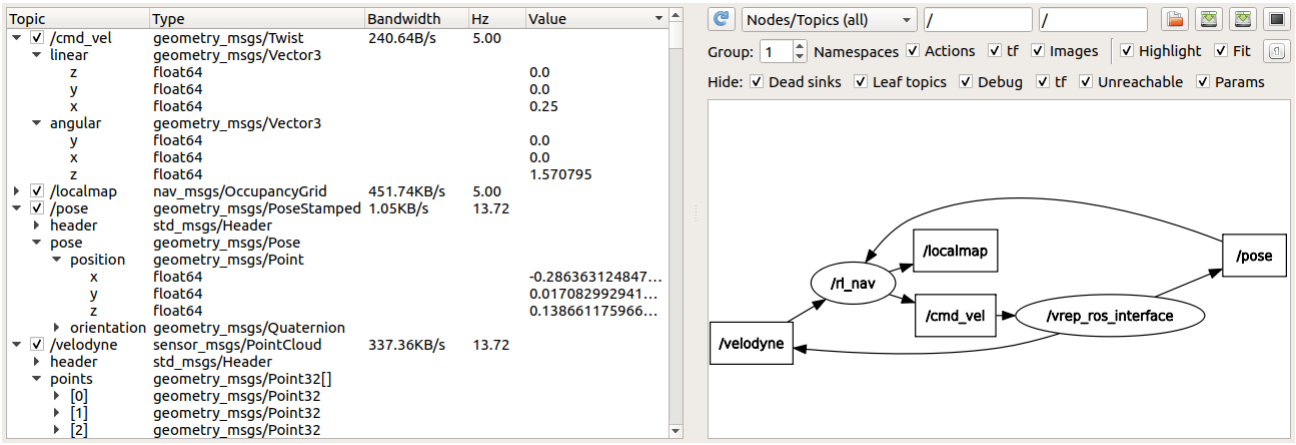


Figure 5.5: Overview of the interaction between simulator and Outside World with the topics that are published and subscribed. V-REP publishes topics as /pose and /velodyne and subscribes topic /cmd_vel. RL_nav package publishes topic /cmd_vel and subscribes topics /velodyne and /pose.

5.3 Software implementation

5.3.1 Robot Operating System

ROS [35] is not considered a real-time operating system, although it provides functionalities of an operating system. It allows code reuse, communication between processes on multiple machines and package management. Among others, ROS also contains a wide range of libraries, using C++ and Python as main programming languages. One of its major advantages is that it is Open Source. Basically the concept of the ROS framework, is based

on nodes, messages, topics and services. Nodes are seen as processes that can communicate through messages based on a publication-subscription model. A service is defined by a pair of messages, a request message and the respective reply message.

In this type of system, where multiple machines/computers are running ROS, there is always a machine called a master. The master has the function of initializing all services so that the nodes can interact with each other.

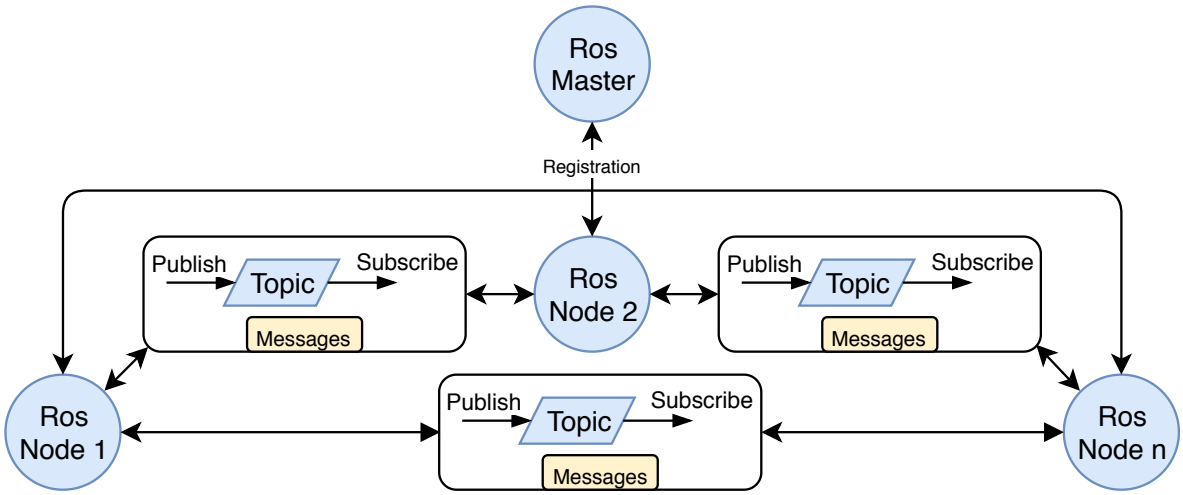


Figure 5.6: Representative diagram of the ROS framework.

5.3.2 RL_nav package

A ROS package named RL_nav was created and contains the implemented local navigation method described in Chapter 4. This package enables the communication/interaction with a real or a virtual platform. All the code for the RL_nav package was written in C++ language, using Qt Creator, operating on a ROS architecture.

Figure 5.7 shows a block diagram with the main nodes and the respective topics that the proposed method can use. The **velodyne_node** and **hokuyo_node** are software drivers provided by the ROS community, to be used with a Velodyne and a Hokuyo laser range finder, publishing a /velodyne_points and /scan topic message of the obtained laser data, respectively. The **RoboteQ Motor Controller**, which was developed for the ISR intelligent wheelchair Robchair and adapted to InterBot, receives the pulses and subscribes the /cmd_vel topic, and the **RoboteQ_node** publishes the odometry data through the /odom topic. Topics such as /pose and /velodyne are published by VREP and it subscribes the topic /cmd_vel.

Table 5.1 presents all topics that can be used by the RL_nav package.

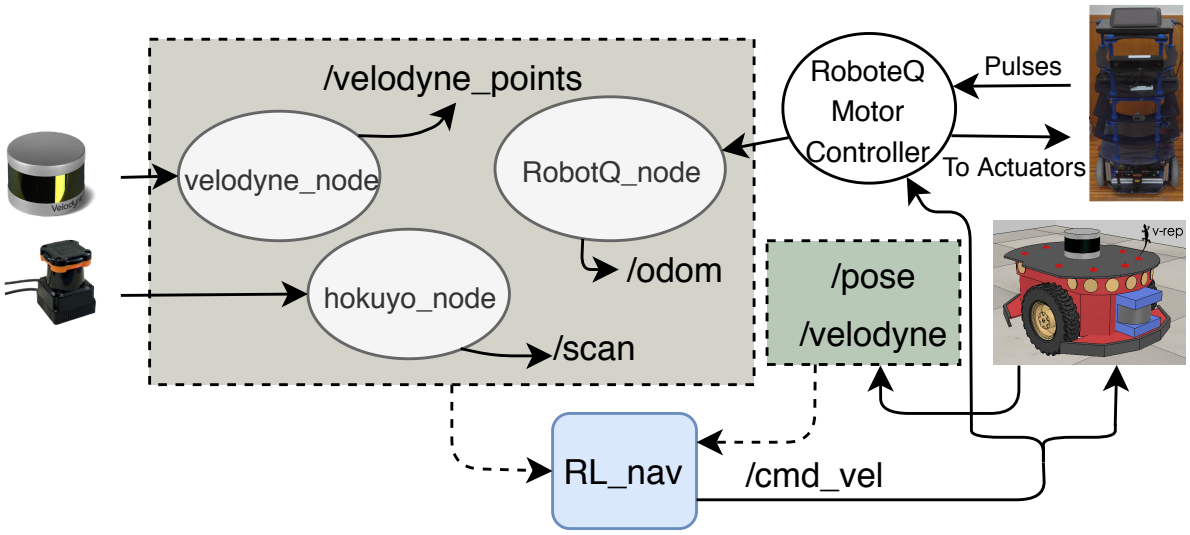


Figure 5.7: Diagram of the all topics that can be published and subscribed by RL_nav.

Topic	**Message type**	**Message main content**
/odom	nav_msgs/Odometry	geometry_msgs/PoseWithCovariance pose geometry_msgs/TwistWithCovariance twist
/velodyne_points	sensor_msgs/PointCloud2	uint32 height uint32 width sensor_msgs/PointField[] fields uint8[] data
/scan	sensor_msgs/LaserScan	float32[] ranges float32[] intensities
/pose	geometry_msgs/PoseStamped	geometry_msgs/Pose pose
/velodyne	sensor_msgs/PointCloud	geometry_msgs/Point32[] points
/cmd_vel	geometry_msgs/Twist	geometry_msgs/Vector3 linear geometry_msgs/Vector3 angular

Table 5.1: Topics that are published and subscribed by RL_nav package, as well as the type of message each one uses and its content.

Figure 5.8 displays the sequence of events related to the training stage. The agent always starts learning from an initial condition. During the training process, the value of the reward is constantly verified since the objective is indirectly to always maximize this value. If in any circumstance it has a negative value, the agent (robot) is sent to the initial condition. As the training is performed, the Update Q-Matrix uses the reward value and the current

pair (*state*, *action*) to update the Qvalues of the previous pair (*state*, *action*) (3.1). As long as the agent does not reach the goal, and the reward is negative, he is sent to the initial condition again, and the learning continues until he reaches the goal, with a positive reward. After a significant number of successful episodes, the learning stage is concluded.

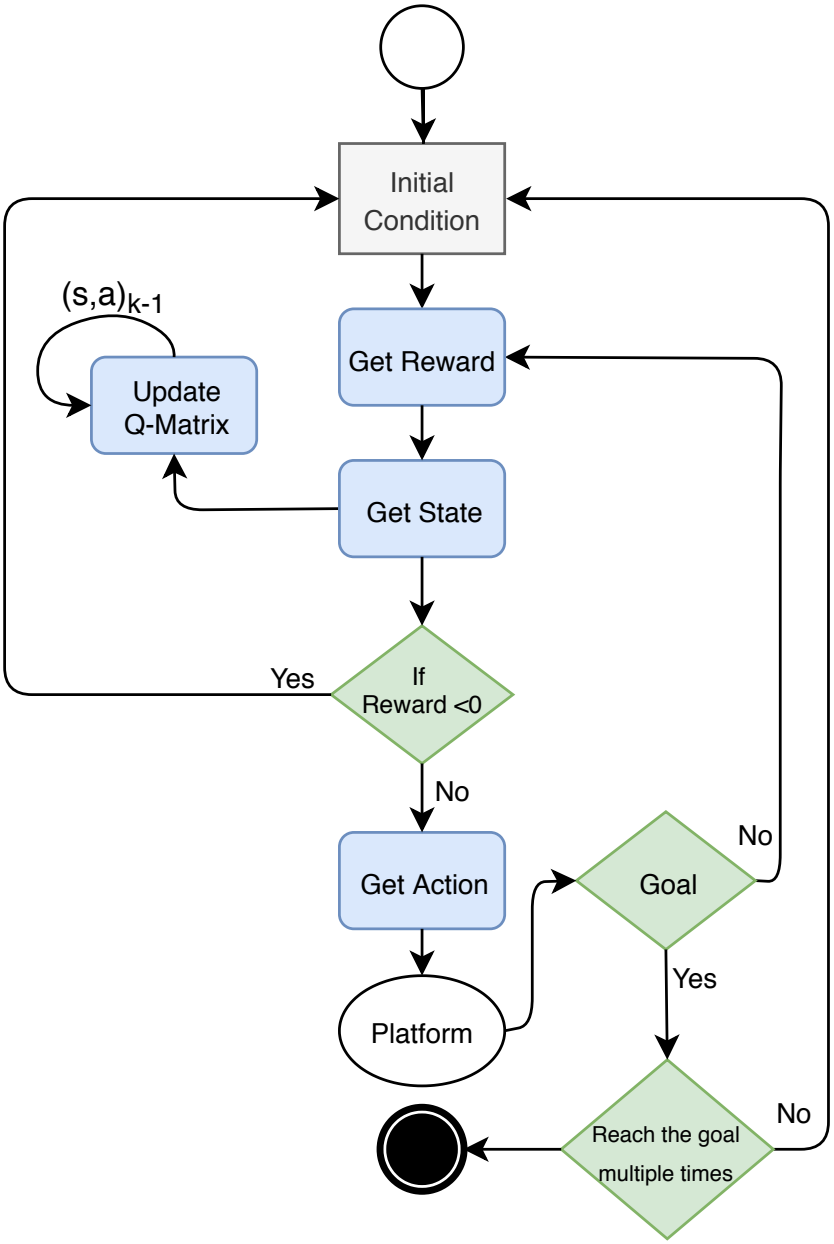


Figure 5.8: Diagram of the sequence in training stage.

Chapter 6

Experimental Results

This chapter presents the main experimental results obtained from different test scenarios in a simulation environment. To test and evaluate the method, presented in Section 4.1, six tests were performed:

1. Agent with only 3 possible actions for two simple paths using the two types of representation present in Fig. 4.2.

2. Agent with 162 possible actions for six scenarios, using the representation from Fig. 4.2a for paths with either segment lines or cubic splines and the representation from 4.2b for paths with cubic splines.

3. Agent with only 3 Actions vs 162 Actions available.

4. Agent with 162 possible actions for a scenario that has obstacles near the path.

5. Use of an user's driving motion commands while controlling the InterBot platform as the basis of an RL-model, tested in six scenarios, whose paths are represented by cubic splines using the representation presented in Fig. 4.2a.

6. Agent with 162 actions available for 2 scenarios in a simulator developed in the ISR-UC using the representation presented in Fig. 4.2a.

After a series of tests, the values for the constants/variables used in the method are presented in Table 6.1. Throughout the presentation and discussion of the results obtained, the term Representation 1 ou 2 will be used, which refers to the representations present in Fig. 4.2.

In the distance graphics presented below, these will have as axes: x - number of episodes and y - meters.

Table 6.1: Constants/Variables used in the experimental tests.

Constant/Variable name	**Symbol**	**Value**
Learning rate	α	0.5
Discount factor	γ	0.5
Lookahead distance(m)	L_a	0.25
Temporal window(nºiterations)	N	30
Multinomial Threshold	m_t	0.8
Gain K_d	K_d	1
Gain K_e	K_e	0.1
Gain K_A	K_A	1000
Temperature factor	T	0.1
Size map cell to state1 (cm)	$cellsize\_s1$	(7x7) 20
Size map cell to state2 (cm)	$cellsize\_s2$	(20x20) 5

6.1 Validations scenarios

Figure 6.1 presents the paths used to test and validate the developed method. Two types of path representation were used. At the top of the figure, the paths are represented by line segments, and at the bottom, they are represented by cubic splines.

These paths (line segments) could be generated by planners such as A* or classic RRT, but the paths could contain sharp curves that are hard for a non-holonomic robot. However, some planners (e.g., RRT* [11]) can use the kinematics or dynamics of the robot to create smooth paths (paths such as the ones presented at the bottom of Fig. 6.1). As the focus of the work is on local motion planning, there is a complete interest in the robot's behaviour and not in the path that is generated/found even though it could influence the model obtained.

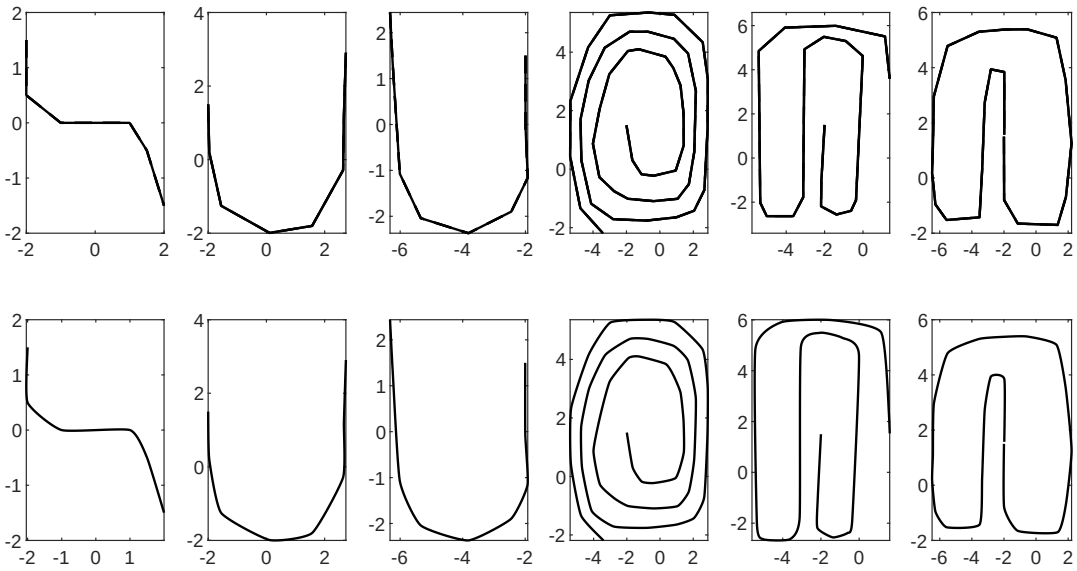


Figure 6.1: The six paths used to validate the method developed. At the top row, paths are represented by line segments and at the bottom row by cubic splines. The paths are enumerated from 1 to 6, from left to right.

6.2 3 Actions

This first test involves testing the proposed method (without the BSRA approach) in two simple paths, a straight line and a curve as shown in Fig. 6.2, allowing the robot to use only 3 actions. The 3 actions consist of: moving forward with linear speed v, and moving forward with a certain angular speed ω or -ω.

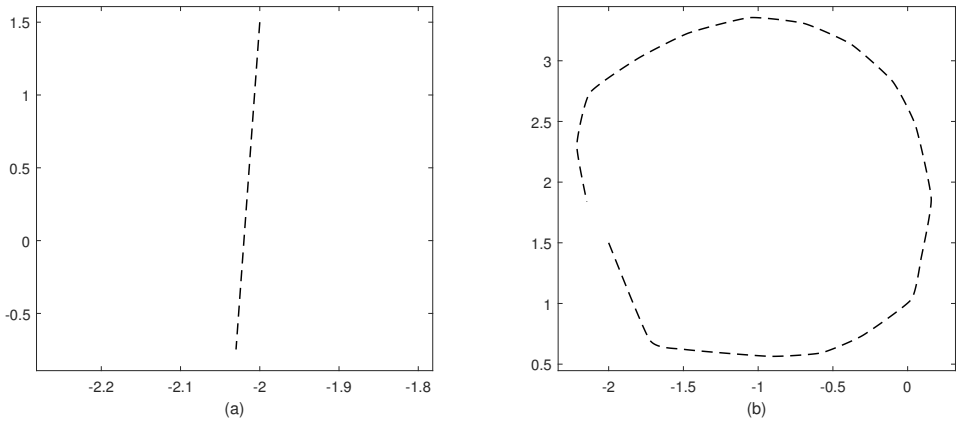
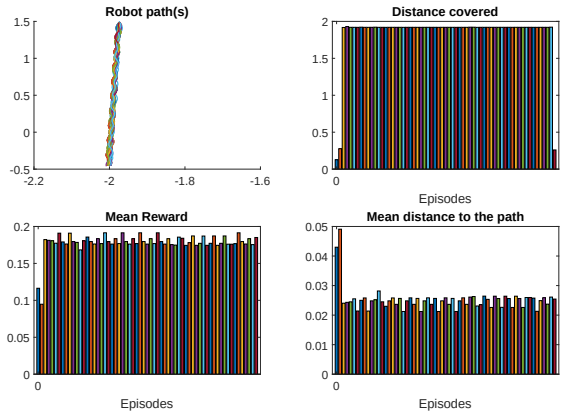
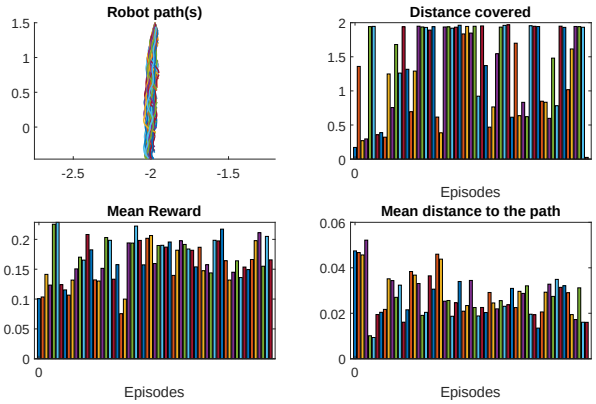


Figure 6.2: Scenarios used to test and obtain the results when the agent can use only 3 actions.

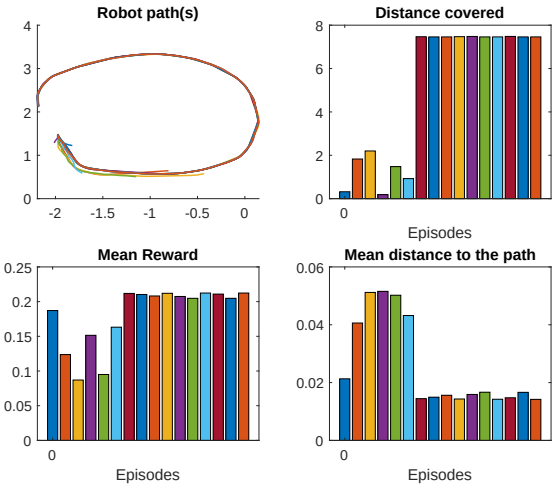


(a) Results using scenario in Fig. 6.2a with Representation 1 (Number of states: 7).

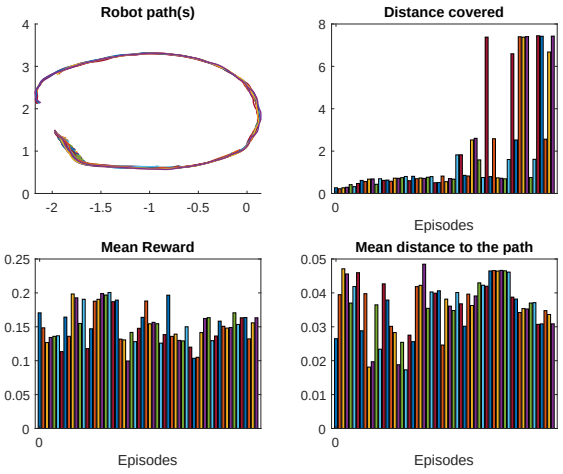


(b) Results using scenario in Fig. 6.2a with Representation 2 (Number of states: 680).

Figure 6.3: Results obtained using scenario from Fig. 6.2a for both Representations.



(a) Results using scenario in Fig. 6.2b with Representation 1 (Number of states: 16).



(b) Results using scenario in Fig. 6.2b with Representation 2 (Number of states: 1220).

Figure 6.4: Results obtained using scenario from Fig. 6.2b for both Representations.

Analyzing Figs. 6.3 and 6.4, it can be seen that robot's behavior is unstable. However, it is remarkable that it is possible to train the method for both scenarios and obtain a model for each of them since the algorithm used only RL to learn how to follow the objective paths. As for the type of representation, the use of Representation 1 gives better results because the complexity is lower (a smaller number of states). Observing at the mean value of the reward, as the number of episodes increases, it also increases until it stabilizes at a given value when using Representation 1. The same does not happen when using Representation 2 because the value, although increasing with the number of episodes, shows notable variations. As for

the metrics related to mean distances, using Representation 1, the values tend to stabilize at the end of some episodes, showing that the robot moves better with this representation.

6.3 162 Actions

In this test, the agent can use a total of 162 actions to move along a path. One of the possible advantages, without observing the results, is the fact that the robot's behavior should be more stable than when only 3 actions are used, because the robot has several combinations of speeds which can translate into a smooth movements.

Contrary to the first test, the method can not converge to an optimal policy using only RL. However, when the BSRA algorithm is added, the method (RL+BSRA) can already converge.

The idea in this second test is to train a model on Path 1 and use this model to navigate in the other paths using Representation 1. The results are presented initially for paths whose representation is done by line segments. Subsequent results are for paths whose representation is done with cubic splines.

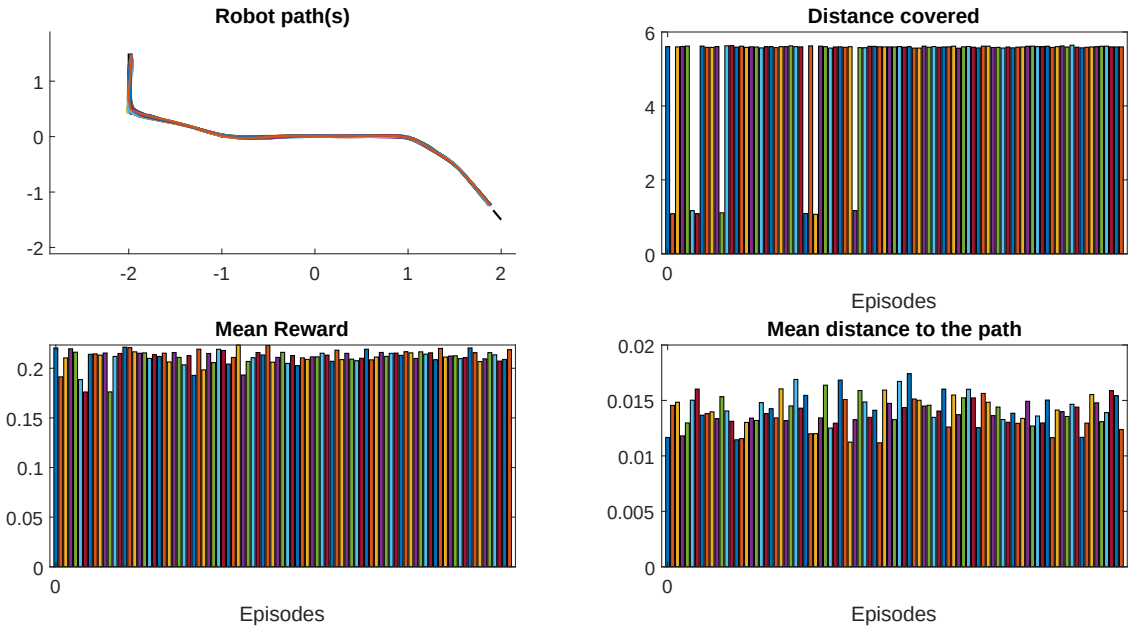
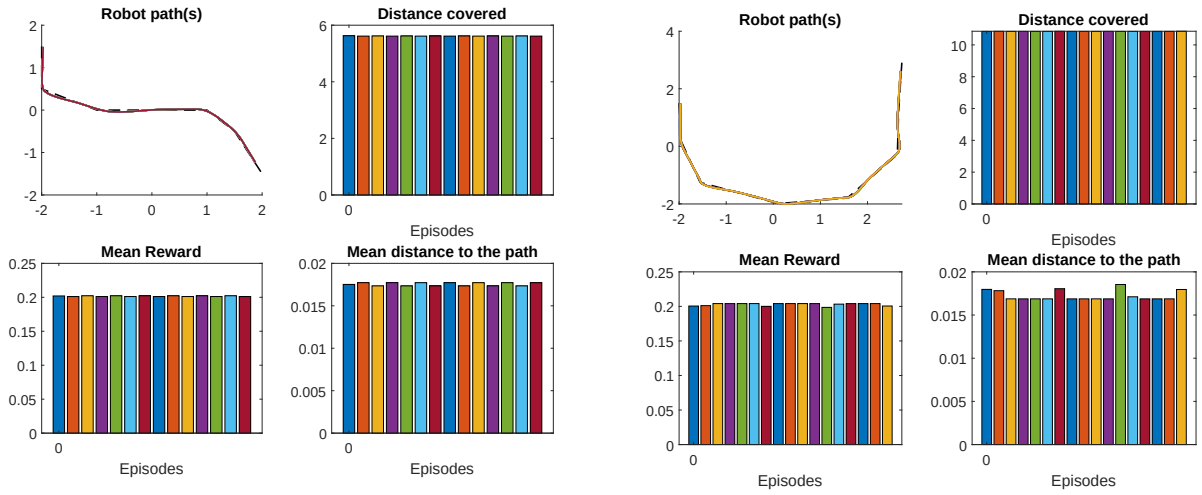


Figure 6.5: Results obtained after training to get a model (Model 1) on Path 1.

After obtaining the results related to the training performed on Path 1 as shown in Fig. 6.5, the generated model was then used in the other paths.

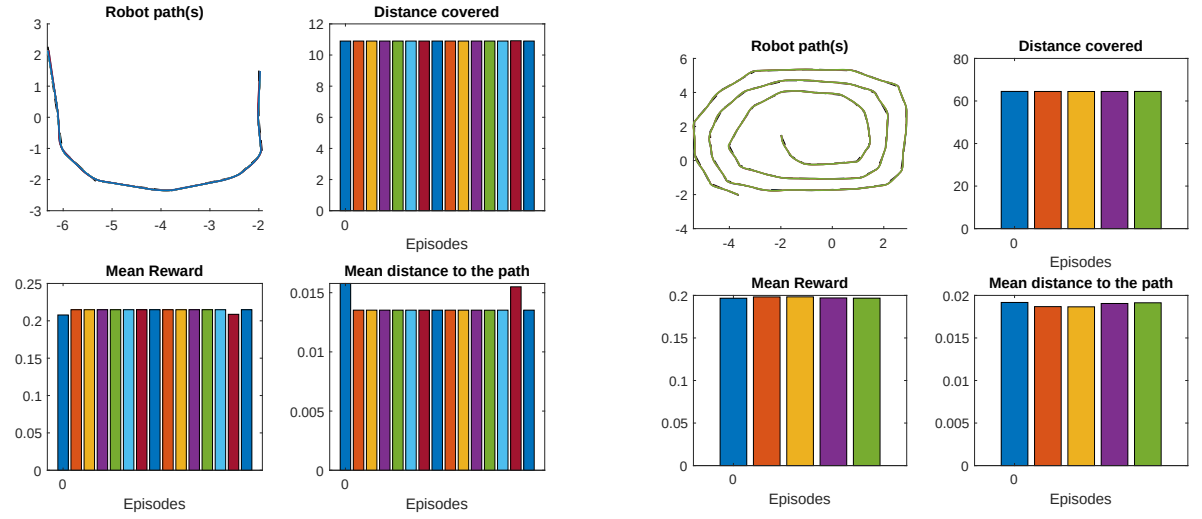
Figures 6.6 and 6.7 show the results obtained using Model 1 on Path 1 to Path 4. The results show that the model can be used to navigate on these paths. In all cases the value

of the average reward and the distance to the path become stable in the initial episodes, allowing to see that the robot follows the path correctly.



(a) (b)

Figure 6.6: Results obtained using Model 1 on: (a) Path 1 and (b) Path 2.



(a) (b)

Figure 6.7: Results obtained using Model 1 on: (a) Path 3 and (b) Path 4.

Figure 6.8 shows the results of using Model 1 on Path 5. The robot can follow most of the path correctly, failing when it reaches the last curve. This means that Model 1 fails when used for Path 5. To get around this situation, a new model called Model 5 was trained on Path 5.

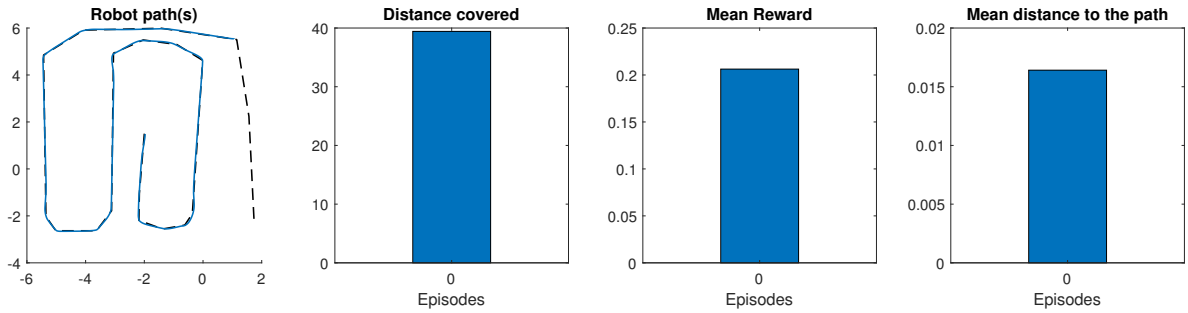
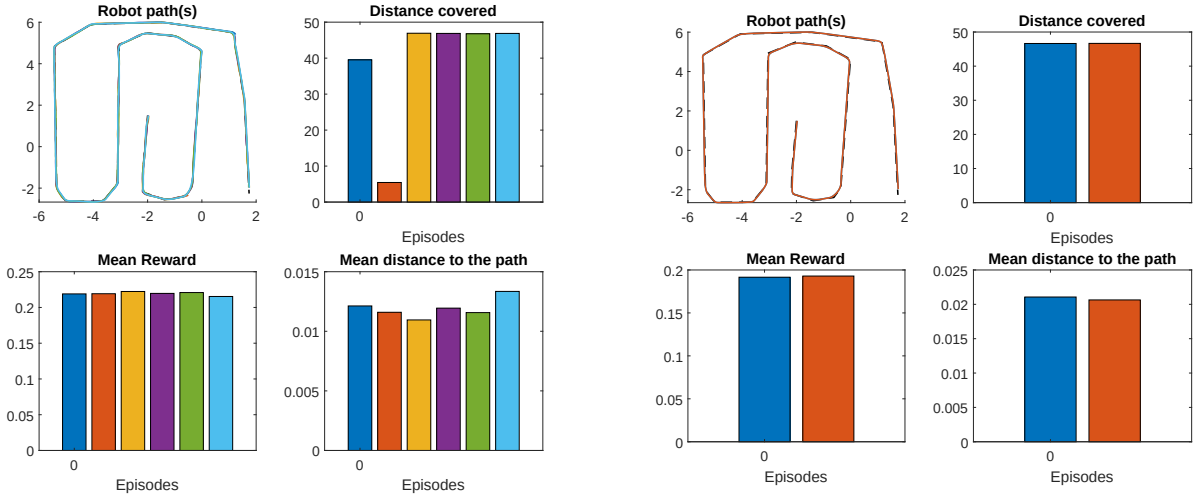


Figure 6.8: Results obtained using Model 1 on Path 5.

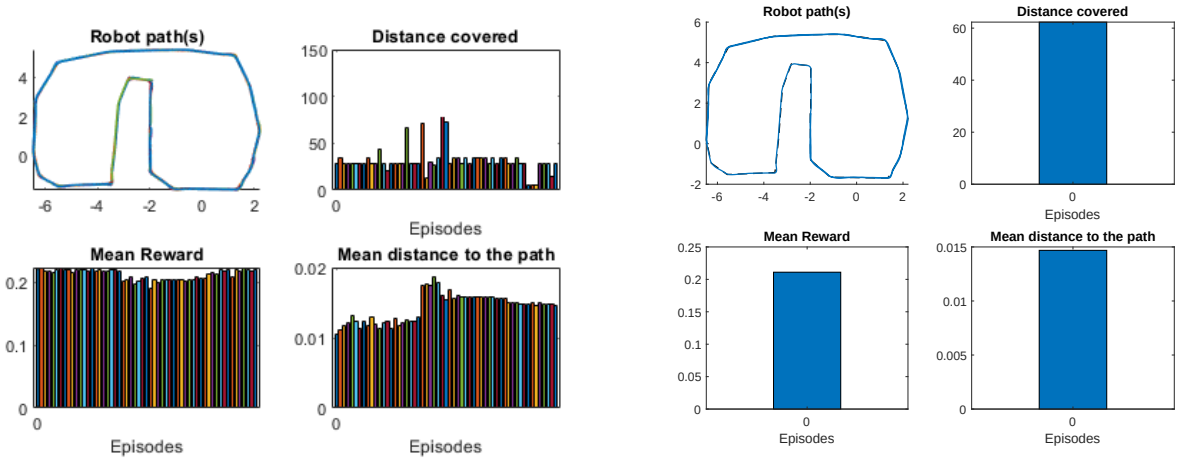
After training on Path 5 as shown in Fig. 6.9a, the results obtained using Model 5 on Path 5 are represented in Fig. 6.9b. Although during the training stage the average reward value is reasonably stable and the average path distance increases slightly with the increasing number of episodes, when model 5 is used on Path 5, the robot is able to follow the entire path.



(a) (b)

Figure 6.9: Results obtained: (a) after training to obtain Model 5 and (b) using the Model 5 on Path 5.

Figure 6.10a shows the results obtained from the training done on Path 6. The Model 5 generated previously was not enough to enable the robot to complete Path 6 certainly due to the difficulty in doing curves with almost 90º of curvature. With the new model, Model 6, the robot was already able to complete the whole path. The results obtained, using Model 6 on Path 6 are present in Fig. 6.10b.



(a) Results obtained after training to get a model(Model 6) on Path 6.

(b) Results obtained using Model 6 on Path 6.

Figure 6.10: Results obtained: (a) after training to obtain Model 6 and (b) using the Model 6 on Path 6.

Figure 6.11 presents a summary of what happened during this test. Whenever the model failed in a new path, a new model was obtained. A total of 3 models (1, 5 and 6) were obtained/trained.

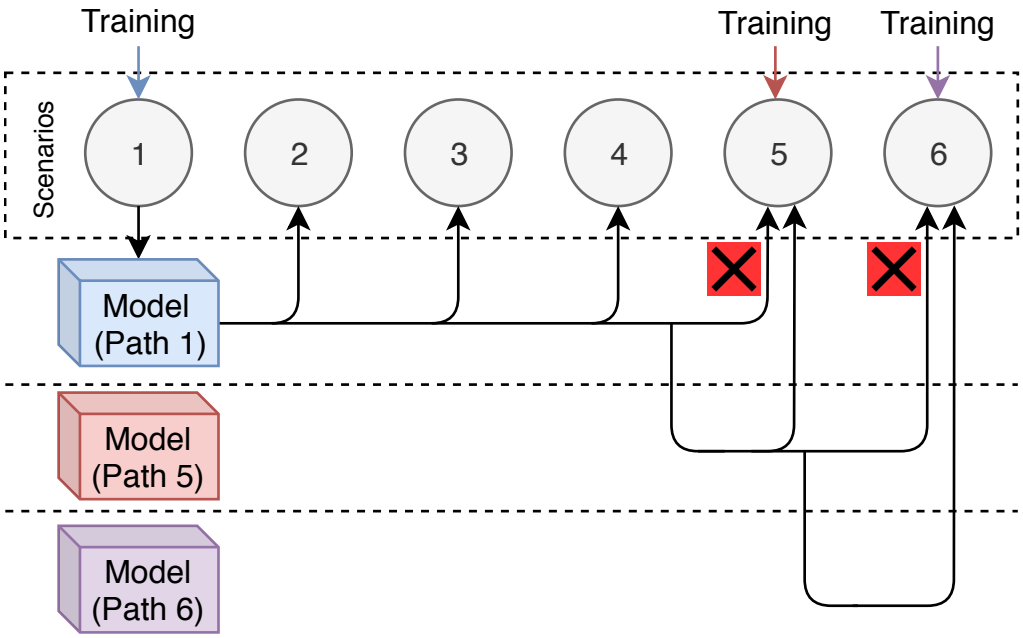
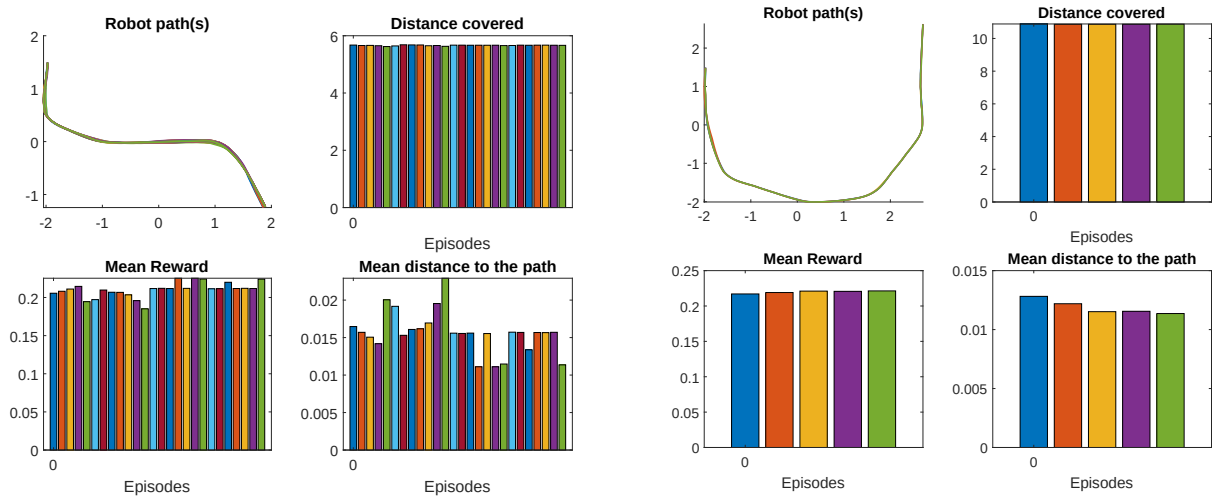


Figure 6.11: Diagram of the steps done during test 2.

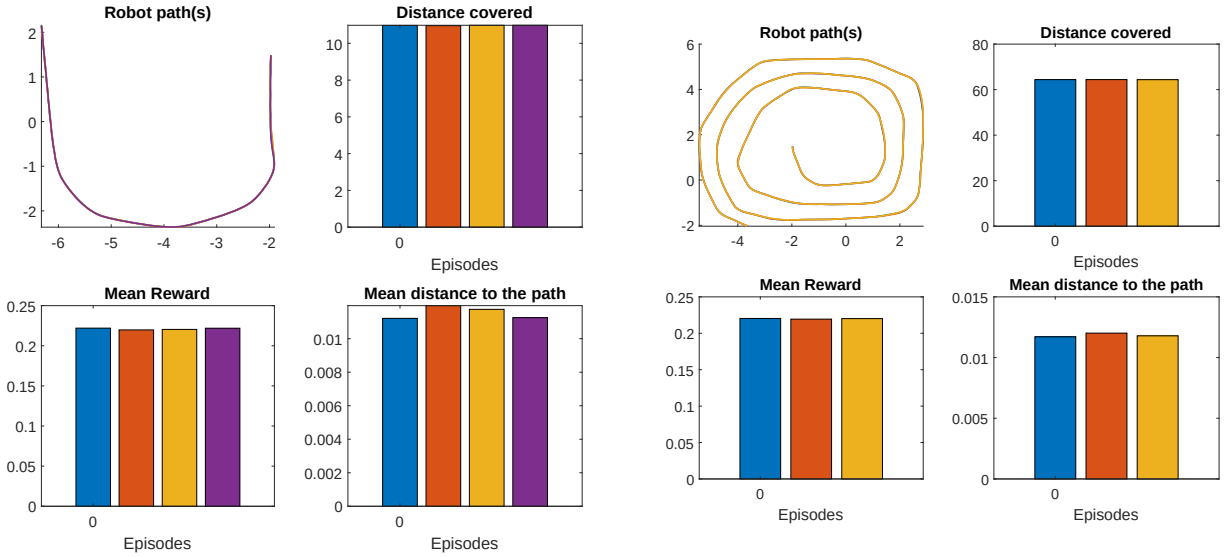
Figures 6.12, 6.13, and 6.14 presents the results obtained on the six paths, represented with cubic splines, using the previously generated Model 6.



(a) (b)

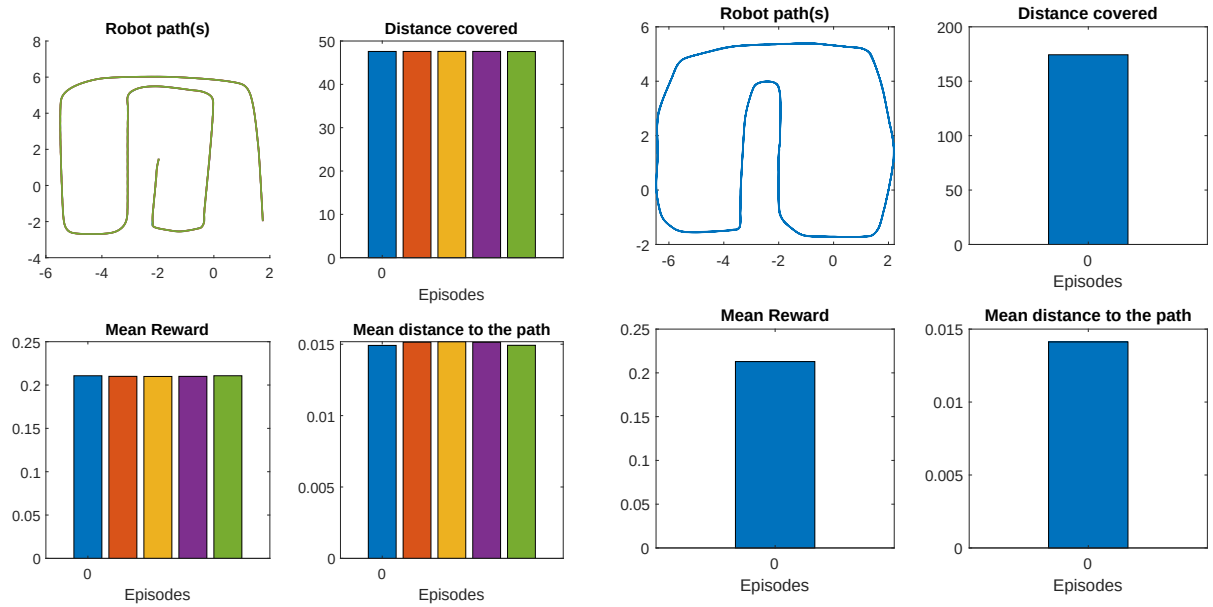
Figure 6.12: Results obtained using Model 6 on: (a) Path 1 and (b) Path 2.

Analyzing the Figs. 6.12, 6.13, and 6.14, it is possible to verify that using Model 6 the robot can complete all the paths. In general the results obtained are similar for paths defined by line segments or cubic splines. It is also expected that paths with curves with a high degree of curvature will cause an increase in training time.



(a) (b)

Figure 6.13: Results obtained using Model 6 on: (a) Path 3 and (b) Path 4.



(a) (b)

Figure 6.14: Results obtained using Model 6 on: (a) Path 5 and (b) Path 6.

Since the results obtained using Representation 1 were considered good, the method was also tested using Representation 2 for the six paths. As said before, the use of Representation 2 involves a greater number of states, which increases the overall complexity.

Figure 6.15 shows the results obtained when Representation 2 is used in paths defined by cubic splines. A new model was trained in Path 1 and then all the paths were tested. Instead of retraining the model when a state is unknown to the RL-model, the BSRA algorithm is used in the online stage to select a suitable action for each new state. In Fig. 6.15 the red circles represent iterations where the BSRA algorithm was activated. It is noticeable that in the more complex paths (such as Paths 5 and 6), the proposed approach allowed the robot to choose the best actions to continue to follow the path correctly. Using the BSRA algorithm as support, the robot is able to follow all the paths using the proposed RL approach.

After obtaining the results of the test in which the agent has at its disposal 162 possible actions, the Model 6 was analyzed. This analysis made it possible to verify the number of actions that were used.

For the states obtained, the count of all actions with positive Qvalue is shown in Fig. 6.16. However, considering only for each state the action with maximum Qvalue, it is presented in Fig. 6.17 which actions are chosen for the total of states.

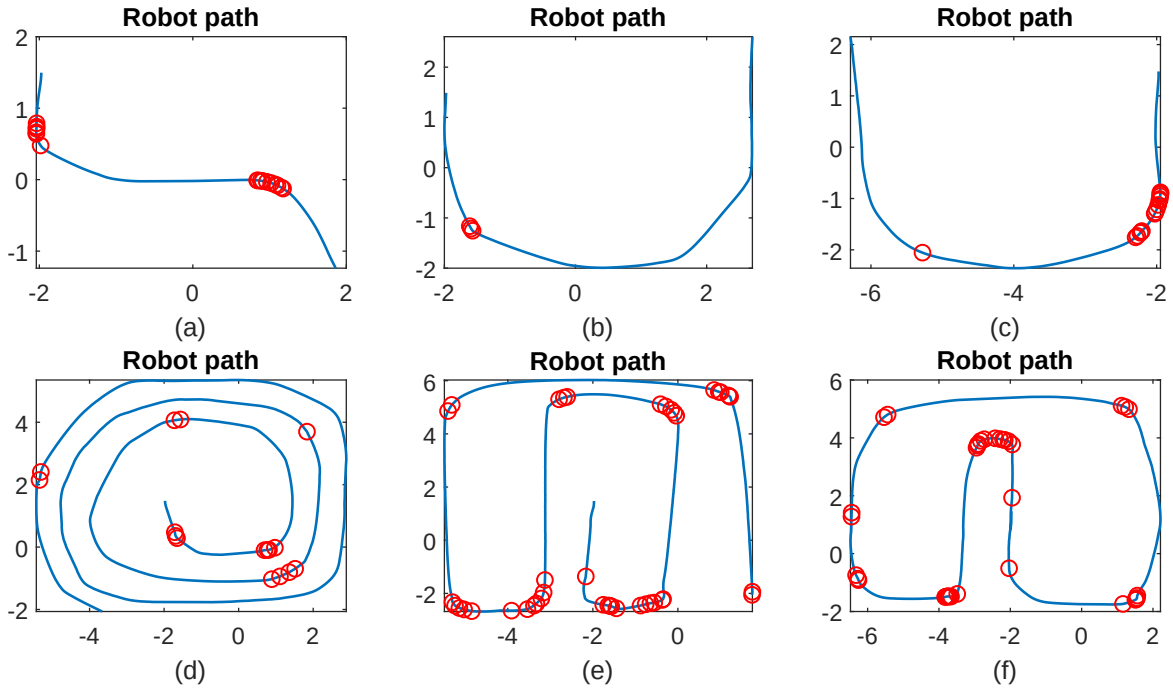


Figure 6.15: Results obtained using Representation 2 with the new model on all paths (defined by cubic splines). Red circles represent the iterations when the BSRA algorithm was activated. The values for the mean reward and mean distance to path are: (a) 0.2243 and 0.0103, (b) 0.2254 and 0.0100, (c) 0.2189 and 0.021, (d) 0.2252 and 0.0101, (e) 0.2013 and 0.0163, and (f) 0.2195 and 0.0130.

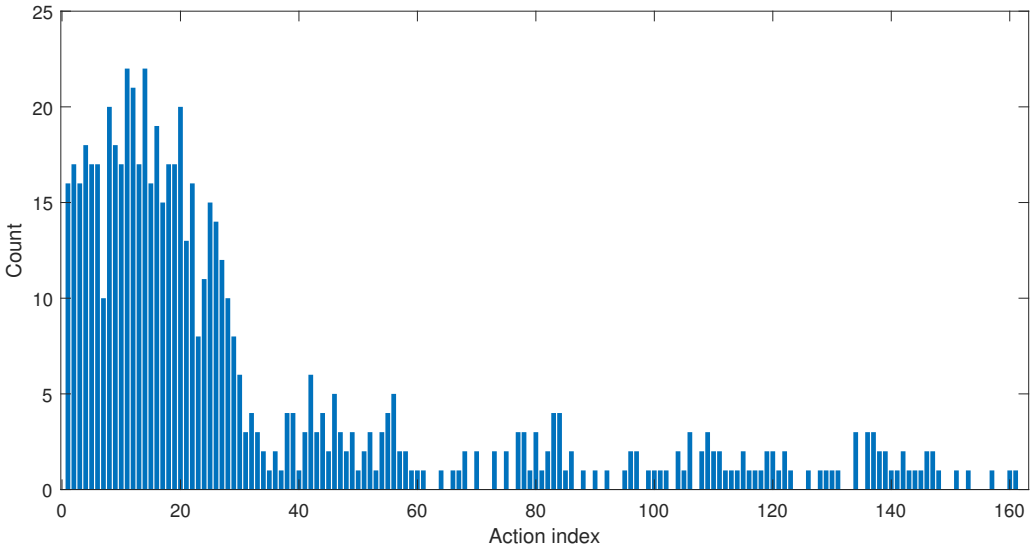


Figure 6.16: Bar graph with count of all actions with positive Qvalue.

Analyzing Fig. 6.16 it is notable that there is exploration of actions. On the other side, by observing Fig. 6.17, it can be concluded that the model does not use the 162 actions previously defined, but only 34, i.e. not all the actions are necessary.

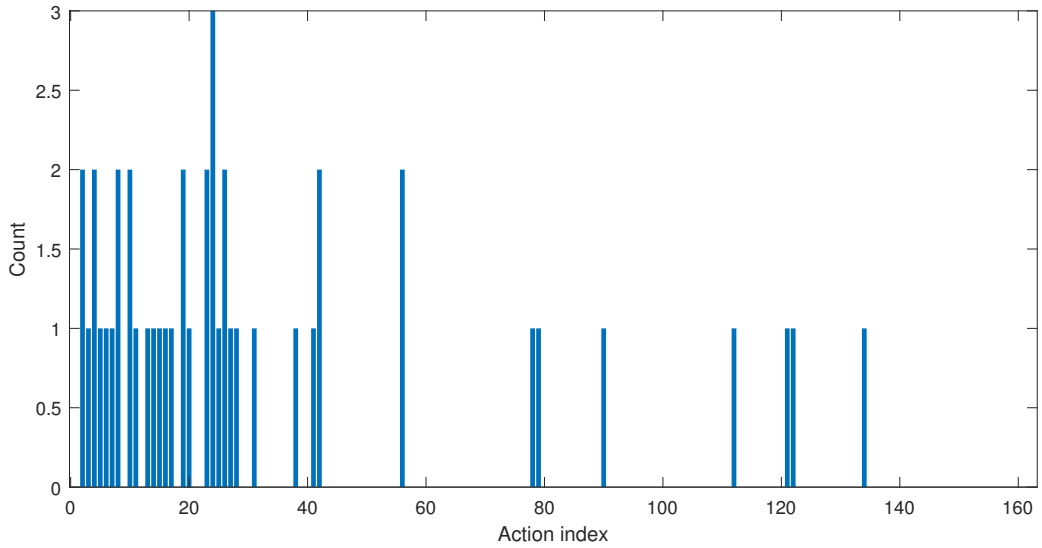
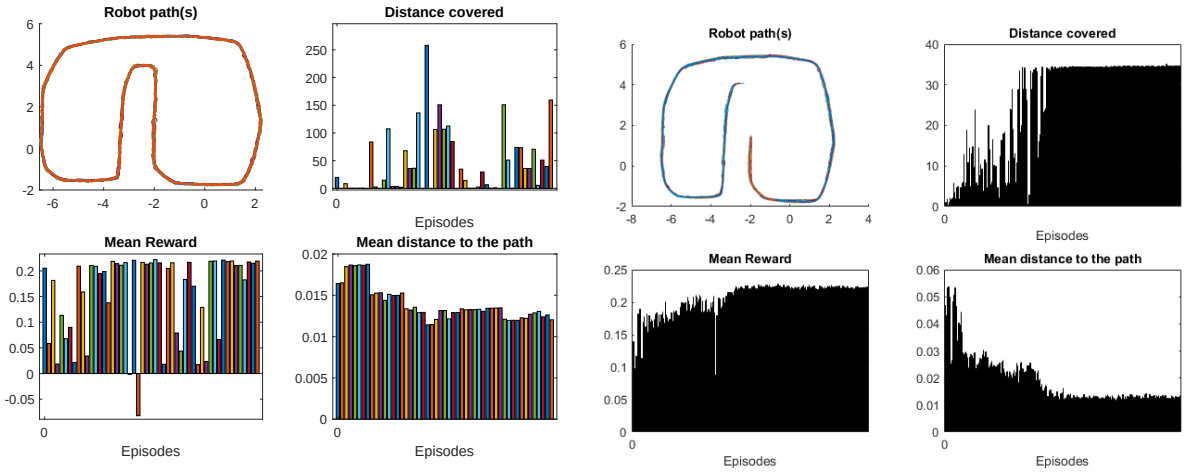


Figure 6.17: Bar graph with actions chosen for the total of states.

6.4 3 Actions vs 162 Actions

Figure 6.18 provides a comparison of how the method works, without the BSRA algorithm, when the agent uses only 3 actions or 162 actions. The path used in this test was Path 6. When the robot is limited to 3 actions, its behavior is highly unstable compared to 162 actions. However, despite the instability, the robot can complete the circuit. The same does not happen when it has 162 actions available. The reason why the robot is unable to complete the path is that there are a large number of possible actions and at some point, the robot will have to explore the best action of all the possible actions. It may happen that only after a significant amount of time the robot determines that action. The same thing surely happened while following this path, but the test was stopped before the robot could choose the action. Although the robot is unable to complete the path, it behaved well since the average reward value grows and stabilizes, and the average distance to the path decreases significantly as the number of episodes increases.



(a) (b)

Figure 6.18: Results obtained after training on Path 6 using (a) 3 actions and (b) 162 actions.

6.5 Path with near obstacles

Once the tests were done for scenarios whose paths were completely free of obstacles, it is possible to verify that the method developed presents promising results and goes according to expectations.

Therefore, we tested a scenario that presents obstacles near the path. By doing this, it was also possible to evaluate the robot behavior when confronted with close obstacles. In this test the agent has 162 actions available. The scenario used is presented in Fig. 6.19.

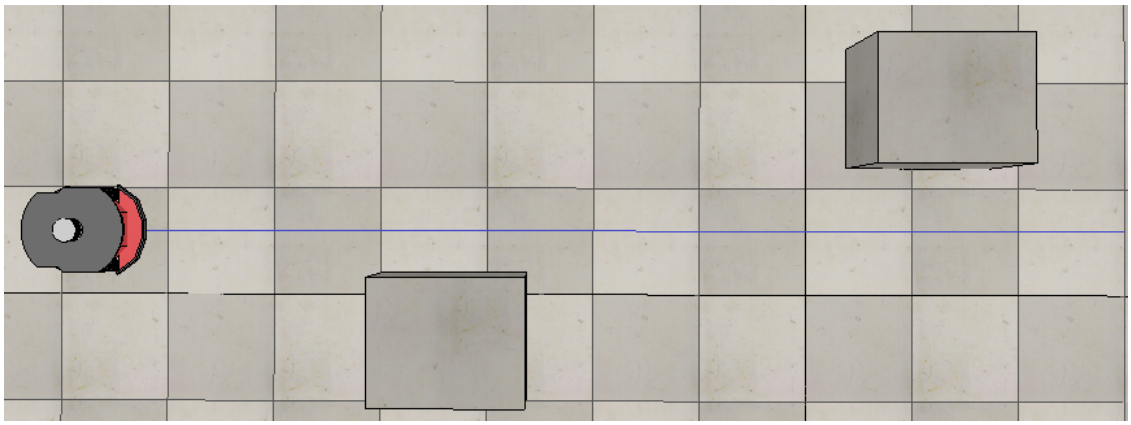


Figure 6.19: Scenario used to test the method for a path with nearby obstacles. The blue line represents the path and the blocks obstacles.

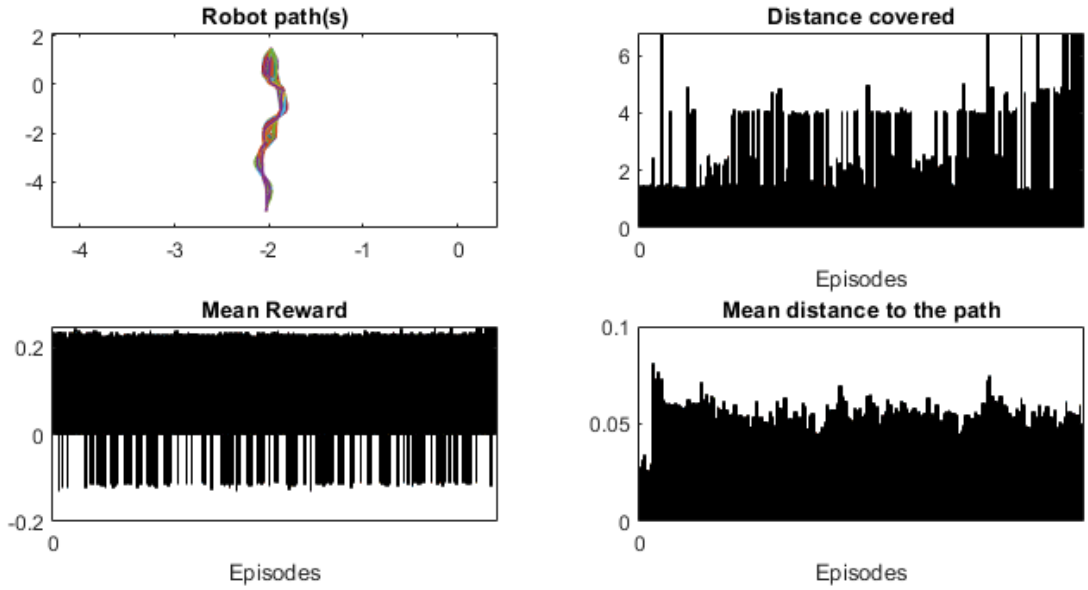


Figure 6.20: Results obtained after training on a scenario with nearby obstacles.

Figure 6.20 shows that the method developed can follow the path avoiding the obstacles. Initially the distance covered is short because the robot fails when it comes close to the obstacles. As the training continues this distance becomes longer since the robot is able to follow the path while avoiding obstacles. The mean distance to the path is variable because the robot has to avoid obstacles while following the path. It is important to mention that solving the problem of the existence of obstacles requires more than just computing the cost (Algorithm 1, line 16). Using only the cost evaluation it is not possible to follow the path, however with the introduction of the RL approach a solution can be reached.

6.6 Learning based on a user's driving behavior

For this test, a model using user inputs was created in the ISR Shared Experimental Area (ISRsea) room.

Figure 6.21: ISRsea room used to obtain a model using user's driving motion command.

The user inputs consisted of sending speed commands using a gamepad to move the InterBot platform along a circuit present on the ground. The model consists of a dataset that was recorded using a ROS tool (rosbag file, *i.e*, ".bag" file), which saves data from the sensors and speed comands sent by gamepad.

In an initial phase the user drove the InterBot platform according to the circuit present on the floor using the gamepad and a dataset was obtained. This dataset, when used directly as a model in the validation scenarios, it was not possible to do the learning due to the continuous nature of the control commands. One of the problems when using the gamepad was that some of the speeds sent by the gamepad were not present in the set of actions that the method uses. For this purpose it was necessary to adapt the gamepad, i.e, a button on the gamepad was set to adapt the speeds sent by the gamepad to the Interbot when pressed. This ensures that the method can interpret any action (speed) sent by the user during the control of InterBot. The resulting path after gamepad adaptation is shown in Fig. 6.22.

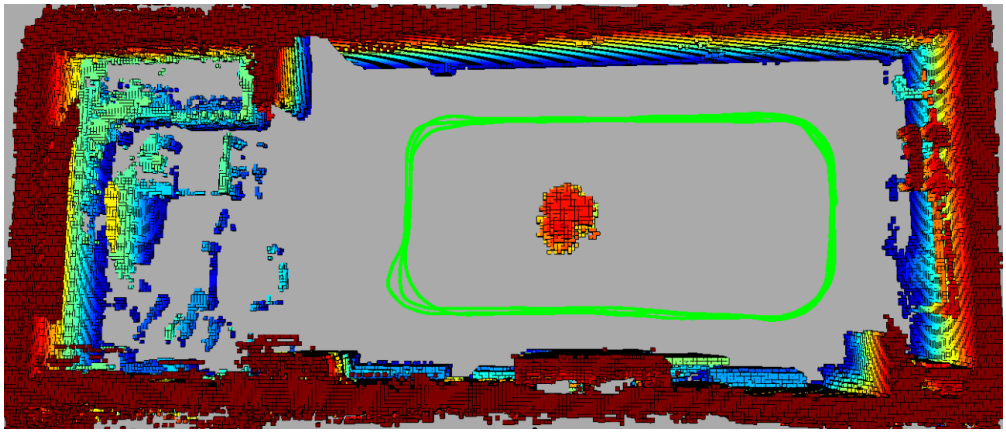
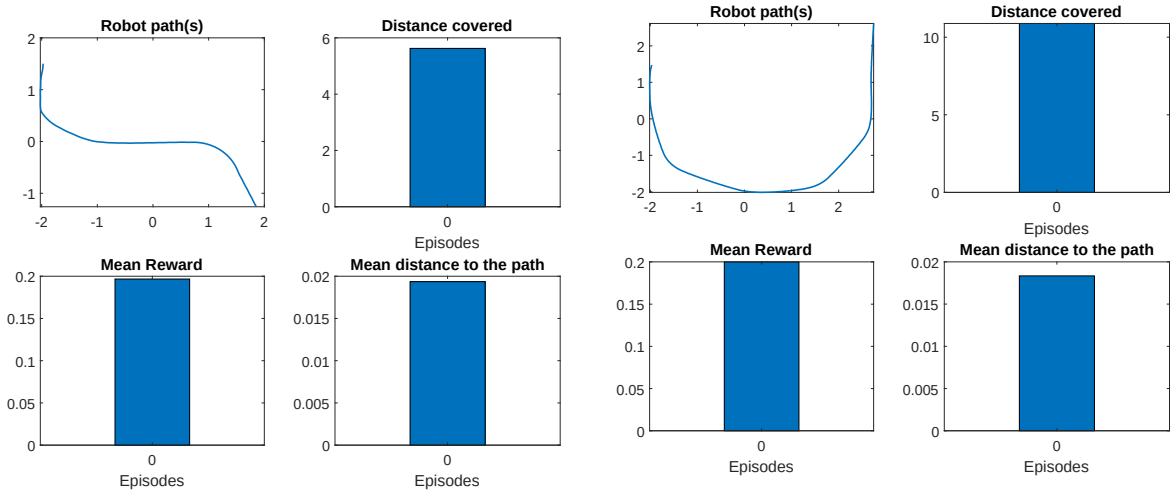


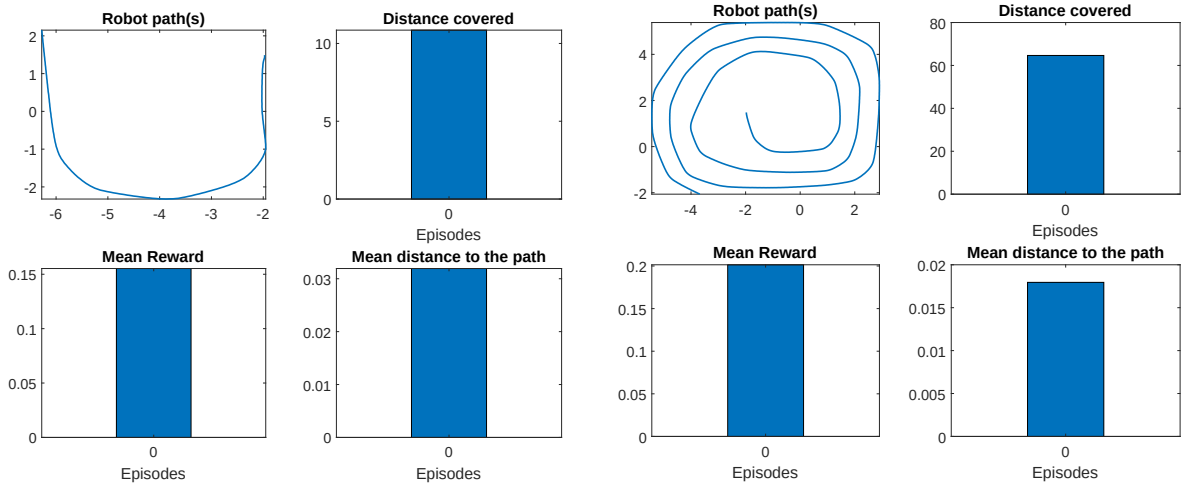
Figure 6.22: The environment representation was obtained from a voxel grid. The resulting path of the trajectory made by InterBot is represented in green. The other colors represent the detected objects, and there is an object in the middle of the path that concerns the human who was controlling the InterBot.

This test came from the necessity of further exploring the concept of training. In most of the tests previously performed, a model was generated in one path and used in the other paths. In the situations/paths that failed, a new model was generated, meaning that the training was always done in a known path (validation scenario). However, for this test, the model was generated using a user's motion commands and trajectory, and then tested in the validation scenarios. The results obtained are presented in the Figs. 6.23, 6.24 and 6.25.



(a) (b)

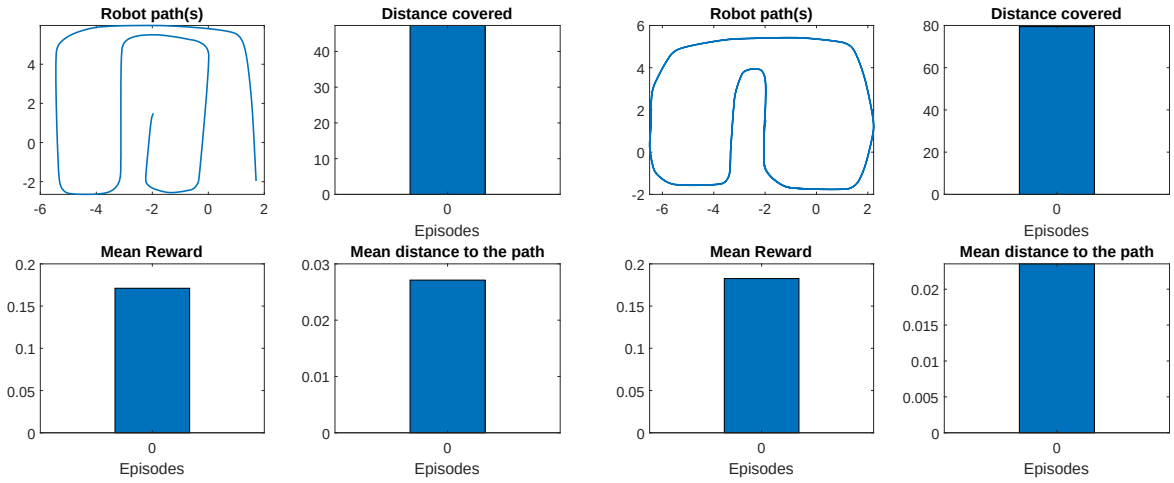
Figure 6.23: Results obtained using the model on: (a) Path 1 e (b) Path 2.



(a) (b)

Figure 6.24: Results obtained using the model in: (a) Path 3 e (b) Path 4.

Analyzing the figures it is possible to verify that the model generated by the user inputs (with the gamepad adapted) allows the robot, when placed in a validation scenario, to complete all the paths without presenting failures.



(a) (b)

Figure 6.25: Results obtained using the model in: (a) Path 5 e (b) Path 6.

6.7 ISR-UC Simulator

As a final test, the method was tested in a simulation environment developed at the ISR-UC [36]. This test is intended to validate that the method can be used in other simulators and that promising results can also be achieved.

Two scenarios were used as presented in Figs. 6.26 and 6.27. In the first one, the platform starts from the inside of the room and must leave the room through the door to go outside. In the second scenario, the same thing is done although the objective path, in the door area, is placed tangent to the left wall.

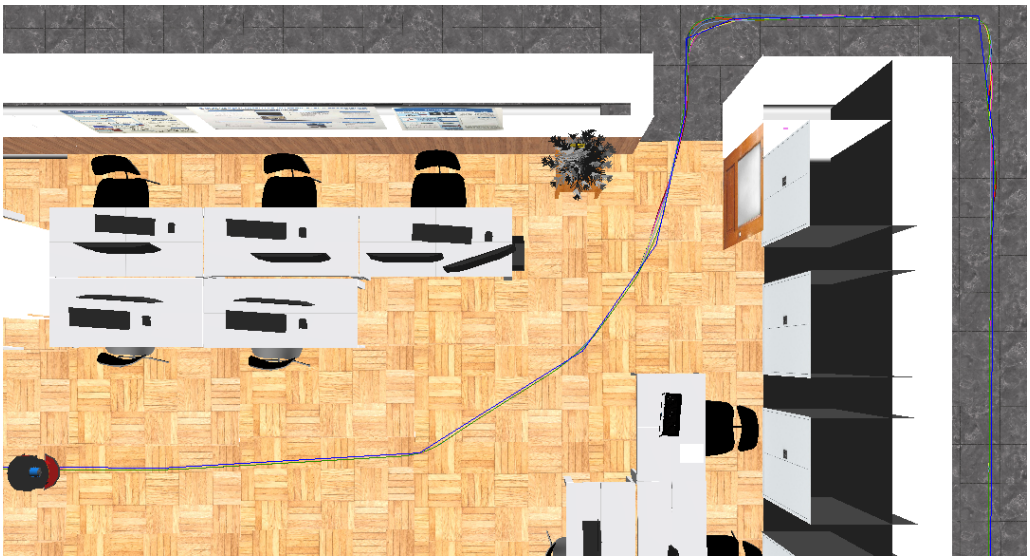


Figure 6.26: Result obtained on scenario 1.

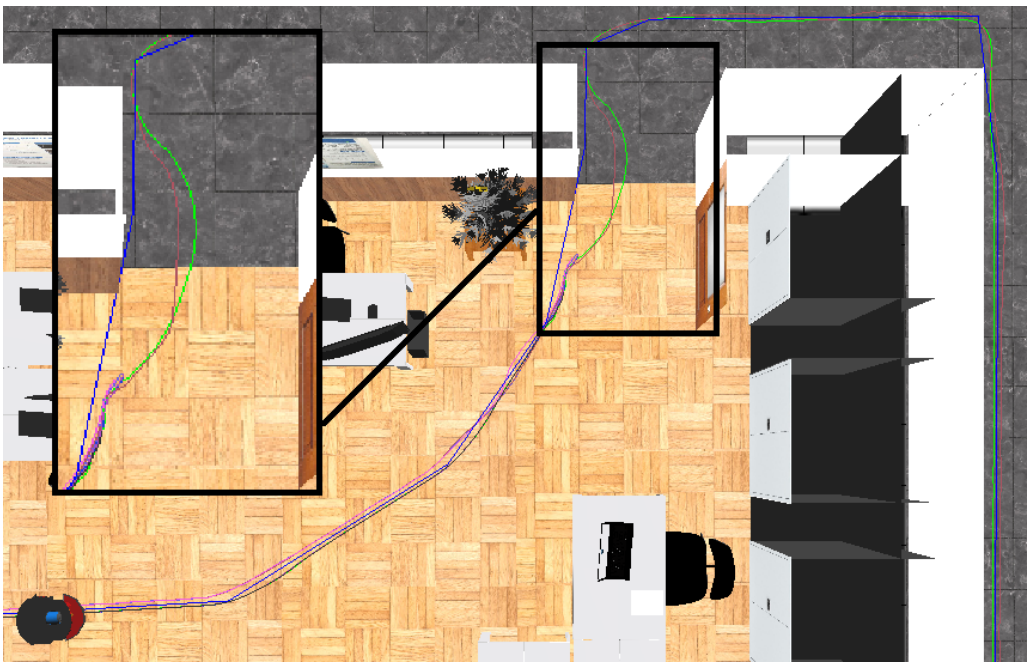


Figure 6.27: Result obtained on scenario 2.

Analyzing Figs. 6.26 and 6.27, it is possible to see that the robot is able to accomplish the proposed objective. It is important to observe that in the second scenario there is a reaction by the robot to avoid collision with the wall and thereby leave the room.

Chapter 7

Conclusion and future work

7.1 Conclusion

In this dissertation, most of the work focused mainly on the development of an RL-based navigation method. A study of different types of implementation of RL algorithms applied to navigation in mobile robots led to the discovery of interesting solutions for the problems that emerged resulting in a method that presents satisfactory results.

In this way, it is possible to deduce some important conclusions from the work accomplished in this dissertation.

As regard to the implemented representations for states using Representation 1 for any path is easier to train and has less complexity, the number of states associated with the path is equal to NxN. With Representation 2 it is more complex but when similarity techniques are used it is possible to obtain very similar results. Increasing the resolution of the grid implies an increase in the complexity of the learning stage (small adjustments of the robot position cause multiple different states).

Representing the paths with cubic splines or line segments to the RL turned out to be indifferent with the particularity of being faster to reach a solution with cubic splines (e.g., due to the curves with a curvature of $\approx 90^{o}$).

The number of actions has an impact on the type of behavior that the robot describes during the movement. With 3 actions it is easier to train but causes a very unstable behavior of the robot.

With the integration of obstacles in the scenario, although the method developed is very similar to a DWA approach (a set of speeds that are evaluated using a cost function), the method alone can not solve the problem with obstacles. To train an RL model with obstacles

to apply in a social or assistance context, where navigation should be precise, it will be necessary to iterate the method for thousands of episodes, with a costmap with smaller grids (e.g. 5cm).

The reward model implemented is simple but sufficient to be able to navigate. Due to the delayed reward problem it is impossible to learn from a simple RL model but the proposed solution can successfully learn.

Finally, dynamic information is missing from the RL state representation due to insufficient time, however, a solution as in [37] can be used. Using the state representation implemented in the developed method, it is possible to deal with dynamic information by representing areas with moving objects with a different code (e.g., empty - 0, occupied - 1, moving - 2).

7.2 Future work

To continue improving the current work several concepts have to be focused on, such as:

- **Dynamic Obstacles:** The robot should be able to handle dynamic obstacles while following a path.
- **Reward:** Add more elements to the reward to control for example the speed limits or accelerations of the robot.
- **Representation of states:** Transition from a 2D representation to a 3D model as a costmap representation.
- **Social behavior:** Use human behavior data to refine the RL model.

Bibliography

[1] NASA's Mars Exploration Program. Mars 2020. 2012. Available at `https://mars.nasa.gov/mars2020/mission/overview/`.

[2] R. Cruz, L. Garrote, A. Lopes, and U. J. Nunes. Modular software architecture for human-robot interaction applied to the InterBot mobile robot. In *2018 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, 2018.

[3] Ricardo Cruz. Interbot mobile robot: Human-robot interaction modules. Master's thesis, University of Coimbra, 2017.

[4] A. Conceicao. Interbot mobile robot: Navigation modules. Master's thesis, University of Coimbra, 2016.

[5] Jorge Perdigão Gabriel Pires Ana C Lopes, João Rodrigues and Urbano J. Nunes. A new hybrid motion planner applied in a brain-actuated robotic wheelchair. Master's thesis, University of Coimbra, 2016.

[6] Nicola Tomatis, Illah Nourbakhsh, and Roland Siegwart. Combining Topological and Metric: A Natural Integration for Simultaneous Localization and Map Building. *Proceedings of the Fourth European Workshop on Advanced Mobile Robots (Eurobot)*, 2001.

[7] Huijuan Wang, Yuan Yu, and Quanbo Yuan. Application of Dijkstra algorithm in robot path-planning. In *2011 Second International Conference on Mechanic Automation and Control Engineering*, 2011.

[8] Simon Wimmer and Peter Lammich. The Floyd-Warshall Algorithm for shortest paths. *Archive of Formal Proofs*, 2017.

[9] Harika Reddy. Path finding: Dijkstra's and A* Algorithm's. 2013.

[10] L. Zhang, H. Min, H. Wei, and H. Huang. Global path planning for mobile robot based on A* algorithm and genetic algorithm. In *2012 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2012.

[11] Amna Khan2 Iram Noreen1 and Zulfiqar Habib3. A comparison of RRT, RRT* and RRT*-Smart Path Planning algorithms. *International Journal of Computer Science and Network Security*, 2016.

[12] Rodriguez, Xinyu Tang, Jyh-Ming Lien, and N. M. Amato. An obstacle-based rapidly-exploring random tree. In *Proceedings 2006 IEEE International Conference on Robotics and Automation*, 2006.

[13] E. R. Firmansyah, S. U. Masruroh, and F. Fahrianto. Comparative analysis of A* and basic Theta* Algorithm in android-based pathfinding games. In *2016 6th International Conference on Information and Communication Technology for The Muslim World (ICT4M)*, 2016.

[14] D. Fox, W. Burgard, and S. Thrun. The Dynamic Window Approach to collision avoidance. *Robotics Automation Magazine, IEEE*, 1997.

[15] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to Deep Reinforcement Learning. *CoRR*, 2018.

[16] Christopher J. C. H. Watkins and Peter Dayan. Technical Note: Q -Learning. *Machine Learning*, 1992.

[17] Anton Schwartz. A reinforcement learning method for maximizing undiscounted rewards. In *ICML*, 1993.

[18] Prasad Tadepalli and Dokyeong Ok. H-learning: A Reinforcement Learning Method to Optimize Undiscounted Average Reward. 1996.

[19] J. Perdiz, L. Garrote, G. Pires, and U. J. Nunes. Measuring the impact of reinforcement learning on an electrooculography-only computer game. In *2018 IEEE 6th International Conference on Serious Games and Applications for Health (SeGAH)*, 2018.

[20] T. Tamei, T. Matsubara, A. Rai, and T. Shibata. Reinforcement learning of clothing assistance with a dual-arm robot. In *2011 11th IEEE-RAS International Conference on Humanoid Robots*, 2011.

[21] R. Cruz, L. Garrote, A. Lopes, and U. J. Nunes. Modular software architecture for human-robot interaction applied to the InterBot mobile robot. In *2018 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, 2018.

[22] B. Zuo, J. Chen, L. Wang, and Y. Wang. A Reinforcement Learning based robotic navigation system. In *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2014.

[23] Nahit EMANET Nihal ALTUNTAŞ, Erkan İMAL and Ceyda Nur ÖZTÜRK. Reinforcement learning-based mobile robot navigation. *Turkish Journal of Electrical Engineering & Computer Sciences*, 2014.

[24] Gedson Faria and Roseli A. Francelin Romero. Navegação de Robôs Móveis utilizando aprendizado por reforço e Lógica Fuzzy. *Sba:Controlo e Automação Sociedade Brasileira de Automática*, 2002.

[25] G. Yen and T. Hickey. Reinforcement Learning algorithms for robotic navigation in dynamic environments. In *Proceedings of the 2002 International Joint Conference on Neural Networks*, 2002.

[26] Michael Everett, Yu Fan Chen, and Jonathan P. How. Motion Planning Among Dynamic, Decision-Making Agents with Deep Reinforcement Learning. *CoRR*, 2018.

[27] Yu Fan Chen, Michael Everett, Miao Liu, and Jonathan P. How. Socially aware motion planning with Deep Reinforcement Learning. *CoRR*, 2018.

[28] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[29] Steffen Nissen. Large Scale Reinforcement Learning using Q-SARSA(λ) and Cascading Neural Networks. Master's thesis, University of CopenhagenDenmark, 2007.

[30] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 1989.

[31] David V. Lu, Dave Hershberger, and William Smart. Layered costmaps for context-sensitive navigation. *IEEE International Conference on Intelligent Robots and Systems*, 2014.

[32] M. Silva, L. Garrote, F. Moita, M. Martins, and U. Nunes. Autonomous electric vehicle: Steering and path-following control systems. In *2012 16th IEEE Mediterranean Electrotechnical Conference*, 2012.

[33] Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009.

[34] E. Rohmer, S. P. N. Singh, and M. Freese. V-rep: A versatile and scalable robot simulation framework. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013.

[35] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*, 2009.

[36] L. Garrote, J. Perdiz, G. Pires, and U. J. Nunes. Reinforcement Learning Motion Planning for an EOG-centered Robot Assisted Navigation in a Virtual Environment. In *2019 28th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, 2019.

[37] Q. Baig, M. Perrollaz, J. B. D. Nascimento, and C. Laugier. Using fast classification of static and dynamic environment for improving bayesian occupancy filter (BOF) and tracking. In *2012 12th International Conference on Control Automation Robotics Vision (ICARCV)*, 2012.