Alma Mater Studiorum · University of Bologna

**School of Science**
**Department of Physics and Astronomy**
**Master Degree in Physics**

# Navigation and human-robot interaction using reinforcement learning

**Supervisor:**
**Prof. Daniel Remondini**

**Co-supervisors:**
**Prof. Benoît Girard**
**Prof. Mehdi Khamassi**
**Dott. Nico Curti**
**Ph.D. Augustin Chartouny**

**Submitted by:**
**Keivan Amini**

Academic Year 2022/2023

# Acknowledgements

This section of the thesis delves into my more personal dimension.

It would have been much simpler if I had chosen to write the thesis in Bologna. However, as is often the case, I am drawn to challenges and demanding endeavors. Perhaps it's because I find it more rewarding to derive something meaningful from them, and that might just be true. I ventured into a new country, a new city, with a new language. I made new friends and adapted to new routines.
But in the end, every effort pays off. Every single time! My experience was truly amazing, encompassing diverse aspects. I balanced work and study while connecting with people. I had moments of enjoyment and acquired valuable lessons. I consider myself incredibly fortunate.

Every month I spent in Paris was accompanied by a great, unique common denominator: I had great people by my side. In my beloved laboratory, my friend and colleague Augustin accompanied me on this six-month journey, among the towers of the Sorbonne and meals at Crous. In Cité Universitaire, I was surrounded by carefree and strong friendships, and I want to highlight Alessandro and Marcello, who helped me during the toughest times and cooked pasta (cliché) when I needed it.

Augustin, you are an extremely intelligent person, and we both know that without you, this thesis would have achieved only half of the results. I appreciated many things about you: you gave me total freedom to decide what to focus on in the project, but at the same time, you were there, guiding me along the way. As I mentioned a few days ago to Mathilde, doing an internship under the guidance of a PhD student is like riding a scooter while all the other intern students are running on foot. Inevitably, you travel much faster, as evidenced by the fact that we presented our work at the IMOL conference. How lucky I have been to work with you!

Huge thanks goes to Benoît Girard and Mehdi Khamassi. You believed in me before anyone else, when I had no experience in the field, and I am eternally grateful for that. Benoît, you should know that during the first meetings for the ELSA Project I was thrilled: sitting at a table and squeezing the brain to discuss an interesting, multidisciplinary, and useful project was really what I was looking for. Augustin and Benoît: you taught me many things, both technically and as a human being, but what I should thank you for the most is that during a period when I was not convinced on my future career, you transmitted me enthusiasm for the world of research. Without saying a word about it, you guys convinced me to pursue a doctorate program. It will not be easy, but I am sure that in the end, I will thank you immensely once again.

The project involved in this thesis was carried out in Paris, but before this adventure, I spent a year and three months in the beautiful city of Bologna, once again surrounded by wonderful people. Lorenzo, Damiano, and Giulia were the people I bonded with the most, strolling around Piazza Aldrovandi with a spritz in hand or spending time together in the study room. Thank you so much for the great moments we shared, guys. I love you.

I am infinitely thankful to my family. You have always encouraged me to travel, and in the end, I have to admit that, although it was tiring, I have developed a taste for it. This year, I spent very little time at home, taking a total of 19 flights. I am sure you are proud of me, and you don't know how lucky I feel every day to have had the opportunity to grow alongside you. You have always pushed me to aim high, and slowly but surely, I know I am succeeding. I love you wonderfully.

Lastly, my dear Francesca. My desire to embark on new adventures has kept us apart and caused us to suffer, cry, and suffer again. But once I returned home, it was as if we had never been apart, and this time, distance did not keep us apart either. Francesca, you are a brilliant girl, perhaps even more so than me, and I am sure you have a bright, wonderful future ahead, full of achievements. I apologize for making you feel bad, you know that deep down I love you infinitely and that without you, I would be only half of half. Thank you from the bottom of my heart for the person you are and for your enduring patience with me over these past three years.

Thank you all so much.

This thesis was written in Paris, Washington, Zurich, and Thessaloniki. In the office, at home, at the beach. Until a few months ago, I only had a vague idea of what reinforcement learning was, and I discovered it to be a simple yet elegant theory with boundless levels of application. Every year of my life, I have always tried to raise the bar for my projects and future plans. I hope I have succeeded with this project, and I hope I have contributed, even in a small way, to doing something minimally useful for society.

Enjoy the read!

*19th September 2023,*

*Keivan Amini*

*"I believe there is no deep difference between what can be achieved by a biological brain and what can be achieved by a computer. It, therefore, follows that computers can, in theory, emulate human intelligence — and exceed it."*

*Stephen Hawking, Leverhulme Centre for the Future of Intelligence, Cambridge, U.K., October 2016*

## Abstract

Advances in artificial intelligence (AI) have revolutionized human-robot interaction and paved the way for increasingly sophisticated autonomous systems. Reinforcement learning (RL), a subfield of machine learning, has emerged as a powerful paradigm for training intelligent agents through trial and error. In the field of human-robot interaction, RL offers an unprecedented opportunity to give robots the ability to perceive and control social situations.

This thesis presents a simulation and an architecture designed to facilitate the learning of social affordances by an autonomous agent, enabling seamless interactions with humans. We propose a framework that combines RL algorithms with a comprehensive understanding of social affordances, with the primary goal to develop an agent capable of interacting with a human, grasping his attention and carrying him around specifics goal areas.

To evaluate the efficacy of our architecture, simulation trials were conducted using simulated humans with different characteristics, to assess the agent's ability to learn in different situations. In addition, an attempt was made to initialise a real-world experiment using Turtlebot robot, which can serve as a versatile platform for testing the agent acquired knowledge in a table scenario, focusing only in a navigation problem.

The simulation results clearly demonstrated the success of the designed learning architecture in integrating the knowledge acquired by the agent across different tasks, resulting in a great performance in terms of accumulated reward. Furthermore, in the experimental setup, the navigation problem tackled with the Turtlebot yielded excellent results.

This thesis contributes to the burgeoning field of human-robot interactions by leveraging RL techniques to enable robots to comprehend and engage with social affordances.

# Contents

**Introduction** **1**

**1 Reinforcement Learning** **6**

1.1 Markov Decision Processes . . . . . . . . . . . . . . . . . . . . . . . . 7

1.2 Dynamic Programming . . . . . . . . . . . . . . . . . . . . . . . . 10

1.2.1 Policy Iteration . . . . . . . . . . . . . . . . . . . . . . . . 12

1.2.2 Value Iteration . . . . . . . . . . . . . . . . . . . . . . . . 13

1.3 Action Selection . . . . . . . . . . . . . . . . . . . . . . . . 14

1.4 Temporal-Difference Learning . . . . . . . . . . . . . . . . . . . . . . . . 16

1.5 Model-Free and Model-Based methods . . . . . . . . . . . . . . . . . . 18

1.6 Tabular methods . . . . . . . . . . . . . . . . . . . . . . . . 20

1.7 Hierarchical Reinforcement learning . . . . . . . . . . . . . . . . . . 21

**2 The ELSA Project** **23**

2.1 Visit the lab scenario . . . . . . . . . . . . . . . . . . . . . . . . 25

2.1.1 Navigation module . . . . . . . . . . . . . . . . . . . . . . . . 27

2.1.2 Social module . . . . . . . . . . . . . . . . . . . . . . . . 31

2.1.3 Go to human option . . . . . . . . . . . . . . . . . . . . . . 36

**3 Materials and methods** **38**

3.1 Simulation . . . . . . . . . . . . . . . . . . . . . . . . 39

3.1.1 Learning architecture . . . . . . . . . . . . . . . . . . . . 39

3.2 Experiment . . . . . . . . . . . . . . . . . . . . . . . . 41

3.2.1 TurtleBot 3 Burger . . . . . . . . . . . . . . . . . . . . 42

3.2.2 Mapping . . . . . . . . . . . . . . . . . . . . . . . . 43

3.2.3 ROS . . . . . . . . . . . . . . . . . . . . . . . . 46

3.2.4 State machine . . . . . . . . . . . . . . . . . . . . . . . . . . 48
3.2.5 Experimental Transition Probability Matrix . . . . . . . . . . . 50

**4 Results** **54**
4.1 Navigation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 55
4.2 Go to Human . . . . . . . . . . . . . . . . . . . . . . . . . . . . 57
4.3 Social Interaction . . . . . . . . . . . . . . . . . . . . . . . . . . 59
4.4 Combined tasks . . . . . . . . . . . . . . . . . . . . . . . . . . . 61
4.4.1 Human adaptability . . . . . . . . . . . . . . . . . . . . . . . 63
4.5 Experiment . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 64

**5 Conclusions** **68**

**Appendices**

**A Manhattan Distance** **71**

**B LIDAR sensor** **73**

**C IMOL 2023 Poster** **76**

**Bibliography** **78**

# List of Figures

1 Photo of the ISIR pyramid in Jussieu Campus . . . . . . . . . . . . 1

1.1 The agent-environment interaction in a Markov Decision Process . . 7
1.2 Graphical representation of policy iteration process . . . . . . . . . . 12
1.3 Options over Markov Decision Processes . . . . . . . . . . . . . . . . . 22

2.1 Affordances schema in human-robot interaction . . . . . . . . . . . . 24
2.2 Table-top scenario in LAAS laboratory . . . . . . . . . . . . . . . . . . 25
2.3 Visit the lab environment . . . . . . . . . . . . . . . . . . . . . . . . . 26
2.4 Visualization of the Visit the lab scenario . . . . . . . . . . . . . . . 28
2.5 All 24 dynamical actions in the simulation . . . . . . . . . . . . . . . 29
2.6 Gridworld environment . . . . . . . . . . . . . . . . . . . . . . . . . . . 30
2.7 Visual field, facing field and distance . . . . . . . . . . . . . . . . . . 31

3.1 Developed learning architecture for combining navigation and interaction tasks. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 40
3.2 Experimental environment . . . . . . . . . . . . . . . . . . . . . . . . . 41
3.3 Turtlebot3 Burger . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 42
3.4 Voronoi diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 44
3.5 Experimental state decomposition . . . . . . . . . . . . . . . . . . . . 45
3.6 ROS framework architecture . . . . . . . . . . . . . . . . . . . . . . . . 46
3.7 Localization performances comparison . . . . . . . . . . . . . . . . . . 48
3.8 ROS schema of the experiment . . . . . . . . . . . . . . . . . . . . . . 49
3.9 Transitions in real scenario example . . . . . . . . . . . . . . . . . . . 51
3.10 Experimental goal areas . . . . . . . . . . . . . . . . . . . . . . . . . 52

4.1 Model-Based VS Mode-Free performance comparison on the Navigation task . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 55
4.2 Model-Based comparison . . . . . . . . . . . . . . . . . . . . . . . . . 56
4.3 Learned Q-values heat maps for the navigation task . . . . . . . . . 57
4.4 Variance comparison of the reward in the Go to Human task . . . . 58
4.5 Learning performance of $\epsilon$-greedy Model-Based in the Social Interaction task. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 59
4.6 Learning performance of the Social Interaction module with different humans. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 60
4.7 Study of the human adaptability with our learning architecture. . . . 63
4.8 Entropy of transition probabilities . . . . . . . . . . . . . . . . . . 64
4.9 Performance of Model-Based agent on the experimental scenario . . 65
4.10 Cumulative reward over time of the Turtlebot exploiting knowledge . 66
4.11 Photo of the Turtlebot over the heat-map . . . . . . . . . . . . . . . 67

A.1 Example of Manhattan distance and Euclidean distance . . . . . . . 71

B.1 Picture of LIDAR sensor . . . . . . . . . . . . . . . . . . . . . . . . 73
B.2 RPLIDAR A1 Working Schematic . . . . . . . . . . . . . . . . . . . 74
B.3 LIDAR measured distance in RViz . . . . . . . . . . . . . . . . . . . 75

# Introduction

The work proposed in this thesis project is the result of a six-month internship working on the ELSA Project, carried out at the research center Institut des Systèmes Intelligents et de Robotique (ISIR) in the city of Paris, under the Architectures and Models for Adaptation and Cognition team (AMAC), headed by the Research Director Benoît Girard.

The ISIR laboratory is under the dual supervision of Sorbonne Université, which is a world-class multidisciplinary university, and the Centre National de la Recherche Scientifique (CNRS).

**Figure 1:** *Photo of the ISIR pyramid in Jussieu, the main campus of the Faculty of Science of Sorbonne Université in Paris. Credit: (Universitè 2022)*

More specifically, the AMAC team is interested in developing models of perceptual, cognitive and motor functions, and in synthesizing control architectures from an integrated perspective. These research lines have a dual purpose: firstly the understanding of living organisms through the integration of mathematical and computer modeling approaches, and secondly the provision of robots with cognitive and motor skills that integrate decision making and learning. In this context, biology is used as inspiration for robotics development, and engineering proposes tools for modelling, experimenting and validating the considered hypotheses.

When providing robots with cognitive skills, reinforcement learning (RL) plays an important role. This machine learning paradigm focuses on how agents can learn continuous decision-making to maximize reward signals within an environment.

The history of reinforcement learning can be traced back to the 1950s, when researchers began exploring the idea of creating intelligent systems that could learn from interactions. One of the first contributors to RL was Arthur Samuel, who developed a computer program for learning to play checkers in 1959: this program used a process of trial and error to improve its performance, demonstrating the potential of machines to learn through self-play (Samuel 1959). Another significant milestone in RL came in the 1980s with the development of the Q-learning algorithm by Christopher Watkins (Watkins 1989). Q-learning introduced the concept of value functions, which assign the expected reward to each action in a given state; by iteratively updating these values based on the observed rewards, Q-learning enables agents to learn optimal strategies. The 1990s witnessed the rise of neuroscientific and psychological insights into RL: the work of Richard Sutton, and Andrew Barto on temporal difference learning bridged the gap between RL and neuroscience, also providing a framework for understanding how the brain processes reward signals and learns from experience (Sutton 1988). In recent years, RL has experienced significant advancements and breakthroughs, largely due to the development of deep learning techniques. Deep Q-Networks (DQNs) combines deep neural networks with Q-learning, enabling RL agents to achieve human-level performance in Atari games (Mnih et al. 2013). One of the most notable milestones in RL history came in 2016 when the software AlphaGo, developed by the AI laboratory DeepMind, defeated the world champion Go player, Lee Sedol. AlphaGo's success demonstrated the power of RL and deep learning in tackling complex, strategic games.

At the moment, RL has a wide range of applications across various fields:

- **Robotics**: allows robots to learn complex tasks that can be difficult to program explicitly. Moreover, by continuously learning and updating their policies based on feedback, robots can adjust their behavior and make decisions that are suitable for the current situation, so that it is possible to adapt to dynamic environments.

- **Self-driving cars**: by using RL algorithms, cars can learn optimal actions based on environmental states to maximize rewards, such as reaching the destination efficiently while adhering to traffic rules and ensuring passenger safety. This application is typically exploited in conjunction with other techniques and AI fields such as computer vision, deep learning, and classical control algorithms to create comprehensive self-driving systems (Kiran et al. 2021).

- **Finance**: it is possible to develop automated trading systems. RL agents can learn optimal trading strategies by analyzing historical market data and making decisions based on rewards or profits obtained. This enables the discovery of complex patterns and adaptive trading strategies that can also exploit market inefficiencies (Charpentier, Elie, and Remlinger 2021).

- **Healthcare**: RL can be used to optimize treatment plans for individual patients. By learning from patient data and clinical outcomes, RL agents can adaptively recommend personalized treatment options, dosage adjustments, and therapy scheduling to maximize patient outcomes and minimize side effects. These algorithms can be also exploited for clinical decision support (Yu et al. 2021).

However, it should be underlined that most of the works mentioned on the applications of RL in different fields are very recent, which means that this is still a very young field of study that has yet to achieve its full potential.

Within this thesis project, we will use RL algorithms to make a robot learn how to correctly interact with a human and guide it to a certain place within an environment. In particular, we will focus on the idea of social affordances - a concept that will be fully explained within Chapter 2. One of the goals of ELSA Project is to develop an agent able to construct both a general model of characteristics associated with humans as a species (such as the distance of social interaction or the importance of eye contact for proper interaction), and a specific model that characterizes individuals (such as variations in movement speed or differences in attention span and distractibility). To get this project started, we developed a simulation using the programming language `Python` exploiting the object-oriented programming paradigm and coding the RL algorithms from scratch, without using dedicated external libraries. In the RL problem addressed, a module was implemented associated with the navigation problem - reaching a certain location in space - and a module associated with the social interaction problem - interacting correctly with the human and leading it in a certain direction. In this context, an architecture capable of arbitrating between these two modules composing the decision system was developed. We proceeded to the experimental phase, utilizing a Turtlebot robot, where we focused on addressing the navigation problem. Employing our RL algorithms, we trained an agent to tackle a dynamic rewarded area problem, subsequently exploiting the acquired knowledge in a tabletop scenario.

Chapter 1 delves into the technical underpinnings of reinforcement learning and its practical applications. We will commence by exploring the foundational theory of Markov Decision Processes, followed by an examination of the mathematical optimization technique known as dynamic programming. Finally, we will provide a comprehensive explanation of the reinforcement learning algorithms employed in the project.

Chapter 2 centers on the ELSA Project, with a specific focus on the "Visit the lab" scenario. This section is dedicated to delving into the fundamental concept behind the thesis, providing a theoretical rationale for the simulation choices made.

Chapter 3 is dedicated to materials and methods, encompassing both the conducted simulations and the experimental procedures. Within this context, we delve into the constructed learning architecture pertaining to the simulation and provide comprehensive information regarding the experimental scenario, the robot used, and the ROS (Robot Operating System) architecture.

Chapter 4 is dedicated to showcasing the results obtained in both the simulation and experimental phases. Our primary focus is on presenting the learning curves of the developed modules for various models of human behavior in the context of social interaction. We conduct a comparative analysis of Model-Free and Model-Based performance and explore how the rewards vary with changes in episode settings. Additionally, we demonstrate the feasibility of training an agent in our simulation using a Markovian probability matrix extracted experimentally. Finally, we provide evidence that a physically mobile robot can effectively leverage the acquired knowledge, yielding results close to optimal performance.

Chapter 5 centers on providing concluding remarks for this thesis project. It serves as a comprehensive recapitulation of the project's objectives, summarizing the key highlights and emphasizing the central themes explored. Furthermore, it places a strong focus on outlining future directions for the project and potential extensions that can be pursued in the years to come.

Concluding the thesis are two small appendices concerning technical details that may be of interest to the more curious enthusiasts of mathematical and sensor aspects, but which are not necessary for a full understanding of the concepts, and a last appendice focusing on the poster exposed at IMOL conference 2023.

# Chapter 1

# Reinforcement Learning

Machine learning is a branch of artificial intelligence that involves developing algorithms and statistical models that enable computer systems to improve their performance on a specific task based on data input. It encloses different techniques such as supervised learning, unsupervised learning and reinforcement learning.

The supervised learning approach consists in predicting or estimating an output based on one or more inputs (James et al. 2013). The typical supervised framework is characterized by a quantitative or categorical outcome measurement that one wishes to predict, a set of features that will be the basis of the prediction and a training set of data, in which one observes the outcome and feature measurements for a set of objects. Using this data it is possible to build a statistical model which will enable one to predict the outcome for new unseen objects. Examples of such techniques are neural networks, decision trees, or support vector machines extensively used in fields including computer vision, natural language processing and speech recognition.

On the other side, in the unsupervised learning paradigm there is no outcome measure, and the goal is to describe the associations and patterns among a set of input measures. The most famous general techniques are clustering and dimensionality reduction: the first allows to group similar unlabelled data points together, while the second one reduces the number of features in the data, while retaining the relevant information. These techniques can be used before applying a supervised learning methods, or even in exploratory data analysis to gain insights into the underlying structure of the data.

The reinforcement learning approach differs from supervised and unsupervised learning due to its nature on focusing on goal-directed learning from interaction

(Sutton and Barto 2018). Generally, the learner is called agent and its primary objective is to determine the most effective actions to take in order to maximize a specific numerical signal that represents a form of reward. This learning process involves the agent interacting with its surrounding environment and receiving feedback in the form of rewards or penalties based on its chosen actions.

One of the most important aspects of this techniques relies on the exploration-exploitation trade-off. In order to gain a great deal of reward, an agent must favour actions that it has previously performed and determined to be successful in yielding reward. However, in order to uncover such actions, it needs to try the ones that it has not chosen before. The agent should take advantage of its existing knowledge to acquire reward, but also has to explore to make more informed action selections in the future.

Within this chapter, we will focus on the theoretical foundations of reinforcement learning, so that it will be possible to explore and comprehend what it means to learn from interactions.

## 1.1 Markov Decision Processes

Markov Decision Process (MDPs) are a traditional formalization of sequential decision-making, where actions have an impact on future rewards through those stages or situations as well as on immediate benefits. Indeed, MDPs provide a mathematical framework that captures the essential components of RL, including states, actions, rewards, and the dynamics of the environment.

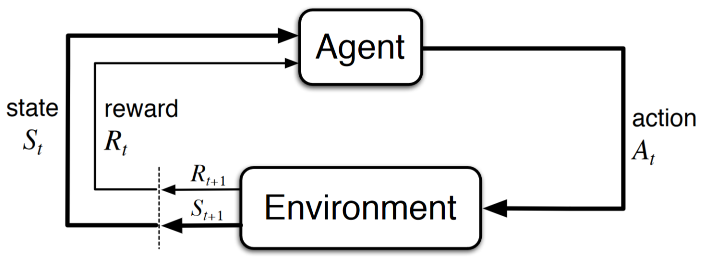


**Figure 1.1:** *The agent-environment interaction in a Markov Decision Process. Credit: (Sutton and Barto 2018).*

In Figure 1.1 is represented the general schema of a MDP in a RL problem, by considering a sequence of discrete time steps. An agent takes a certain action $a$

at time step $t$, described as $a_t$. The agent then receives a representation of the of the environment's state $s_{t+1}$, and a numerical reward $r_{t+1}$. We thus obtain an alternation of states, actions and rewards within our system. Usually these states, action and rewards are formally described as elements belonging to specifics mathematical sets, i.e., $s_t \in \mathcal{S}$, $a_t \in \mathcal{A}(s)$ and $r_t \in \mathcal{R} \subset \mathbb{R}$.

The MDP dynamics is formally described by a transition probability function:

$$p(s'|s,a) \equiv Pr\{s_t = s'|s_{t-1} = s, a_{t-1} = a\} \tag{1.1}$$

This function defines the probability of transitioning to a next state $s'$, given the old state $s$ and the action $a$. Mathematically, the conditional probability notation states that the current state depend only on the immediately preceding state and action, not on the earlier history: this important feature takes the name of *Markov property.* This type of mathematical framework boasts great generality and this guarantees flexibility to be applied in different scenarios. For example, states can represent the moods of a person, or the position of an object in a three-dimensional space. Actions, similarly, can be, for example, grasping an object, or deciding to send an application for an important company. Time steps are also a general concept: they do not need to be associated with the concept of real time, but can be seen as arbitrary successive stages of decision making.

When considering an MDP applied to an RL problem, it is usual to formally introduce the concept of time through time steps. In the presentation of this thesis project, the two key words to be used are step and trial, where a trial consists of a large number of steps. A trial can end when for example a certain number of steps have occurred, or trivially when our agent receives a positive reward. For example, let us consider an RL navigation problem in which our agent must learn to reach a certain destination in a well-defined scenario. Each time the agent performs an action and ends up in another state, a step has been completed. When, let us say, the first 50 steps have been performed, the first trial is over: when this happens, the initial conditions of the system are restored and so our agent starts from the initial position and can return to learning via trial and error. Clearly, the high number of trials and errors is essential to guarantee the learning of our agent.

It is important to emphasise that the aim of an agent is to find a strategy that allows it to maximise the total amount of reward it receives from the environment. This reward signal therefore formalises what our agent's goal is; it does not, however, define how our agent should behave in order to achieve it. For example, if we face a navigation problem and we want our agent to reach a geometric position of coordinates $(x, y)$, we need only to define a reward of $+1$ in the state $(x, y)$, and a reward of zero for each time the agent is in a state which coordinates differ from our goal.

Since one usually wants to maximise not the immediate reward, but the sum of future rewards from a given time step, it is usual to define the return:

$$G_t \equiv \sum_{i=1}^{T} r_{t+i} \tag{1.2}$$

where $T = T - t$ is the final time step of the task, that can be a random variable or a well-defined variable. Moreover, introducing the discount rate parameter $\gamma$ it is possible to define the discount return:

$$G_t \equiv \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{1.3}$$

where $0 \leq \gamma \leq 1$. This coefficient determines how much our agent will focus on the immediate rewards, or on those more distant in time. If $\gamma = 0$, the agent will focus only on the immediate reward and will therefore choose the action $a_t$ that maximises only $r_{t+1}$. At the other extreme, in the case $\gamma = 1$ there is no difference in worth between immediate rewards and more future rewards. Moreover, the definition 1.3 benefits of a useful recursive property which will be exploited later:

$$\begin{aligned} G_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots \\ &= r_{t+1} + \gamma G_{t+1} \end{aligned} \tag{1.4}$$

Another key concept in RL is the policy: a map which determines which action to perform in a given state, usually defined as $\pi(a|s)$. Clearly, the central idea is to find a policy, i.e. a behaviour, which maps for each state $s$ an action $a$ that can guarantee to collect the highest expected return value in a run. This particular policy is called an optimal policy, and it is denoted by $\pi_*$. Fortunately, techniques exist to derive optimal policies, which will be discussed in more detail in the section 1.2.

In learning algorithms, the so-called action-value function (or Q-value) for policy $\pi$ takes an important role:

$$Q_\pi(s, a) \equiv \mathbb{E}_\pi[G_t | s_t = s, a_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s, a_t = a\right] \tag{1.5}$$

this formula quantifies the expected discounted return starting from state $s$, taking action $a$, and thereafter following policy $\pi$. This expression can be estimated from experience: if an agent is following a policy $\pi$ and maintains an average of return that have followed a certain state-action couple, then these average will converge to $Q_\pi(s,a)$ if the number of times that the state-action couple has been visited approaches to infinity. In particular, in RL theory such methods are called Monte Carlo Methods since they estimate quantities by considering the average of many random samples of actual returns.

Action-value function can be useful to define which policies are better than others:

$$Q_\pi(s,a) \geq Q'_\pi(s,a) \; \forall \; (s,a) \in (\mathcal{S},\mathcal{A}) \Rightarrow \pi \geq \pi \tag{1.6}$$

which means that a policy $\pi$ is defined to be better than or equal another policy $\pi'$ if the expected return is greater than or equal to that of $\pi'$ for each state-action couple. Clearly, optimal policies $\pi_*$ have the same optimal action-values function $Q_*(s,a)$.

Having introduced the basic concepts underlying the formalisation of an RL problem, we can now address the ways and algorithms by which MDPs can be solved.

## 1.2 Dynamic Programming

Dynamic programming (DP) refers to a set of executable algorithms which aim is to compute optimal policies, starting from a perfect model of the environment as a MDP. This last requirement refers to having complete and accurate knowledge about the underlying system or process being modelled. Specifically, it means having a complete understanding of the MDP that describes the dynamics of the environment, with the knowledge of all the transition probabilities and rewards for each state-action pair. Unfortunately the requirement to have a perfect model of the environment makes classical DP use specific to only few situations; moreover, the processes requiring this type of technique are computationally expensive because of the curse of dimensionality, whereby the number of states often grows exponentially with the number of state variables.

The Bellman equations are fundamental equations in DP that provide a recursive definition of the action-value function, relating the value of a state-action pair to the values of its successor. In the case of the action-value function, it is derived by combining equation (1.5) and the recursive property definition's of the discounted reward (1.3):

$$\begin{aligned} Q_\pi(s,a) &= \mathbb{E}_\pi[G_t|s_t = s, a_t = a] \\ &= \mathbb{E}_\pi[r_{t+1} + \gamma G_{t+1}|s_t = s, a_t = a] \\ &= \sum_a \pi(a,s) \sum_{s,r'} p(s',r|s,a)\Big[r + \gamma\,\mathbb{E}_\pi[G_{t+1}|s_{t+1} = s', a_{t+1} = a']\Big] \end{aligned} \tag{1.7}$$

where the last expression sums over all values of the variables $a, s, r'$ and for each triple computes its probability $\pi(a,s)\sum_{s,r'} p(s',r|s,a)$ that weights the quantity in squares brackets. Moreover, if one wants to relate the optimal state-action function with its time step successor the so-called Bellman optimality equation can be exploited:

$$\begin{aligned} Q_*(s,a) &= \mathbb{E}_\pi\Big[r_{t+1} + \gamma \max_{a'} q_*(s',a') \;\Big|\; s_t = s, a_t = a\Big] \\ &= \sum_{s',r} p(s',r|s,a)\Big[r + \gamma \max_{a'} q_*(s',a')\Big] \end{aligned} \tag{1.8}$$

where $r$ is the immediate reward obtained in state $s$ and $\max_{a'} Q_*(s',a')$ represents the maximum value of the action-value function over all possible actions $a'$ in the successor state $s'$. The summation is taken over all possible next states $s'$ and rewards $r$ that can occur when taking action $a$ in state $s$, accounting for the probabilities $p(s',r|s,a)$ of transitioning to state $s'$ and receiving reward $r$ given the current state $s$ and action $a$. The equation states that the optimal action-value $Q_*(s,a)$ is equal to the expected return that can be obtained by taking action $a$ in state $s$, considering both the immediate reward $r$ and the maximum expected future rewards $\gamma \max_{a'} Q_*(s',a')$ for the next state $s'$.

Solving the Bellman optimality equation iteratively allows us to find the optimal action-value function $Q_*(s,a)$, which provides the maximum expected return for each state-action pair under the optimal policy.

In DP, two important concepts are policy evaluation and policy improvement. Policy evaluation refers to the process of determining the expected action-value for state-action pair under the given policy. This operation is typically done by iteratively applying the Bellman equation, and its is to estimate the quality of the current policy obtaining an accurate representation of the action-values. Once the policy has been evaluated and the action-value function has been obtained, policy improvement can be exploited. It involves updating the policy by selecting actions that are expected to yield higher action-values in each state, and this is done by greedily[1] selecting actions that maximize the value or action-value function. The policy improvement step aims to enhance the policy based on the information gained from policy evaluation, iteratively improving the decision-making process.

Policy iteration and value iteration are two common algorithms used to solve the Bellman optimality equations and find the optimal policy.

### 1.2.1 Policy Iteration

Policy iteration is an iterative algorithm that alternates between the two processes described above: policy evaluation and policy improvement. In the policy evaluation step, the algorithm computes the action-value function for a fixed policy by solving the Bellman equation. In the policy improvement step, the algorithm improves the policy by greedily selecting actions that maximize the action-value function. These steps are repeated until convergence to find the optimal policy.

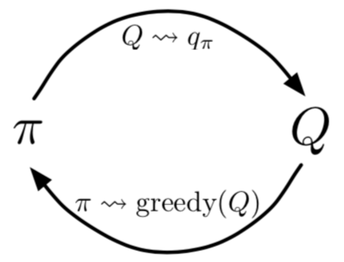

$$\begin{tikzcd}\pi \arrow[r, bend left, "Q \rightsquigarrow q_\pi"] & Q \arrow[l, bend left, "\pi \rightsquigarrow \text{greedy}(Q)"]\end{tikzcd}$$

**Figure 1.2:** *Policy iteration process. Given a starting policy $\pi$, we can evaluate it performing the policy evaluation, obtaining a certain action-value function $Q(s, a)$. Then we can run a policy improvement algorithm in order to obtain a better policy, e.g. $\pi'$; at this point it is possible to run policy evaluation again, and so on. Credit: (Sutton and Barto 2018).*

In Algorithm 1, one can see the pseudo-code of this DP method.

[1]The term "greedily" refers to a decision-making strategy where an agent chooses the action that maximizes the expected return in a given state, so that the agent selects the action with the highest predicted action-value among all available actions.

**Algorithm 1** Policy Iteration

```
Ensure: $Q(s,a)$ and $\pi(s)$ arbitrarily initialized
  Policy Evaluation
  while $\Delta < \theta$ do
      for $s \in \mathcal{S}, a \in \mathcal{A}$ do
          $q(s,a) \leftarrow Q(s,a)$
          $Q(s,a) \leftarrow \sum_{s',r} p(s',r|s,\pi(s))[r + \gamma \max_{a'} Q(s',a')]$
          $\Delta \leftarrow \max(\Delta, |q(s,a) - Q(s,a)|)$
  Policy Improvement
  $PolicyStable \leftarrow$ true
  for $s \in \mathcal{S}, a \in \mathcal{A}$ do
      $OldAction \leftarrow \pi(s)$
      $\pi(s) \leftarrow \text{argmax}_a \sum_{s',r} p(s',r|s,\pi(s))[r + \gamma \max_{a'} Q(s',a')]$
      if $OldAction \neq \pi(s)$ then $PolicyStable \leftarrow$ False
  if $PolicyStable$ then return $Q(s,a)$
  else Run Policy Evaluation
```

### 1.2.2 Value Iteration

Value iteration is a DP algorithm that directly computes the optimal action-value function by iteratively applying the Bellman optimality equation. In each iteration, the algorithm updates the values of state-action pairs based on the maximum expected return according to the Bellman equation. This process continues until convergence to obtain the optimal action-value function.

**Algorithm 2** Value Iteration

```
Ensure: small threshold $\theta > 0$ and $Q(s,a)$ arbitrarily initialized
  while $\Delta < \theta$ do
      for $s \in \mathcal{S}, a \in \mathcal{A}$ do
          $q(s,a) \leftarrow Q(s,a)$
          $Q(s,a) \leftarrow \sum_{s',r} p(s',r|s,a)[r + \gamma \max_{a'} Q(s',a')]$
          $\Delta \leftarrow \max(\Delta, |q(s,a) - Q(s,a)|)$
  if $\Delta \leq \theta$ then return $\text{argmax}_a \sum_{s',r} p(s',r|s,a)[r + \gamma \max_{a'} Q(s',a')]$
```

In each state update, value iteration combines one states update of policy evaluation and one states update of policy improvement: this algorithm is a special case of policy iteration when the policy evaluation process is stopped after just one update of each state, and this is useful since in some cases the full policy evaluation process can be computationally expensive.

Both policy iteration and value iteration have been correctly implemented in the

RL problem presented in this thesis work. However, it is relevant to underline that DP algorithms do not involve learning from data. They rely on explicitly solving subproblems and storing their results to build up to the optimal solution; in contrast, RL algorithms learn from interaction with an environment. They utilize trial-and-error learning, exploring different actions and observing the rewards obtained to update their knowledge and improve decision-making over time.

## 1.3 Action Selection

In reinforcement learning theory, action selection is pivotal as it dictates how an agent engages with its surroundings. It involves a delicate balance between exploration, where the agent experiments with various actions to uncover optimal strategies, and exploitation, where it utilizes existing knowledge to maximize immediate rewards. The goal of action selection in RL is to find a balance between exploring unknown states and exploiting the agent's current knowledge to make the most advantageous decisions. The agent needs to navigate the environment by choosing actions that lead to favorable outcomes and avoid actions that result in penalties or undesirable consequences.

The choice of action selection strategy greatly influences an agent's learning efficiency and its ability to converge towards optimal policies. There are various techniques and algorithms designed to address the exploration-exploitation dilemma and guide the agent towards maximizing its long-term rewards. In this context, action selection strategies can be broadly classified into two categories: deterministic and stochastic. Deterministic strategies involve selecting actions based on a fixed deterministic policy that maps states to actions. In contrast, stochastic strategies introduce randomness and probabilistic decision-making, allowing for a balance between exploration and exploitation.

Let us illustrate some simple and famous methods to select a specific action $a$ from a set of actions $\mathcal{A}$. One method that allows a good balance between exploitation and exploration is the so-called $\epsilon$-greedy method:

1. Initialize the parameter $\epsilon \in [0, 1]$, which represents the exploration rate. A small epsilon value means less exploration, while a large epsilon value means more exploration.

2. At each decision-making step, generate a random number $x \in [0, 1]$.
3. If $x < \epsilon$, then our agent will select randomly an action $a$ belonging to the set $\mathcal{A}$. Otherwise, if $x > \epsilon$, select the action with the highest estimated action-value based on the agent's current knowledge. Thus, select $\mathrm{argmax}_a\, Q(s, a)$.

In the literature, it has been shown how this type of stochasticity introduced by the variable $\epsilon$ helps to increase the cumulative reward collected by the agent (Sutton 1988). In fact, in the case where our agent always chooses the action associated with the highest expected reward, the possibility exists that our agent has not explored the system enough, and thus has not had the opportunity to explore other states that would perhaps lead to a higher reward!

Another popular method of choosing an action takes the name of Softmax exploration (or Boltzmann exploration), that assigns probabilities to each action based on their estimated action-value:

1. We consider the Q-value associated with the current state $s$, i.e., we end up with the estimated expected rewards for each action in the state. Then we apply the softmax function to transform the Q-values into a probability distribution:
$$P(a|s) = \frac{e^{\frac{Q(s,a)}{\tau}}}{\sum_{a'} e^{\frac{Q(s,a')}{\tau}}} \tag{1.9}$$
In literature, he $\tau$ parameter is also referred as temperature, and controls the level of exploration: higher temperature values increase exploration, while lower values promote exploitation.
2. Then it is possible to select an action based on the computed probabilities: e.g, we can can sample from the softmax distribution to select an action stochastically. Clearly, the higher the softmax probability of an action, the more likely it is to be chosen.

Depending on the formalism used, in the literature one may also find the use of variable $\beta = 1/\tau$.

To distinguish between the two methods, it's important to note that the softmax method allows for action selection based on their Q-values, resulting in a smoother exploration approach compared to $\epsilon$-greedy. With softmax, the emphasis is on exploring actions with higher action-values, whereas epsilon-greedy involves random exploration when not being greedy and doesn't take into account the information available on Q-values other than the maximum one.

It is important to emphasise how these methods for selecting decisions in a Markovian process are very useful in cases where we have to make our agent learn, precisely because exploration must be encouraged in cases such as these. But in the case our agent has already learnt, we would have already obtained an optimal Q-table. In this specific case, our agent has no need to explore the system at all: he only needs to exploit the knowledge, and therefore to be able to properly choose the action that allows for the greatest expected reward value along the way we simply select the action that maximises the Q-value in that state, i.e. $\text{argmax}_a$. So acting greedily.

## 1.4 Temporal-Difference Learning

Temporal Difference (TD) learning is a key technique in RL that enables agents to learn value functions through online, incremental updates based on the observed rewards and state transitions. TD learning combines elements of DP with online learning. As mentioned in the section before, DP methods requires a model of the environment dynamics and performs iterative updates based on the Bellman equation to estimate action-value functions. In contrast, TD learning allows agents to learn directly from experience without requiring the complete knowledge of the environment's dynamics.

The core idea behind TD learning is to use bootstrapping[2], where an agent updates its action-value estimates based on the observed reward and the estimated action-value of the next state. By iteratively updating the action-value estimates based on the observed differences between predictions and actual rewards, TD methods gradually refine their action-value functions.

The simplest example of a TD algorithm involves updates of the state-action value in the following manner:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\Big[r + \gamma Q(s',a') - Q(s,a)\Big] \tag{1.10}$$

[2]In this context, bootstrapping refers to the process of estimating action-value functions or making predictions about future outcomes based on limited available information. It involves using existing estimates or predictions to update and improve those estimates incrementally.

where it has been introduced the learning rate parameter $\alpha \in [0, 1)$ that determines the weight given to the new information obtained from the update. In this update, $r$ represents the immediate reward obtained by performing action $a$ in state $s$, while $Q(s', a')$ is the action-value estimate of the next state-action pair. The expression in square brackets is often defined as TD error, since it represents the difference between the observed reward and the estimated action-value of the next state-action pair:

$$\delta \equiv r + \gamma Q(s', a') - Q(s, a) \tag{1.11}$$

and since this error depends on the state, action and reward associated with time step $t + 1$, this quantity becomes accessible only at this time step. Which means that at each time step it is only possible to compute the error associated with the previous time step.

TD learning is a versatile and widely studied approach, leading to the development of various algorithms with different characteristics and applications. Within this thesis project, several such algorithms have been implemented; as an example, we propose the algorithm named Q-learning (Watkins 1989), which update rule for the action-value function is defined by:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\Big[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\Big] \tag{1.12}$$

where the indices concerning the time step have been introduced so that there is no confusion with the $\max_a Q(s_{t+1}, a)$ value, which is simply the state-action value referring to the next state of the system and the specific action that maximises that value.

**Algorithm 3** Q-learning

```
Ensure: step size $\alpha \in (0, 1]$ and $Q(s, a)$ arbitrarily initialized
  for steps in trials do
      Choose an action A from state S with an arbitrary method
      Take action A, observe reward R and state S'
      $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
      $S \leftarrow S'$
```

It is important to make a distinction between so-called on-policy and off-policy learning methods. The former learn and improve the action-value function on the basis of data collected from the current policy, i.e. the data used for learning comes from the same policy that is being updated. Off-policy methods, on the other hand, learn and improve the action-value function using data collected from a different policy than the one being updated. In off-policy methods, there is a clear separation between the behaviour policy, which is responsible for generating the data, and the target policy, which is the policy being learned and improved. In this scenario, Q-learning algorithm is an off-policy learning method because it can learn the optimal policy $Q_\pi$ regardless of the policy being followed during the learning process.

In a stationary environment, and in sufficient exploration conditions, the continuously updated value $Q(S, A)$ has been shown to converge with probability 1 to the optimal action-value function $Q_\pi$ (Sutton and Barto 2018): its simplicity and wide applicability have solidified its status as a famous and influential algorithm in the field of RL.

TD learning algorithms have several advantages. First, they enable online learning, where an agent can update its action-value estimates in real-time as it interacts with the environment, and this makes TD methods particularly useful in domains with continuous or infinite state spaces. Second, TD learning is Model-Free, meaning it does not require knowledge of the underlying environment dynamics: agents can learn directly from experience, making them adaptable to a wide range of RL problems.

## 1.5 Model-Free and Model-Based methods

Up to now, we focused on Model-Free agents, i.e. agents that learn by directly interacting with the environment, without having any explicit knowledge of their dynamics: these agents rely on trial-and-error learning, where they repeatedly take actions, observe the consequences, and update their policies based on the observed rewards. Since these kind of agents do not require knowledge about the environment's dynamics, they are more straightforward to implement and deploy. However, Model-Free agents often require a large number of interactions with the environment to learn an optimal policy effectively, making the learning process slower when the environment is composed of a large number of states.

In RL theory, there is another important category of agents called Model-Based algorithms, which approach the learning process differently. They focus on building an explicit model of the environment's dynamics: this model represents the agent's understanding of how the environment transitions from one state to another based on the actions taken, and the reward associated with that. With this model, the agent can perform simulations or predictions to make informed decisions about which actions to take. Usually, the computational cost associated with these algorithms is larger with respect to the Model-Free ones, but for the same number of steps and trials Model-Based agents tend to perform better in terms of rewards achieved.

Let us see an example of a Model-Based algorithm by considering the dynamic programming method value iteration (Algorithm 2) combined with the Softmax action selection method.

**Algorithm 4** Model-Based Value Iteration Softmax

```
Ensure: Softmax parameter $\tau \in (0,1)$, $Q(s,a)$ and Model$(s,a)$ arbitrarily initialized
  for steps in trials do
      $S \leftarrow$ Current state
      $A \leftarrow$ Softmax $(S, Q; \tau)$
      Take action $A$; observe resultant reward $R$, and state, $S'$
      Model$(S, A) \leftarrow R, S'$ (assuming deterministic environment)
      Run Value Iteration algorithm to update $Q(s,a)$
      $S \leftarrow S'$
```

In the algorithm exposed, Model$(s,a)$ denotes the contents of the model (predicted next state and reward) for state–action pair $(s,a)$. This is what characterises a Model-Based algorithm: that is, the construction of a model of the system, which is updated and on which planning can be carried out to make informed and strategic decisions. Note that in this case, the model of the environment is trivial to construct since we have silently incorporated the assumption of having a purely deterministic environment. But in experimental scenarios, this assumption does not tend to hold and in these cases, the construction of the model is slightly more complicated: let us now look how to deal with this situation, which formalism will be applied later in the Chapter 3. As for the transitions model of the system, this can be deduced storing the number of times each $(s,a,s')$ triplet has been encountered by the agent, then dividing by the number of times $(s,a)$ experienced, as shown in the equation below:

$$T(s,a,s') = \frac{n(s,a,s')}{n(s,a)} \tag{1.13}$$

where $n(s,a)$ stands for the number of visits of state $s$ when action $a$ is then chosen and $n(s,a,s')$ is the number of transitions from state $s$ to state $s'$, having performed action $a$. Clearly, the divisor in the equation is useful for normalising on the $s'$ arrival states of the system, so that the relation $\sum_{s'} T(s,a,s') = 1$ is valid (Massi et al. 2022).

Similarly, for the reward model of the system:

$$R(s,a) = \frac{\sum_{i=1}^{n} r_i(s,a)}{n(s,a)} \tag{1.14}$$

where the reward function $R(s,a)$ represents the average reward signal experienced when effectively performing the $(s,a)$ transition, while the numerator in the definition stands for the sum of the rewards obtained each time the agent made the transition $(s,a)$.

In Chapter 4, some results will be presented to help understand the performance differences between Model-Free agents and Model-Based agents.

## 1.6 Tabular methods

The algorithms and methodologies that have been expounded so far belong to a specific class of algorithms called tabular RL methods, i.e. methods that maintain a table or matrix that explicitly stores the value estimates for each state-action pair in the agent's environment. Clearly, these values in the table based on the agent's experiences (rewards and observations) are updated during interactions with the environment.

Tabular RL methods are most useful in scenarios where the state and action spaces are small enough to allow efficient storage and computation. These methods excel in simple environments with discrete states and actions, where the agent can easily explore and update the value estimates for each state-action pair individually. However, there are some systems in which these assumptions are not met. Here tabular algorithms can encounter several problems, such as:

- **Continuous State Space**: tabular methods are not well-suited for continuous state spaces because they require discrete states to store values in the lookup table. In continuous spaces, the number of possible states becomes infinite, making it impractical or impossible to store and update values for all possible states.

- **Discretization**: one approach to using tabular methods in continuous state spaces is to discretize the state space, dividing it into a finite number of bins or cells. However, this process often leads to a trade-off between accuracy and the size of the discretized table. Fine-grained discretization improves accuracy but increases the computational and memory requirements exponentially.

- **Continuous Time Steps**: Tabular methods typically assume discrete time steps for updating the value estimates. Adapting tabular methods to continuous time steps requires additional modifications, such as interpolation that can help approximate the value function between the explicitly stored time steps.

These problems are solved by another class of algorithms: the so-called function approximations methods, which have not been examined within this thesis project, but will take importance in the continuation of the work associated with the ELSA Project (Chapter 2).

These methods aim to approximate an unknown function that maps inputs (usually state-action pairs) to outputs (value estimates) based on observed data. The key idea behind function approximation methods is to use a flexible model, such as a neural network, decision trees, linear regression, or other supervised machine learning models, to approximate the underlying function. These models have adjustable parameters that are learned from training data, typically obtained through interactions with the environment. The training process involves updating the parameters of the function approximator to minimize the difference (error) between the approximated values and the true values, which are obtained from observed rewards or returns during the agent's interaction with the environment. This optimization process is often done using various learning algorithms, such as gradient descent or stochastic gradient descent, which adjust the model's parameters in the direction that reduces the prediction error.

## 1.7 Hierarchical Reinforcement learning

This chapter of theoretical overviews concludes with a brief mention of the theory of Hierarchical Reinforcement Learning (HRL), which general ideas were exploited in the implementation of the learning architecture proposed in this thesis project.

In HRL, agents operate at multiple levels of abstraction (Barto and Mahadevan 2003). Instead of dealing solely with primitive actions, they can employ the so-called *options*, namely temporally extended actions that allow agents to execute sequences of actions as a single higher-level action (Sutton, Precup, and Singh 1999). These options enable agents to learn reusable skills and strategies, simplifying the acquisition of complex behaviors and offering a framework for handling intricate tasks more efficiently.

Central to the effectiveness of options in HRL are Semi-Markov Decision Processes (SMDPs). Unlike the traditional MDPs that assume fixed time steps, SMDPs introduce a flexible temporal framework since the duration of actions or events can vary, making them an ideal foundation for modeling options, which often have variable lengths: this temporal flexibility enhances the capacity of agents to adapt to diverse environments and tasks.

Without going into technical details, SMDPs are a generalisation of MDPs in which the actions can take a variable amount of time to complete, i.e. time can be formalized by adopting random variables denoting the number of time steps that action $a$ takes when it is executed in state $s$ (Dietterich 2000).

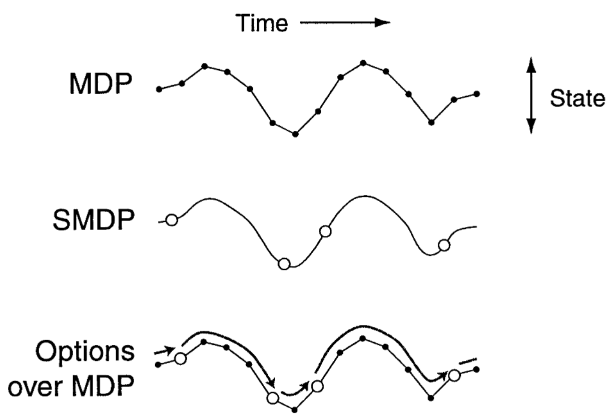


**Figure 1.3:** *Comparison between small discrete-time transitions of MDPs, larger continuous-time transitions of SMDPs and options over MDPs. Credit: (Sutton, Precup, and Singh 1999).*

Within this thesis project, we employed this formalism as a convenient method for encapsulating a sequence of actions into a single, multi-step action. This technique will be fully explained in Subsection 2.1.3, in particular into the so-called *Go to human* module.

# Chapter 2

# The ELSA Project

The ELSA project is a franco-austrian project in collaboration with the Institut des Systèmes Intelligents et de Robotique (ISIR) located in Paris, the Laboratoire d'Analyse et d'Architecture de Systèmes (LAAS) in Toulouse, Department of Computer Science (IFI) and Digital Science Center (DiSC) both from the University of Innsbruck. ELSA is an acronym that stands for *Effective Learning of Social Affordances* which represents the main goal of this project. Every research center focus on a specific task to study and the main coordinator of the project is the Professor Mehdi Khamassi.

According to Cambridge's dictionary (Cambridge 2008), the word affordance is defined as *"a use or purpose that a thing can have, that people notice as part of the way they see or experience it"*. This concept was originally explored in psychology (Gibson 1979) and has influenced research in neurorobotics and AI in recent years.

Essentially, affordances are internal representations of the expected effect of performing a specific actions, and when and how their execution is relevant. When considering the field of AI applied to robotics and specifically the human-robot interaction, it is possible to distinguish between physical affordances and social affordances.

To discern these two different types of affordances, let us consider a simple example, with a robot (agent) interacting with a cube. The definition of physical affordance here refers to the characteristics or properties of the cube that enable or constrain certain actions or interactions by the robot, i.e. the perceivable features of the cube that indicate in which way the robot can interact with it. If the cube has a handle on one side, the physical affordance of the handle would indicate that the robot could physically grasp the cube, suggesting which actions it can perform. However, throughout this thesis work, we will mainly focus on social affordances, that can be divided in turn into human-general and human-specific social affordances. Within our RL algorithms, we tried to make the agent learn that there are certain characteristics that can be associated with humans as members of a group of individuals, but there are also certain characteristics that distinguish each individual. For example, we talk about human-general social affordance when the agent in question is able to understand that no human is capable of grasping a cube that is too far from its field of action. This affordance falls under this definition because this concept is extendable to all human beings, precisely because of our limited anatomical structure. On the other hand, within our algorithms we also tried to make the agent learn that there are humans with different characteristics than others, such as the fact that certain humans may be much more capable of building a tower from cubes than other humans; or that certain humans are more easily distracted than others, having a much lower attention threshold. These affordances fall under the definition of human-specific social affordances.

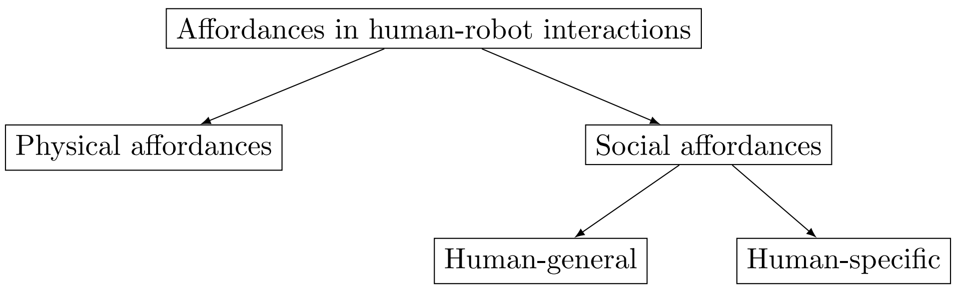


**Figure 2.1:** *Human-robot interactions affordances' graphical schema.*

This research field is very promising. While numerous scientific articles have addressed physical affordances, the same cannot be said for the social affordances that the ELSA project is dedicated to. As previously mentioned, the primary objective here is to equip robots with the capability to acquire social affordances. The two assumptions underlying the project are as follows:

- Robots can learn social affordances as they can learn physical affordances.
- Robots that can autonomously recognize social affordances when the human initiates an interactive action will more efficiently and appropriately respond to the human, thus facilitating human-robot coordination and cooperation.

In Figure 2.2, it is possible to notice the table-top scenario provided by the the LAAS laboratory. Human participants can cooperate with the robot to achieve a particular goal, that can be for example building a tower with cubes. In this scenario, the robot should learn to dissociate humans' competency in specific tasks from specific objects' properties.

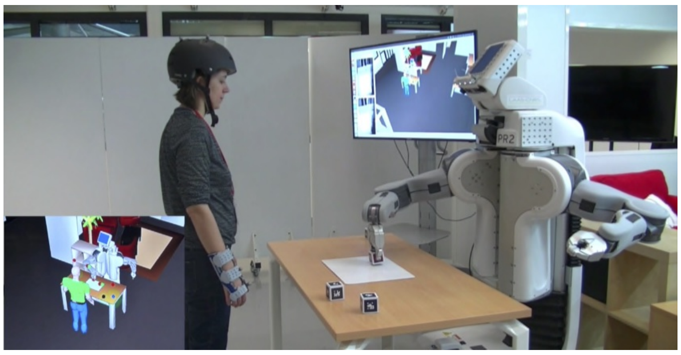


**Figure 2.2:** *Photo of a human-robot interaction experiment at the LAAS laboratory in Toulouse, featuring the table-top scenario. Credit: LAAS-CNRS*

The project is funded for four years, and within this thesis the initial research progress made in the first six months are discussed.

## 2.1 Visit the lab scenario

The *Visit the lab* scenario represents the Institut des Systèmes Intelligents et de Robotique's contribution to the ELSA Project, i.e. the realisation of a RL architecture capable of relating a navigation system and a social human-robot interaction system. In this task, our objective was to develop a simulation set within a laboratory environment featuring corridors, doors, and walls. Within this simulated laboratory, two central characters take center stage: a robot and a human participant. Within this context, the robot serves as the primary agent, tasked with acquiring the capability to provide guided tours to the human, who is unfamiliar with the laboratory layout. This entails the robot's ability to address navigation challenges and extends further into social interaction as the robot must engage with the human in a manner that captures their attention effectively.

In this context, the rooms that make up the laboratory are already labelled and provided to the robot, without any specification as to how they are obtained, thus

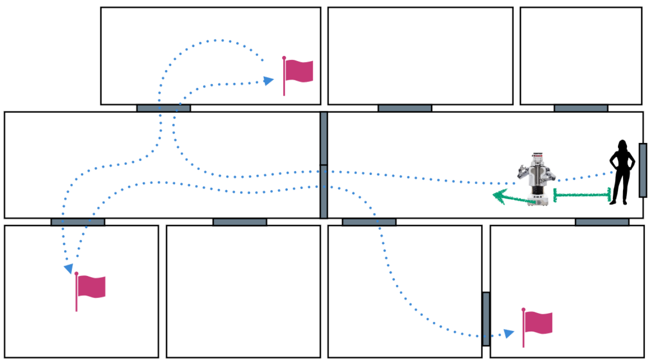


**Figure 2.3:** *Graphical representation of the visit the lab scenario. The robot has learn how to socially interact with the human, in order to guide him to the flagged rooms in which the navigation reward is located.*

avoiding dealing with the problem of symbol grounding[1].

The scenario necessitates the incorporation of two distinct modules: one dedicated to navigation and the other to social interaction. The navigation module enables the robot to traverse the predefined state transitions within the simulation environment, encompassing corridors, doors, and walls. Conversely, the social interaction module empowers the robot to engage with the human participant. However, a key challenge lies in the robot's capacity to autonomously learn when and how to employ these social interaction actions effectively to capture the human's attention.

The specific sequence of execution for these social actions is pre-defined by us and will be elaborated upon in the subsequent subsections, dedicated to formalizing the problem. Once the robot adeptly executes this sequence of actions, the human participant has been programmed to synchronize their movements with the robot's in the general case. Consequently, the robot's objective is twofold: to guide the human towards a designated goal area while simultaneously maintaining a degree of social interaction. For instance, this entails maintaining a specific distance between the robot and the human, ensuring that the social interaction remains intact. Should the robot stray too far, the risk of losing the social connection arises, leading to a potential loss of the human participant's step-by-step guidance.

To distinguish between specific and general social affordances associated with

[1]The symbol grounding problem is a long-standing challenge in AI and cognitive science. It refers to the difficulty of establishing a meaningful connection between symbols or representations in a computational system and the real-world objects or concepts they are intended to represent. An influential article inherent this theme is (Harnad 1990)

humans, we have created a class to model human behavior. This class defines a basic model of a human agent, incorporating characteristics such as speed, attention, orientation, and randomness as influencing factors. The actions and state of the human change in reaction to interactions with a robot and environmental conditions.

Many elements associated with social interactions, such as the field of vision, were introduced within the simulation. The robot will therefore not only have to maintain a certain distance from the human (neither too small because it would be socially inappropriate, nor too large because it would not communicate properly), but will also have to learn to interact with the human in front of it, face to face.

Before going into technical details, we can partition the *Visit the lab* scenario into a series of subtasks that the robot needs to learn:

1. Go to human vision;
2. Presenting, persuading and attracting the attention of the human;
3. Bring the human to the goal area.

At the moment, the task is implemented focusing on a simulation with discrete space and time, and with purely deterministic transitions between states. An extension of the work is planned in the future in which continuous notions of space and time are used: the problem will be technically solvable by means of function approximations methods, e.g. using neural networks that will aim to estimate action-value functions from on-policy data (Section 1.6).

In the next subsections we will clearly define the formalisation of the RL problem, i.e. we will describe states, actions, rewards and percepts concerning the two different modules of the *Visit the lab* scenario.

### 2.1.1 Navigation module

When referring to the *navigation module* we are essentially describing the simulation component responsible for modeling the dynamic behavior of the robot within its environment. This module encompasses not only the dynamics of the robot's states but also the formulation of rewards assigned upon the robot's successful achievement of a specified goal. In simpler terms, this module serves as the formalized framework for applying RL principles to teach an agent how to navigate to a particular room within the environment.

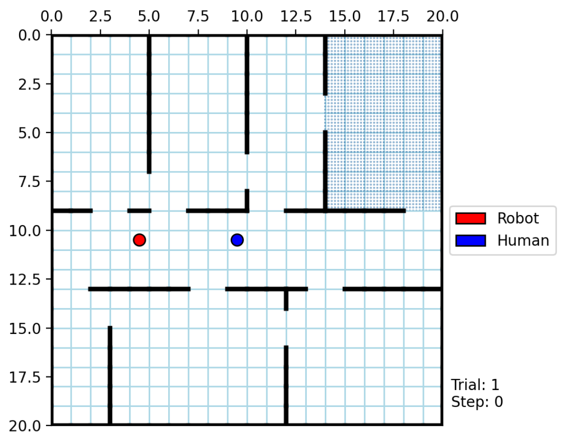


**Figure 2.4:** *Visualization of the Visit the lab scenario. Cells are numbered and represent markovian states. Walls cannot be crossed and serve as obstacles in the agent's (robot's) dynamics. Rewarded room is on the top-right corner.*

In Figure 2.4, one can observe a graphical representation of the simulation. The origin of the reference system, denoted as $(0, 0)$, is situated in the upper left-hand corner of the room. This setup allows us to uniquely identify each cell using a pair of coordinates. For computational convenience, we have also assigned integer numbers to each cell, ranging from 0 to $n$, where $n$ represents the total number of cells within our environment. The counting commences from the upper-left corner.
Formally, the set of possible states related to the navigation system is defined as:

$$\mathcal{S}_n = \{s \in \mathbb{N} : 0 \leq s \leq n - 1\}$$

Where $\mathbb{N}$ is the set of integers numbers. In this description, an $s$ value belonging to this set would represent the location for the Robot in the environment: e.g., $s = 74$ tells us the Robot is simply localized on the cell number 74.

Furthermore, one can observe that the environment has been divided into different areas. There is a central corridor and several labelled rooms: the layout of the room (number of rooms, number of vertical and horizontal doors, size of rooms) is randomly determined for sake of generality, but can be fixed setting a random seed. Still looking at the Figure 2.4, one can see that in the top right-hand corner there is a particular area marked by a light-blue shade. This is the goal area, in which each cell is marked by a reward worth +1; all the other white cells on the contrary are without any kind of reward. Since the reward area is enclosed in the environment, we can define a subset $\mathcal{S}r \subset \mathcal{S}_n$ as the subset that contains all the states constituting the goal area, so that it is possible to formalise the reward $r$ as a function of the states of the system $s$:

$$r(s) = \begin{cases} 1 & \text{if } s \in \mathcal{S}_r \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

Another fundamental step in designing our problem RL is to define the actions of our agent. In the navigation problem, a total of 25 dynamical actions were defined: considering eight different directions and three different speed levels, we have 24 actions plus the `STAY` action that allows the robot to just remain in the cell it is.

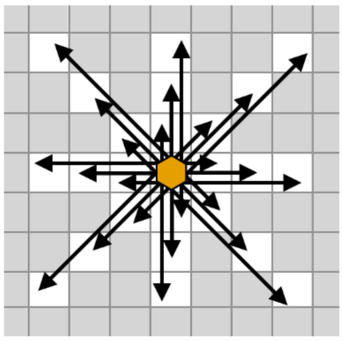


**Figure 2.5:** *Representations of the dynamical actions in the simulation. The directions are spaced at an angle of 45°, and the different size-steps constitute the speed of our agent. Credit: [Benoît Girard, ISIR]*

In this case, the set of possible actions related to the navigation system is $\mathcal{A}_n$, which contains all of these dynamical actions, encoded with integer numbers:

$$\mathcal{A}_n = \{a \in \mathbb{N} : 0 \leq a \leq 24\}$$

| | | | | | | |
|---|---|---|---|---|---|---|
| 0: UP | 4: UP2 | 8: UP3 | 12: UP-LEFT | 16: UP-LEFT2 | 20: UP-LEFT3 | 24: STAY |
| 1: DOWN | 5: DOWN2 | 9: DOWN3 | 13: UP-RIGHT | 17: UP-RIGHT2 | 21: UP-RIGHT3 | |
| 2: LEFT | 6: LEFT2 | 10: LEFT3 | 14: DOWN-LEFT | 18: DOWN-LEFT2 | 22: DOWN-LEFT3 | |
| 3: RIGHT | 7: RIGHT2 | 11: RIGHT3 | 15: DOWN-RIGHT | 19: DOWN-RIGHT2 | 23: DOWN-RIGHT3 | |

**Table 2.1:** *Table of the actions related to the navigation module.*

After formalizing the states and actions, the next step was to define the markovian probability matrix, as discussed in Chapter 1. However, this had to be done while taking into account the presence of walls affecting the system's dynamics. Initially, the elements $p(s'|s, a)$ were defined in a deterministic manner, without introducing any stochastic noise. Furthermore, in situations where the executed action would have resulted in the agent colliding with a wall, the action itself did not alter the system's state, i.e., $s' = s$.

To give an example, let us consider the top-left corner $s = 0$ of the Figure 2.4. Starting from this state, if we move to the right we arrive in state 1, which means that $p(1|0,\texttt{RIGHT}) = 1$, and thus the probability of ending up in state 1 given the fact that the agent is in state 0 and performs the action `RIGHT` is maximum. Similarly, given that on our left (and also above) we have a wall, if we start from this state and perform the action `LEFT`, we will still end up in state 0 due to the way the Markovian matrix has been defined, i.e. $p(0|0,\texttt{LEFT}) = 1$.

Like any RL problem, it will be up to the agent to figure out the best actions to take to achieve the goal. As input to the navigation system, we will provide two elements:

- The state (location) of the robot;
- The label of the room it is in.

These elements are the navigation percepts, that clearly will be different from the human-interaction percepts.

In the *Visit the lab* scenario, we developed also a *Gridworld* environment: the only difference to the former being the total absence of walls.

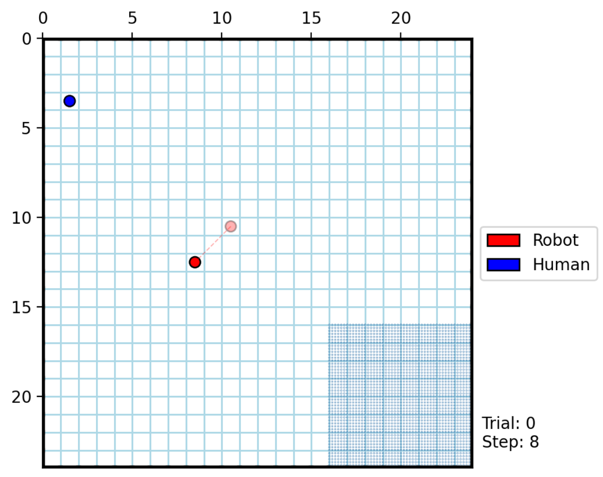


**Figure 2.6:** *Gridworld environment example. The reward zone is in the righter-down area of the environment, and the agent just performed the down-left action with the second level of speed, i.e., $a = 18$. This simple environment is convenient for testing algorithms.*

Considering the system just described, it is possible to run an RL algorithm to make the robot learn to reach the goal area of our environment. We are still far from our final goal, however, because we have not yet considered the human-robot interaction, contained in the social module.

### 2.1.2 Social module

The social module is in charge of attracting the human attention as well as selecting suitable actions to properly interact with them. This type of RL problem is more complicated to deal with, and therefore requires the introduction of new variables into our system. Let us start the description of this module by introducing the percepts of the social module, that can be graphically seen in Figure 2.7.

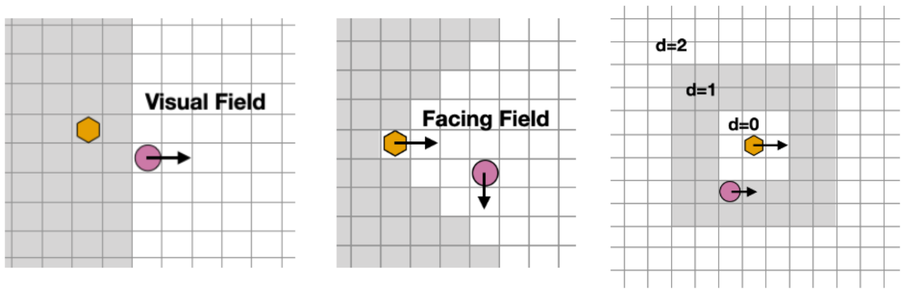


**Figure 2.7:** *Visual field, facing field and distance in the simulation, with robot in yellow and human purple. These percepts help to quantify the degree of social interaction between the human and the robot. [Benoît Girard, ISIR]*

Both characters in the scenario are characterised by a body direction, defined in the same way as the directions of dynamic actions in Figure 2.5. The relationship between the directions of the human and the robot and their positions define both the field of vision and the facing field.

- `Visible`: a boolean variable that informs the robot whether it is currently in the visual field of the human, as defined by the white area in the Figure 2.7 at the left. Here the robot, in the gray area, is not visible; and clearly the human does not see through walls.

- `RLooksAtH`: a boolean that informs the robot that the human is in his facing field. In Figure 2.7 in the middle, the robot is facing the human (`RLooksAtH` = True), but the human does not see the robot (`Visible` = False). If the human is in the facing field of the robot, hut hidden behind a wall, `RLooksAtH` will return False.

- `HLooksAtR`: same as `RLooksAtH` but with opposite characters.
- `Distance`: the distance between the human and the robot is measured with the Manhattan distance (see Appendix A), and then discretized in five categories. Either too close for the human to accept to follow the robot's movements ($d = 1$), at a correct distance ($d = 2, 3, 4$), or too far ($d = 5$).

With this description we are able to describe the set of states $\mathcal{S}_s$ related to the social module. Indeed, states can no longer be uniquely determined by the position of our agent: a method must be found to be able to quantify the level of social interaction between the robot and the human.
The states for the social system are originally based on a four-tuple as follows:

$$\mathcal{S}_s = \{\vec{s} \in \mathbb{N}^4 : \vec{s} = (interaction, distance, direction, angle)\} \tag{2.2}$$

before being mapped onto a 1D-array for computational reasons.
The four scalar components of the vector $\vec{s}$ can only take some predefined values:

- the parameter *interaction* $\in [0, 7]$ refers to the degree of interaction between the human and the robot. The larger this value is, the more the human is engaged with the robot. The values from 0 to 4 only refer to visual fields, and the human is engaged in the interaction process when the interaction parameter is 5 or larger.
- The parameter *distance* $\in [0, 4]$ is one of the percepts of the social module, and it is important because within the simulation, it adjusts the range of successful applicability of social actions.
- The parameter *direction* $\in [0, 7]$ refers to the required input direction, necessary to reach the goal destination. This value represents one of the possible directions that can be seen in Figure 2.5. In the vanilla social module, this direction is totally random.
- The parameter *angle* $\in [0, 7]$ represents the orientation of the human with respect to the robot. It is computed using trigonometric functions based on the difference in the positions of the robot and the human in the $(x, y)$ plane.

For example, $\vec{s} = (2, 4, 5, 3)$ where $\vec{s} \in \mathcal{S}_s$ would represents a social situation in which the human is looking at the robot; the robot is too far from the human because the distance is 4, the required direction is `LEFT-DOWN`, and the human is located `LEFT-RIGHT` with respect to robot. The goal of the agent would be to socially engage with the human and thereafter bringing him to the left-down direction of the environment.

Note how within this representation of states there is no parameter associated with the position of the robot or human, in contrast to the navigation module. But this makes perfect sense: it is not the absolute position of one individual or the other that matters, it is their relative position (both in terms of angle and in terms of distance) that quantifies the level of social interaction.

With regard to the actions of the social module belonging to the set $\mathcal{A}_s$, it was decided to keep all the actions contained in the navigation module and to introduce new ones, i.e. $\mathcal{A}_n \subseteq \mathcal{A}_s$, thus having both the dynamical and the social actions. One action enables the robot to look at the human. Moreover, another action referring to the greeting was introduced, exactly as one is used to start an interaction. After that, an action associated with the invitation to be followed was introduced, and an action also associated with pointing in a certain direction to suggest heading towards a destination. We can therefore extend the Table 2.1 with the following actions:

| |
|---|
| 25: LOOK H |
| 26: HELLO |
| 27: COME |
| 28: POINTING |

**Table 2.2:** *New actions introduced in the social module: look at the human, hello, come with me and pointing in the required direction.*

Clearly, a precisely defined sequence of actions must be followed to effectively enhance the *interaction* parameter that characterizes the states of the system. The following list illustrates the value of this parameter associated with the social situation:

- $0 \rightarrow$ No interaction
- $1 \rightarrow$ Human sees the robot

- ◆ 2 $\rightarrow$ Human is looking at the robot
- ◆ 3 $\rightarrow$ Human sees the robot, and robot looks at the human
- ◆ 4 $\rightarrow$ Human looks at the robot, and robots looks the human
- ◆ 5 $\rightarrow$ After the `HELLO` action
- ◆ 6 $\rightarrow$ After the `COME` action
- ◆ 7 $\rightarrow$ After the `POINTING` action

where the last three circumstances require that all previous above conditions are true.

In the formalization of reward-associated states, we made the decision to assign a positive reward of $+1$ to all system states where the level of social interaction between the human and robot reaches its maximum, and the robot is also progressing in the required direction. This markov decision process empowers the agent to learn the most effective strategy for engaging with the human, thereby persuading them to move into the designated required direction.

To model this simulation, a class representing humans was introduced. As discussed earlier in Section 2.1, introducing a class for humans has the purpose of formalizing various individual characteristics that distinguish humans. Within our human class, the following adjustable characteristics have been defined:

- the parameter *speed* $s_H = (p_1, p_2, p_3)$ is a ordered three-tuple which each element refers to the probability at each time step to move according to the $1, 2$ or $3$ speed level. For example, if $s_H = (0.5, 0.5, 0)$ then the human has a 50% chance to do a one-step movement and 50% chance to do a two-step movement; it will never go for three-step movement. Clearly, the sum of the probabilities must be equal one, i.e, the condition $\sum p_i = 1$ must hold.

- The *failing rate* $F_H \in [0, 1]$ parameter represents the probability related to the failure of the action `HELLO`. This parameter activates only if the human is seeing the robot and not actually looking at the human. In some cases, the agent shall learn to repeat twice this action when interacting with the human.

- Through the formalisation of the social module, we have decided to prevent certain humans from also needing the action `POINTING`. This concept was formalised through the *pointing need* $P_H \in [0, 1]$ parameter which governs the fact that in certain cases it will not be necessary for our agent to perform this kind of action. In fact, in these cases, after the `COME` action, the maximum level of social interaction is instantly reached.

- The parameter *losing attention* $L_H \in [0, 1]$ regulates the probability of needing to restart the entire social interaction process. This parameter is designed to model the real-life scenario in which humans can become distracted during interactions, such as stopping in front of a poster instead of following the robot. Its purpose is to simulate the human's loss of attention, requiring the agent to say `HELLO` again and restart the entire interaction process. However, if the human and the agent are facing each other, the *interaction* level will remain at 4. As will be demonstrated in Chapter 4, maintaining this value at a low level, approximately around 0.05, proves to be crucial in promoting successful social interactions.

- The *orientation change rate* $O_H \in [0, 1]$ parameter controls the probability that at each time step the human changes orientation randomly, when not engaged in human-robot interaction.

- The *random movement* $r_H \in [0, 1]$ parameter controls the probability that at each time step the human moves randomly, when not engaged in human-robot interaction.

These described features serve dual purposes. Firstly, they aim to characterize the human class, allowing us to define categories such as fast or slow humans, highly distracted or exceptionally attentive humans, and more. Secondly, they introduce a level of complexity into the RL problem: including these stochastic parameters increases the challenge for our agent, necessitating a greater number of iterations for the algorithms. However, this complexity also offers the opportunity to train agents on various human classes, and to quantitatively study the adaptability of these agents to different human behavioral types.

### 2.1.3 Go to human option

We have established the navigation module for reaching a goal area and the social interaction module for engaging with the human. However, a crucial step remains: how do we instruct the agent to approach and reach the human? It is important to note that reaching a designated location and reaching a dynamic character like a human are distinct tasks. Generally, a human can make random movements and change their orientation stochastically in our environment.

To address this challenge, we introduced the *Go to human* module, whose purpose is self-explanatory. In this module, the states of the system are defined by:

$$\mathcal{S}_g = \{\vec{s} \in \mathbb{N}^3 : \vec{s} = (orientation_H, distance, angle)\} \tag{2.3}$$

where we introduce the new parameter $orientation_H \in [0, 7]$ which represents the orientation of the human. Indeed, this new parameter plays a crucial role in enabling the agent to learn how to navigate into the human's visual field. Comparatively, in contrast to the *Social* module, we remove two parameters: those dedicated to modeling the interaction level and the direction for moving the human. Instead, we introduce a single parameter: the human's current orientation.

Given that the task primarily involves agent movements, the set of actions in the *Go to human module* aligns with those in the navigation module, resulting in $\mathcal{A}_g = \mathcal{A}_n$.

Our agent's objective is to learn to position itself within the human's field of vision effectively. To achieve this goal, we assign a reward of $+1$ for each time step in which two conditions are met concurrently: first, the robot must be within the human's facing field, and second, the distance between the two entities must be within the range of 1 to 3.

| 29: GO TO HUMAN |
|---|

**Table 2.3:** *The multi-step action Go to human is thought firstly as task, and then it is encoded as an option, i.e., a multi-step action in the framework of Hierarchical Reinforcement Learning.*

What sets this module apart is our intention to encode the ultimate task accomplishment as an *option* in the framework oh HRL, as previously discussed in Section 1.7. The concept here is to train our agent to execute the task of reaching the human's facing field, which involves numerous dynamic actions, and subsequently encode this task into a single action.

More specifically, this option is implemented in the learning architecture that is designed to combine the different tasks of navigation and social interaction. This type of framework is particularly useful when the action one would like to perform is actually made up of many, small actions. In fact, it is natural to think that the task of reaching the human's vision is actually made up of many small subtasks: turning towards the human, walking dynamically towards his position, and adjusting the orientation in case the human has meanwhile turned and/or changed position.

In Table 2.4 is represented a recap of the modules that have been developed:

| | **Navigation** | **Go to human** | **Social interaction** |
|---|---|---|---|
| **States** | $\mathcal{S}_n$ | $\mathcal{S}_g$ | $\mathcal{S}_s$ |
| **Actions** | $\mathcal{A}_n$ | $\mathcal{A}_n$ | $\mathcal{A}_s$ |
| **Reward** | R reaches goal | R goes H visual field | Max interaction & H following required direction |

**Table 2.4:** *States, actions and reward designed for the Navigation, Go to human and Social interaction modules. Specifically,* $\mathcal{S}_n = \#$ *cells,* $\mathcal{S}_s = \{interaction, distance, direction, angle\}$, $\mathcal{S}_g = \{orientation_H, distance, angle\}$. $\mathcal{A}_n = \{0, ..., 25\}$. $\mathcal{A}_s = \{0, ..., 28\}$.

# Chapter 3

# Materials and methods

This Chapter will outline the materials used and the methodology employed to carry out the thesis project. It will focus mainly on two areas: one associated with the simulation carried out, and that associated with the experimental apparatus constructed.

With regard to the methodologies used for the development of the simulation, part of this topic has already been addressed in the Chapter 2. However, the learning architecture employed to make the different learning tasks communicate has not been addressed: with classes implemented towards the learning of different tasks, how can these classes be made to communicate? That is, if two RL problems have been implemented in two parallel tracks, how can these two scenarios be merged into a single, unified problem?

With regard to the experimental methodologies employed, it is necessary to introduce the characteristics of the robot employed within the experiment. Beyond this, attention will be paid to the Simultaneous Localization and Mapping (SLAM) algorithm, employed to construct markovian states in a real-world scenario, and attention will be paid to the distinction between a matrix of probability transitions within the simulation, and a matrix of probability transitions experimentally derived. All of this is accomplished through the renowned Robot Operating System (ROS) framework, which serves as a valuable tool for implementing the learning algorithms within our robot.

## 3.1 Simulation

While we have methods to instruct an agent on reaching specific areas within an environment, teaching it how to approach humans and engage in social interactions, the challenge lies in integrating these distinct modules synergistically once they've been formalized. When faced with scenarios where an agent needs to master multiple tasks, the field of Multi-Task Reinforcement Learning provides a theoretical framework to address such complexities (Zhang and Yang 2021). In the context of this thesis project, our aim was to unify these modules by developing a novel architecture developed by us, that seamlessly integrates the navigation and social interaction components.

### 3.1.1 Learning architecture

The learning architecture we developed requires three main ingredients: the Q-tables associated with the three specific tasks that make up the human-robot interaction, i.e., *Navigation*, *Social* and *Go to human* modules. It is therefore necessary to train the three tasks individually through offline reinforcement learning, so that the acquired knowledge from the agent is available.

Once we trained the agent, the following steps must be done:

1. Considering the learned Navigation Q-table, we focus on the value $Q(s = s_H, a)$, i.e. the Q-values associated with the actual position of the Human.

2. We define a speed coefficient $w = (p_1, p_2, p_3)$ vector representing the weight we want to assign to the three different velocities implemented in simulation. Then we weighs each action's Q-value based on the associated speed coefficient.

3. Aggregate the weighted Q-values for each direction: the direction with the largest value compose the `required_direction` array. This array then will be used as input into the social module. In this way, this module has information regarding which is the direction associated with the rewarded area.

4. Repeat the process at each times step. Moreover, if the agent chooses the action 29, i.e., `GO TO HUMAN`, then read up the Q-table associated with the *Go to human* module in the state of the robot $Q_G(s = s_R)$ to select the proper action.

We implement the speed coefficient to allow for adaptation to the varying speeds of humans. In practice, we set an arbitrary value (particularly high for $p_3$), and this approach proved to be effective, e.g. we set $w = (0.1, 0.1, 0.8)$. A simple schema illustrating this learning architecture can be seen in Figure 3.1

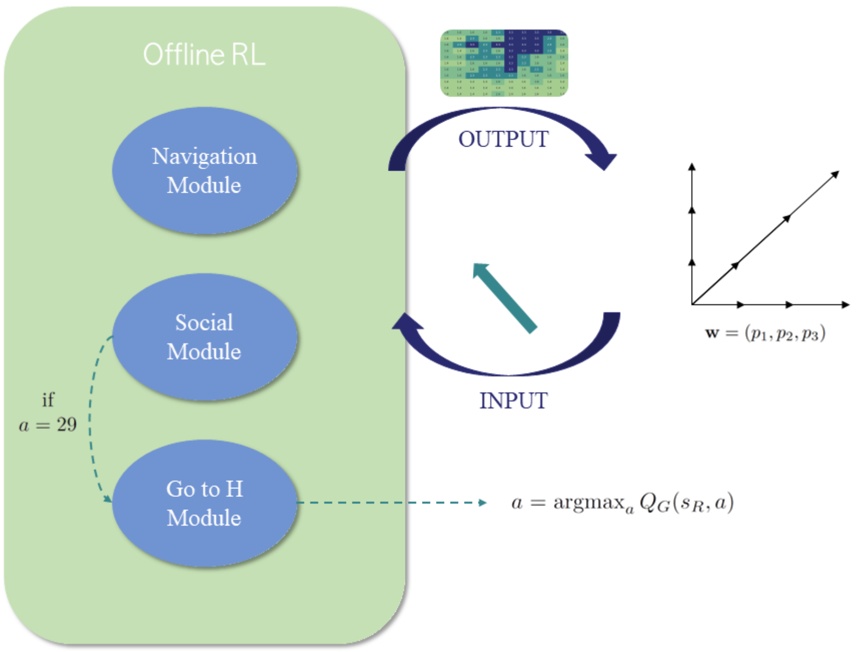


**Figure 3.1:** *Developed learning architecture for combining navigation and interaction tasks. Every module has been learnt trough offline RL. We consider the learned Navigation Q-table in the position of the Human, then we weigh all the 24 dynamical actions with the speed parameter in order to get a required direction that will be fed into the social module. If the agent chooses the action 29, read the Q-table from Go to Human module and select the optimal action according to* $a = \mathrm{argmax}_a\, Q_G(s_R, a)$.

It's important to highlight that within the proposed architecture, the social module exerts a significant influence as it governs the action selection process. Notably, the chosen action, unless it's $a = 29$, always comes from the social module.

In order to evaluate the performance of this architecture and the modules presented above, simulations were run in which the reward obtained by the agent as the number of trials increased was observed. The details of these simulations are discussed in Chapter 4.

## 3.2 Experiment

Structuring an experiment involving a robot and RL algorithm requires careful planning and the integration of various essential components. From one hand, we have to develop software capable of:

- enabling the robot to learn a specific task.
- Establishing the connection between the developed algorithm and the robot's motherboard.
- Providing the robot with an understanding of its physical environment, which involves creating programs capable of reading sensor measurements.

On the other hand, we also need to reconsider the entire scenario in terms of mapping the environment. The *Visit the lab* scenario is a simulated environment that differs from a real-life setting, such as a physical table. Therefore, we need to obtain a map of this new environment, and to achieve this, we can leverage a combination of the LIDAR sensor (see Appendix B) and the SLAM (Simultaneous Localization and Mapping) algorithm. The latter algorithm initially enables us to map the space, meaning it estimates a function $\mathcal{F} : (x, y, z) \rightarrow n$ that associates coordinates in the space with a markovian state of the system, represented by an integer number $n$. The LIDAR sensor measurements enable the robot to avoid collisions.

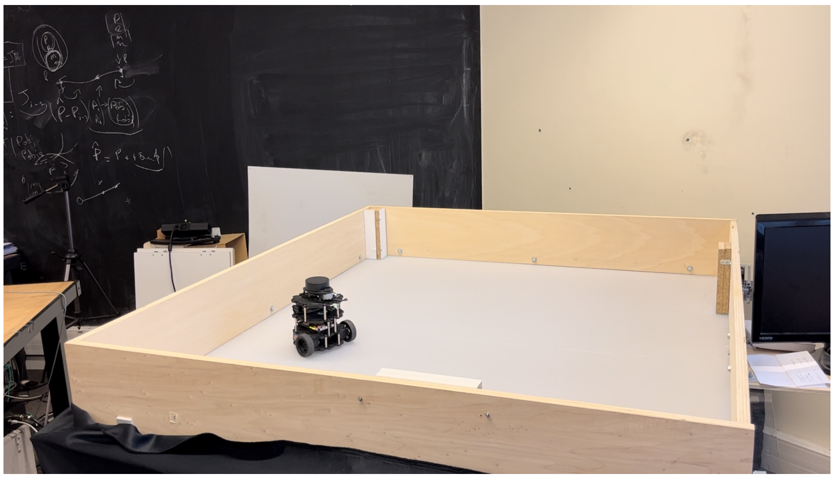


**Figure 3.2:** *Experimental environment of the project, at the ISIR laboratory. A square table with approximately a side of* 1.5 *m, with a Turtlebot3 Burger.*

In Figure 3.2 it is possible to see a photo of the table used as experimental environment. In our implementation, the Turtlebot3 Burger robot can move at two different speed levels and in eight different directions, resulting in a total of 16 available actions.

### 3.2.1 TurtleBot 3 Burger

In our project, we utilized the TurtleBot3, a compact and programmable ROS-based mobile robot, for experimental testing within a controlled environment. This versatile robot, equipped with a LIDAR sensor, served as the ideal platform for evaluating our algorithms. Developed by ROBOTIS in collaboration with the Open Source Robotics Foundation (OSRF), the TurtleBot3 Burger is a part of the TurtleBot series, renowned for its accessibility and adaptability in the field of robotics (Robotis 2023). It offers users the flexibility to incorporate additional sensors, cameras, and computing hardware, allowing for precise customization to suit specific research or educational needs.

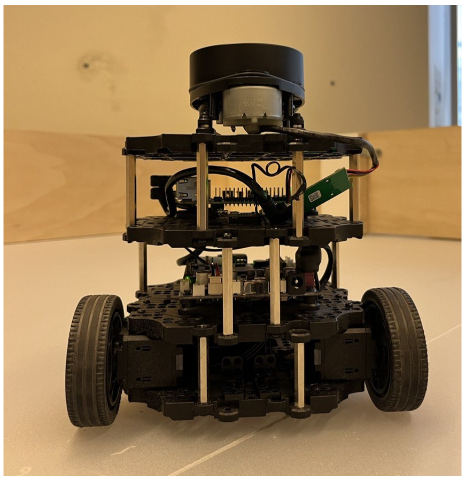


**Figure 3.3:** *Close look-up to the Turtlebot3 Burger equipped with a LIDAR sensor. One can see wheels, motherboard, (hidden) battery and connecting cables.*

The TurtleBot3 Burger exhibits stable maneuverability in various environments, making it well-suited for a wide range of applications. Its mobility is further enhanced by its compact design, making it easy to navigate through confined spaces and interact with objects in its surroundings. This capability opens up opportunities for research and experimentation in fields such as autonomous navigation: in fact, it is a proper tool on which it is possible to test RL algorithms.

In the Table 3.1 it is possible to see some physical property of the Turtlebot3 Burger.

| Physical property | Expected quantity |
|---|---|
| $v_{max}$ | $0.22\ m/s$ |
| $\omega_{max}$ | $2.84\ rad/s$ |
| $l \times w \times h$ | $138 \times 178 \times 192\ (mm)$ |
| $m$ | $1\ kg$ |
| $\Delta t_{op}$ | $2\ h\ 30\ m$ |
| Power connectors | $5.0\ V/4.0\ A$ |
| Maximum payload | $15\ kg$ |
| Single Board Computers | Raspberry Pi |
| Programmable LEDs | User LED $\times 4$ |
| Battery | $11.1\ V\ 1800\ \ mAh/19.98\ Wh$ |

**Table 3.1:** *Turtlebot3 Burger hardware specifications, with physical property and expected quantity. Maximum velocity, angular velocity, geometrical lengths and maximum operation time. Credits: (Robotis 2023).*

### 3.2.2 Mapping

When talking about autonomous systems, Simultaneous Localization and Mapping, often abbreviated as SLAM, stands as a critical problem. It represents a complex task that robots face when navigating through unknown environments; indeed SLAM addresses the question of how a robot can simultaneously construct a map of its surroundings while accurately determining its own position within that map (Durrant-Whyte and Bailey 2006). This simultaneous process is inspired to the way humans perceive and navigate the world around them: humans effortlessly understand their environment, recognizing landmarks and orienting themselves within it, and robots equipped with SLAM capabilities seek to achieve a similar level of spatial awareness and self-localization (Aulinas et al. 2008).

The crux of the SLAM challenge arises from the fact that robots typically lack prior knowledge of their environment, requiring them to explore and map it autonomously. Furthermore, they must accomplish this task in real-time, often in dynamic and unpredictable settings. To surmount these challenges, SLAM algorithms combine sensor data, such as laser scans, camera images, or sonar readings, with computational techniques to simultaneously estimate the robot's pose (position and orientation) and construct a coherent map of the environment.

Even tough we will not address the details of this algorithm, the general idea is to find an estimate of the following quantity:

$$P(m_{t+1}, x_{t+1} | o_{1:t+1}, u_{1:t}) \tag{3.1}$$

where $m_t$ represents the map of the environment at time $t$, $x_t$ is the state of the agent at time $t$, $u_t$ represents a series of controls and $o_t$ a sensor observation. Thus, we want to compute an estimate of the map of the environment and the agent's state at the next time step. Applying Bayes' rule it is possible to update the location posteriors:

$$P(x_t | o_{1:t}, u_{1:t}, m_t) = \sum_{m_{t-1}} P(o_t | x_t, m_t, u_{1:t}) \sum_{x_{t-1}} P(x_t | x_{t-1}) P(x_{t-1} | m_t, o_{1:t-1}, u_{1:t})/Z \tag{3.2}$$

where $P(x_t | x_{t-1})$ is the transition function.

In this project, the Gmapping (Grid-based Mapping) algorithm was used to map the experimental environment. It employs a grid-based representation of the environment, where each grid cell can denote a state. Using sensor measurements such as the ones coming from LIDAR, GMapping probabilistically updates these grid cells, progressively constructing a map. Simultaneously, it refines the robot's pose estimate by iteratively minimizing the discrepancies between sensor measurements and the expected map. GMapping excels in indoor environments with structured layouts, and it is favored for its robustness and accuracy (Weigl et al. 1993).

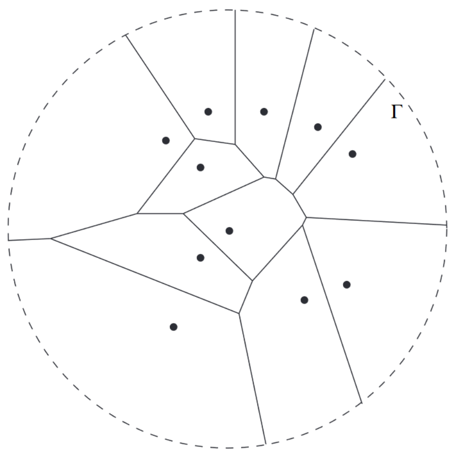


**Figure 3.4:** *A voronoi diagram of 11 points in the euclidian space represented by* Γ. *Credits: (Aurenhammer and Klein 2000).*

More specifically, Voronoi centers play a pivotal role in GMapping, particularly in the context of path planning and navigation within complex environments. The Voronoi diagram is a geometric construct that partitions space into regions based on proximity to points known as Voronoi centers (Aurenhammer and Klein 2000). In Gmapping, these centers serve to extract meaningful features from the environment, aiding in landmark recognition and map creation. By identifying Voronoi centers within the map, robots gain a better understanding of the spatial layout of their surroundings, enhancing their navigational capabilities. An intuitive example of what is a Voronoi diagram can be seen in Figure 3.4.

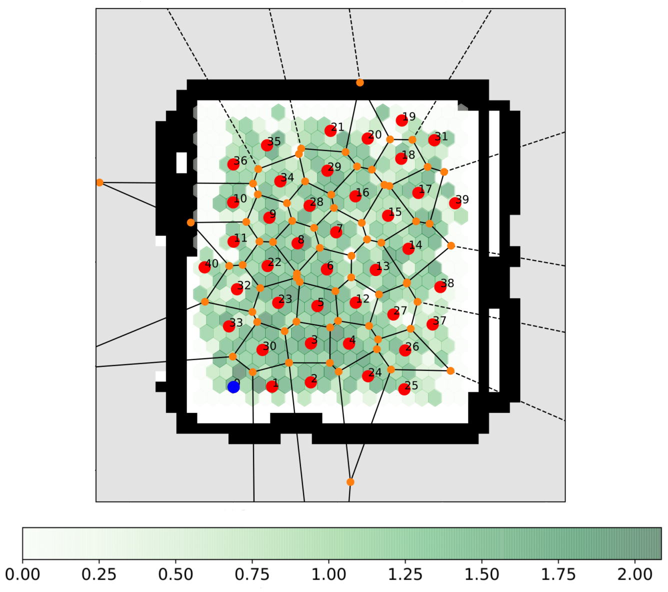


**Figure 3.5:** *States decomposition performed by the gmapping algorithm, with an exploration time of $\Delta t = 73.1$ min and a state radius of $r_V = 0.09$ m. In total, we found $N = 41$ markovian states. In the lower part of the figure, a legend of the displayed-grid colors in the state, computed by dividing the times the robot walked in the cell over the $\log_{10} N$.*

In summary, we exploited SLAM with the Gmapping algorithm in order to construct a map of the environment, and to constantly encode the localized position of the robot in a markovian state. The Gmapping algorithm gave us possibility to define two parameters:

- the number of states $N$, representing the number of wanted markvoian states.
- the Voronoi radius $r_V$, i.e., the physical value determining the size of the

neighborhood around each laser scan point where GMapping considers other points for calculating the likelihood of occupancy.

The basic framework we used to establish a communication between the RL algorithms and the robot was the already mentioned ROS (Robot Operating System).

### 3.2.3 ROS

In the realm of robotics and automation, the Robot Operating System, commonly referred to as ROS, is a pivotal tool and framework that has redefined the way we design, develop, and deploy robotic systems (Koubâa et al. 2017). ROS is a flexible and open-source middleware platform that plays a fundamental role in enabling the creation of robotic applications. It provides a comprehensive suite of libraries, tools, and capabilities that enable developers to build, simulate, and control robots with versatility.

At its core, ROS is built on a distributed computing framework, allowing for the seamless integration of various software components (Koubâa et al. 2017). It operates on a modular architecture, where individual nodes, can be developed independently and communicate with one another, forming a network of interdependent entities. This decentralized approach empowers developers to create complex robotic systems by composing smaller, specialized components.

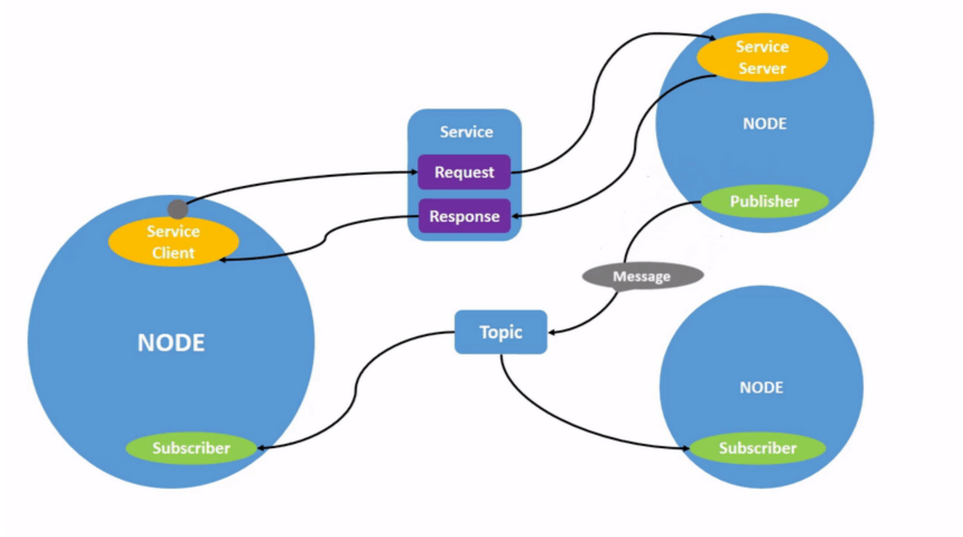


**Figure 3.6:** *ROS framework architecture. A full robotic system is comprised of many nodes working in concert. Credit: (Macenski et al. 2022)*

Key concepts within ROS are nodes, messages, topics and services.

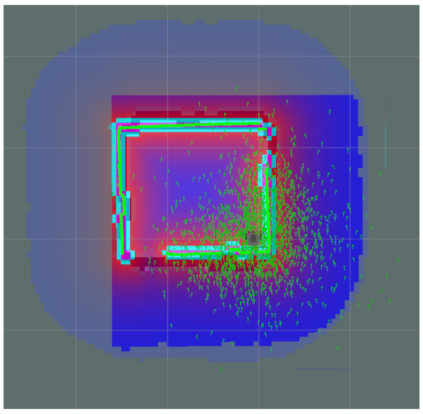
- **Nodes** are the fundamental building blocks of a ROS application: they consist in software modules or processes that perform specific tasks or computations within the robotic system. They can be written in various programming languages like C++ and Python. The most important aspect is that node can communicate with each other by passing messages through topics, enabling modular development and easy integration.

- **Messages** are structured data types that nodes use to communicate with each other: they encapsulate information or instructions, allowing nodes to exchange data. ROS provides a rich library of predefined message types for common robotic data, such as sensor readings and control commands; developers can also define custom message types tailored to the specific needs of their robotic applications.

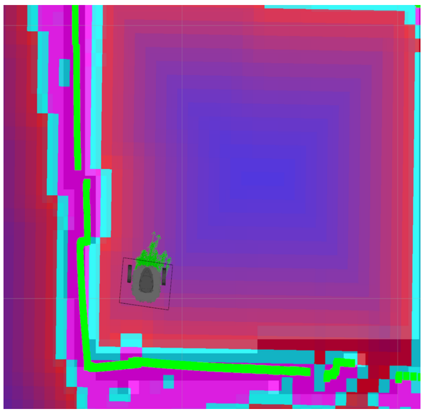
- **Topics** serve as the communication channels or conduits through which nodes exchange messages. A topic is like a named pipe or a message bus where nodes can publish messages for others to consume. Multiple nodes can subscribe to the same topic to receive the messages they are interested in, enabling seamless data flow within the robotic system.

- **Services** provide a request-response mechanism for performing specialized tasks within ROS. Nodes can offer services, and other nodes can request these services to perform specific actions or retrieve information.

Moreover, we employed RViz: a 3D visualization tool used in the ROS ecosystem, designed to help developers visualize various aspects of their robot's operations and environment in real-time. RViz was used to visualise the trajectories the robot executed in the real environment, and to visualise the accuracy of the localisation algorithm. This phenomenon can be observed in Figure 3.7, located in the next page.



(a)



(b)

**Figure 3.7:** *Localization performances comparison.* ***(a)****: full RViz scenario. Different walls: black is referred to the physical reconstructed map coming, extracted from the gmapping algorithm. Green line is given by the LIDAR sensor; in purple the tolerance precision delimited by the cyan lines. Robot is still, and the localization is not precise.* ***(b)****: zoom-in of the robot moving in the environment. The localization algorithm is more precise, since the green arrows identifying the position of the robot are less spread.*

### 3.2.4 State machine

The developed architecture for the experimental part of this project consists of several interconnected ROS nodes, plus a useful bash file to automate all processes. The developed scripts are the following:

- `agents.py`: classes container of RL algorithms, useful both for the simulation and the experimental part.
- `qserver.py`: server node useful to establish connections between the RL algorithms contained in the `agents.py` and the experimental environment.
- `reading_pos.py`: node designed to read the position of a robot and publish it in the map reference frame.
- `detect_obstacles.py`: by incorporating the information obtained from the LIDAR sensor and setting a certain distance treshold, this node aims to return a True boolean variable when an obstacle in front of the robot is detected.
- `go_back_initial_pose.py`: useful node to make the robot return to its initial position when the episode (trial) ends.

- `state_machine.py` node that utilizes learning algorithms and sensors to enable the robot to navigate within its environment. It handles state transitions, reward management, and communication with other ROS nodes to control the robot's behavior.

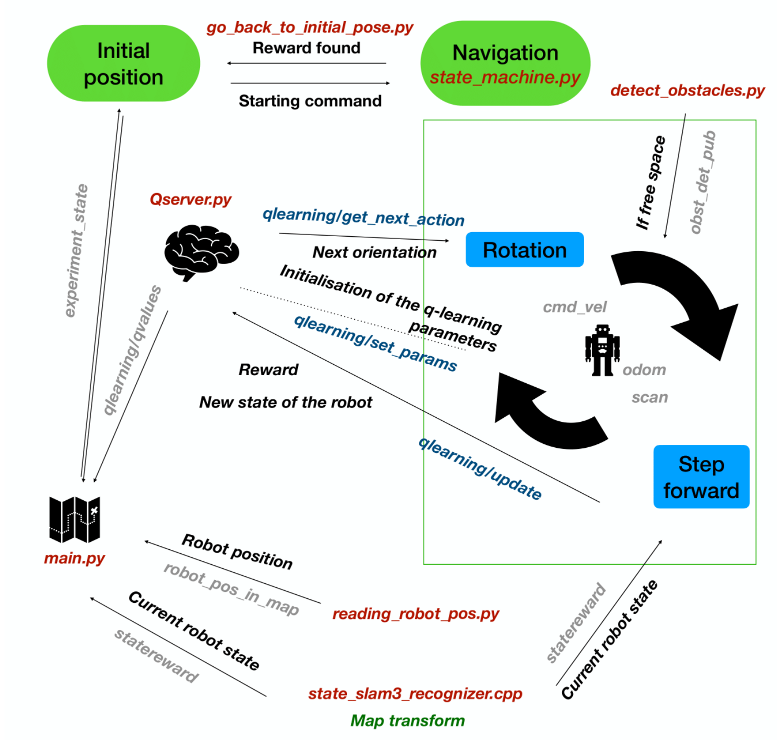


**Figure 3.8:** *ROS schema of the experiment. In red the name of the scripts nodes, in navy blue the services and in grey the topics. Credits: [Elisa Massi]*

The sequence of steps to take in order to start an experiment was:

1. Initialize connection with the robot. This can be done e.g through SSH remote protocol connection.

2. Run `roscore`: a collection of nodes and programs that are pre-requisites of a ROS-based system (Macenski et al. 2022).

3. Load the preferred navigation map. This map contain informations about the borders of the environment and the markovian states location.

4. Run `detect_obstacles.py`, useful to avoid collision of the robot with the environment. In this context, the LIDAR sensor that scans the obstacles has paramount importance.

5. Run `reading_pos.py`, in order to that assign to each location of the robot $(x, y, z)$ a scalar integer number associated with the markovian state.

6. Run `go_back_initial_pose.py`.

7. Run `qserver.py` node.

8. Run `state_machine.py` node, the one in charge of orchestrating between all the above mentioned nodes.

In Figure 3.8 is exposed a summary diagram of the architecture used to make the robot face a navigation problem.
The Turtlebot starts in a certain initial position. Instantly, the `main.py`, communicate with the `qserver.py` that outputs the action the robot will take. With the LIDAR, the `detect_obstacles.py` node checks if there is an obstacle in the direction of the next actions, and if not, the robot performs the action, ending up in the next state and updating the Q-values. In the meanwhile, the position of the robot and current state are constantly read, and if the Turtlebot finds a reward, it goes back to the initial position and a new trial start.

This is the general of the conducted experiment: however, in our case, we suppressed the `go_back_initial_pose.py` node so that when the Turtlebot got a new rewarded area it did not go backs to the starting state.

It is important to emphasise that different types of experiments can be carried out with this architecture. For example, it is possible to make the robot tackle an online navigation problem, i.e. by making the robot actually learn by trial and error. After a certain number of hours, the robot is expected to have learnt a good policy, thus being able to reach the rewarded areas in a short time. This knowledge can be saved via $Q(s, a)$ values, and thus it is also possible to exploit the acquired knowledge without any learning process. Another efficient method to tackle this problem in real-life is to train an agent to solve the navigation problem through simulation *Visit the lab* in the `Gridworld` environment, and then simply exploit the acquired knowledge in real-life.

### 3.2.5 Experimental Transition Probability Matrix

In Chapter 2, probability transition matrices $P(s, a, s')$ were formalised for the different tasks implemented in the *Visit the lab* scenario. Thus, the dynamics were defined, i.e. it was clear in which state our agent ends up after performing an action in a defined state of the system. This was done for all the three modules implemented in the task.

When referring to this mathematical object, however, it is necessary to make a clear distinction between simulation and real experiment. Within our simulation, transitions between states have been defined in a purely deterministic manner: e.g., by performing the action `RIGHT` in state number 0 of the system, in 100% of the cases our agent will end up in state 1 of the system. This is always true, and this type of dynamics has been chosen by us on purpose. However, it is also possible to introduce noise into this probability transition matrix: e.g. in this case, introducing an 80% probability of ending up in state 1 of the system, and perhaps a 20% probability of ending up in state 2, where in this case our agent would take a longer step to the right.

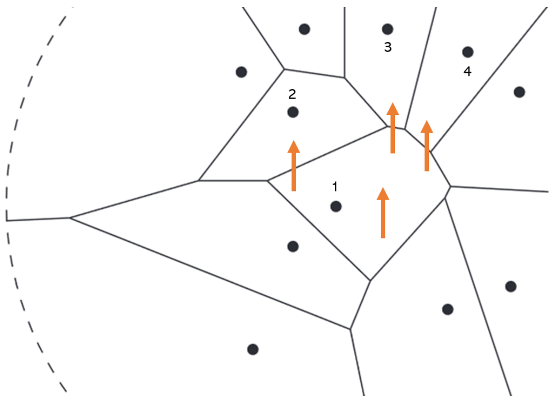


**Figure 3.9:** *Transitions in real scenario example. The SLAM algorithm decompose the states with non-predefined geometrical structure, thus providing a stochastic transition matrix. In orange, the action "UP" executed from state* 1 *can bring towards different next states:* $2, 3, 4, 1$.

This kind of stochasticity is obvious when dealing with a real experiment, where transitions between states are associated with probabilities (see Figure 3.9), that can be empirically estimated by letting the robot move within the environment. For this reason, we let the robot walking for approximately 10 hours, saving a total of 8448 transitions with the aim to execute all the possible state-action couples $(s_i, a_i)$ in the environment and see the resulting next states $s'_i$.

More specifically, in order to obtain the experimental probability transitions $P(s, a, s')$, we stored the number of times each $(s, a, s')$ triplet was encountered by the robot, then we divided by the number of times $(s, a)$ experienced, as explained in Equation 1.13. In order to be able to map all possible triplets $(s, a, s')$, the robot was set in motion using the $\epsilon$-greedy algorithm with a value of $\epsilon = 1$ as the action selection method, so that at each state of the system the action to be performed would be completely random, so that to explore the system as much as possible.

This procedure was extremely useful because it allowed us to run a simulation of the *Visit the lab* scenario, using the extracted probability transition matrix of the experimental environment. Thus, it was possible to use RL algorithms to train an agent to solve a navigation problem in a real map, obtaining a good policy $\pi$ that could be exploited by our Turtlebot. Indeed, after the training part, a Q-table has been obtained containing the expected rewards for each action-state pair, for the different reward areas that were arbitrarily and manually defined in Figure 3.10.

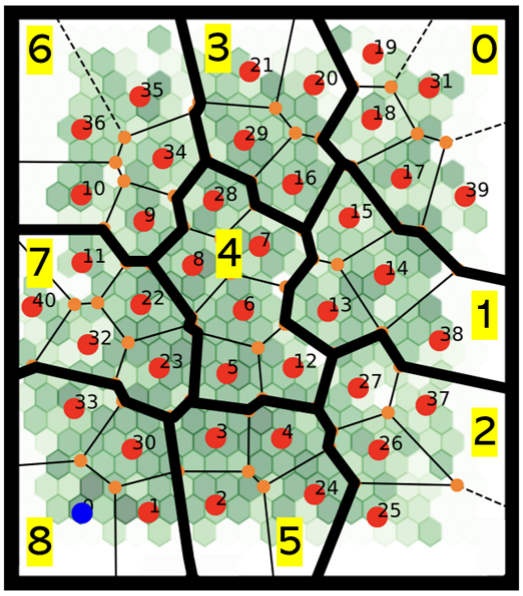


**Figure 3.10:** *Experimental goal areas. We selected and labelled a total of* 9 *different goal areas in the experimental environment.*

With this process, we have brought simulation and reality closer together: the `Gridworld` environment consists exclusively of states (cells), like the table environment after discretizing the states of the system. In the `Gridworld` environment, different goal areas can be defined, exactly as can now also be done experimentally. Thus, the only major difference between the two scenarios is the different transitions between states that the agent can make; but these transitions can be saved and used within the simulation.

The next step therefore remains the presentation of the results associated with the simulation *Visit the lab* in the `Gridworld` environment, those of the learning architecture found and the experimental one perfromed by Turtlebot.

# Chapter 4

# Results

The results that will be shown are mainly scatter visualisations from simulated or experimental data. When it comes to RL, the goal is to make an agent learn to perform a certain task: to determine this quantitatively, one can resort to the sum of the rewards an agent obtains at the end of each trial, or another visualisation can be given a Q-table heat map, in which deeper colours are associated with states in which the Q-value is higher.

With regard to simulation, data from the learning of the *Navigation*, *Go to human*, and *Social interaction* tasks will be presented, and the exploitation of the knowledge of the three tasks learned by the agent will also be presented. The difference in training the social interaction with diverse classes of humans - easy, medium, difficult - will also be shown, and the adaptability of the agent towards different kind of humans will be exposed. For what concerns the experimental part, an interesting fact will be shown concerning the entropy of the experimentally extrapolated probability transition matrix. Furthermore, the agent's training on the newly extrapolated transitions will be presented, as well as the verification of our Turtlebot's cumulative reward performance in the experimental scenario.

## 4.1 Navigation

As was anticipated in the Chapter 1, there is a relevant distinction between Model-Based and Model-Free agents. Let us start by considering a comparison of the two between the performances over the number of trials.

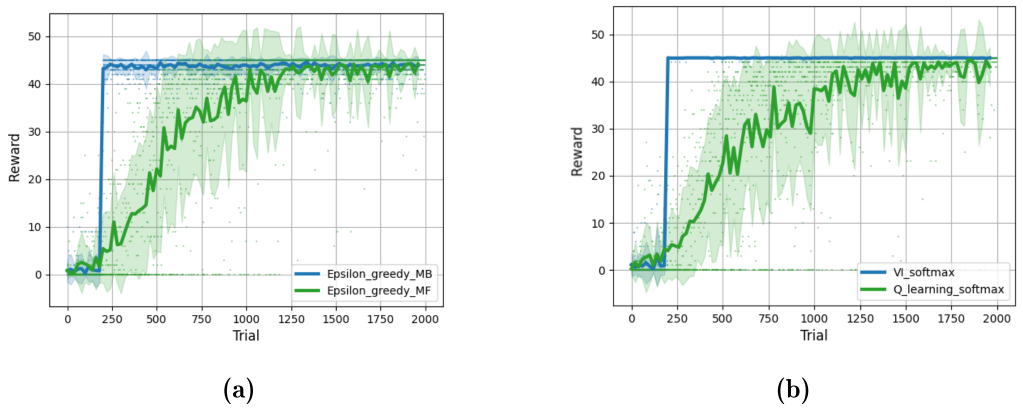


**Figure 4.1:** *Model-Based VS Mode-Free performance comparison on the Navigation task.* ***(a)****: learning curve of $\epsilon$-greedy Model-Based in blue compared with the $\epsilon$-greedy Model-Free algorithm in green.* ***(b)****: Value Iteration in blue and Q-learning in green, both with softmax as action selection method. Parameters used:* $\gamma = 0.9$, $\alpha = 0.7$, $\epsilon = 0.05$, $\tau = 0.1$. *For the Value Iteration, the number of maximum iterations equals* 50, *and the update is done each* 10000 *steps.*

In the Figure 4.1 it is possible to observe two simple plots in which the reference values are on the x-axis the number of trials (episodes) performed by the agent taken into consideration, while on the y-axis we have the value of the reward obtained by our agent on each trial. Each trial consists of 50 steps: at each step, the agent is in a certain state of the system and must perform an action. The code was written in such a way as to obtain a series consisting of 100 reward values: thus if the total number of trials as shown in the plot is 2000, this means that the moving average was carried out over a window of 20 original values. In addition to the main lines, one can also see shaded areas associated with the standard deviation of the selected moving window: its purpose is to understand the variability of the rewards obtained.

In this precise context, the problem addressed is a simple navigation task: there is a goal area in the system marked by reward +1 (see Picture 2.6). It is relevant to underline that in this context we do not have any interaction with the human; it is not even in the environment.

These Figures underline the important difference between a Model-Free and a Model-Based agent: the speed of learning. Looking at the blue lines, we can see that after about 250 trials our agent seems that it has already figured out the strategy needed to obtain the maximum reward value, i.e. choosing in sequence the actions needed to go from the initial state of the system until the reward area is reached. The average reward value seems to be about $\simeq 45$: this means that the agent takes about 5 steps to reach the goal zone, after which he either performs the action `STAY` many times, or simply wanders around the goal zone. Similarly, the Model-Free algorithms were also able to work out what the winning strategy is. The only difference is the amount of episodes required: in fact, it seems that Model-Free agents learned this after about 1250 trials, having no model of the environment and being forced to update the Q-values $Q(s, a)$ only by trials and errors.

These plots derive from the situation in which at each trial, the agent spawns in a specific state of the system (which does not change) and likewise, the area in which the reward is present is always the same. Thus, the conditions are extremely "fixed": and for this reason the standard deviation associated with the blue lines is practically non-existent as far as Model-Based algorithms are concerned. In the next plots we will see that changing the cited environment options will result in curves with greater variance in the data.

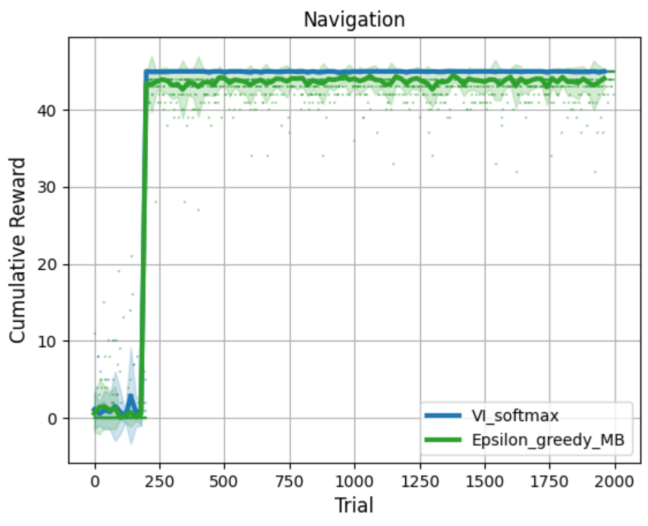


**Figure 4.2:** *Model-Based algorithms comparison. Compared to the $\epsilon$-greedy, the Value Iteration with softmax seems to be the algorithm which leads to the highest reward and the lowest variance.*

Another way to evaluate the performance of an agent in a navigation task can be heat-maps: maps that associate a colour with each state of the system based on the maximum Q-value associated with the state. In Picture 4.3 one can see heat-maps related to all the different goal areas.

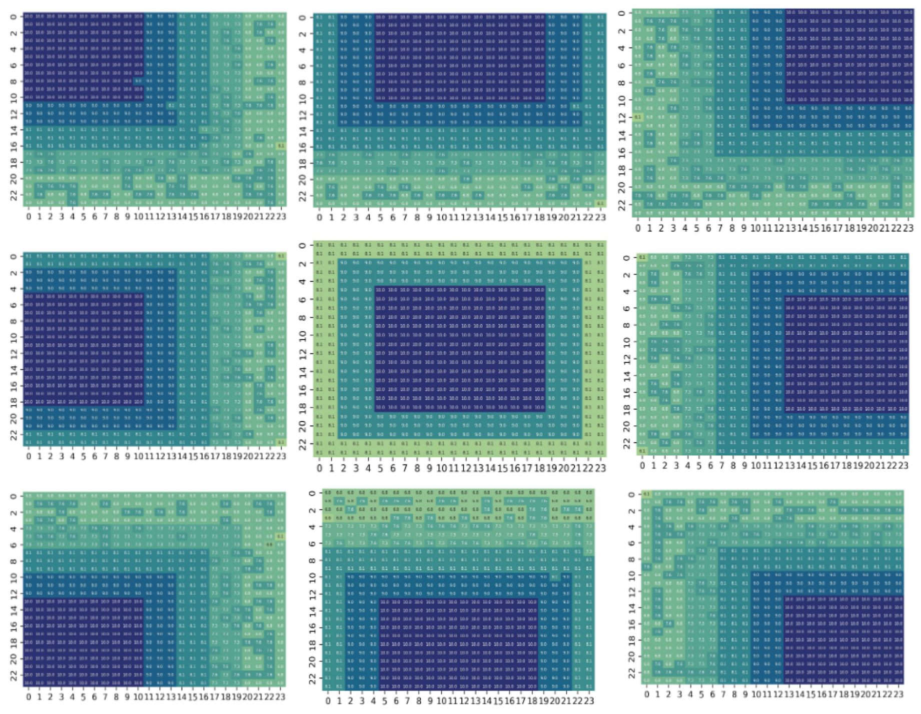
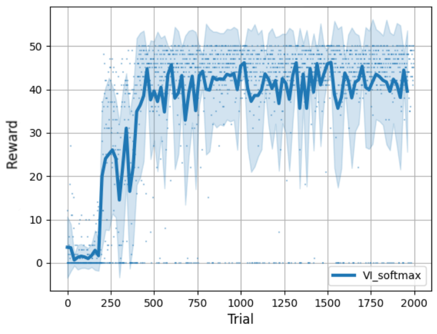


**Figure 4.3:** *Learned Q-values heat maps for the navigation task, related to the different* 9 *goal rooms. The bluer colours are associated with the states of the system in which the agent has learnt to find an higher future expected reward. The agent employed is a Value-Iteration softmax.*

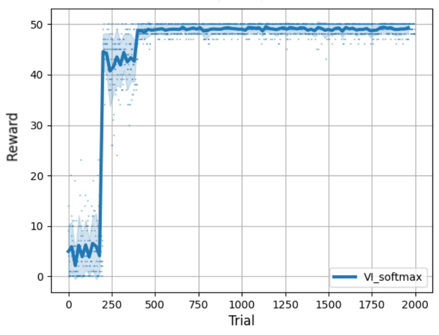
## 4.2 Go to Human

Similarly, the learning rate of the agent performing the *Go to Human* task was also evaluated. Here, an attempt was made to qualitatively study how the variance of the rewards changes according to small changes in the settings associated with new episodes.

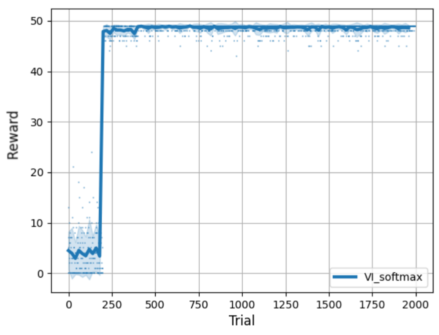
In Picture 4.4 it can be seen how the variance varies when the episodes settings are changed. It is in fact obvious that, by spawning the agent in random locations at each new trial, the agent may feel disoriented (and thus forced to explore the environment). At the same time, by changing the agent's initial state each time, there will naturally be trials in which the agent will be closer to the goal area, and



**(a)**



**(b)**



**(c)**

**Figure 4.4:** *Variance comparison of the reward in the Go to Human task, for different new episode settings.* ***(a)****: at the beginning of each episode, positions and orientations of robot and human are completely random.* ***(b)****: positions fixed, orientations are random.* ***(c)****: positions and orientations are fixed.*

other times in which he will perhaps spawn directly in the goal area! That means, we can naturally expect a great deal of variability in the data here. Nevertheless, it is also possible to make the position of the goal area random as well: another element that for the same reasons increases the variability of the rewards obtained by the agent.

These data suggest that the initial conditions of each episode are correlated with the variance of the reward. In fact, if these conditions are fixed (positions, orientation of the robot and the human), the variance associated with the distribution of rewards is almost not visible. Clearly, once we introduce noise, established by the stochasticity of the positions and/or orientations of the characters, we can see how this variance increases for the above explained reasons.

## 4.3 Social Interaction

Regarding the Social Interaction task, given the large number of state-action pairs, we decided to increase the number of episodes (trials) of the algorithm up to 40000. In this way, the exploited agent has a larger number of trials to learn the one that approaches an optimal policy.

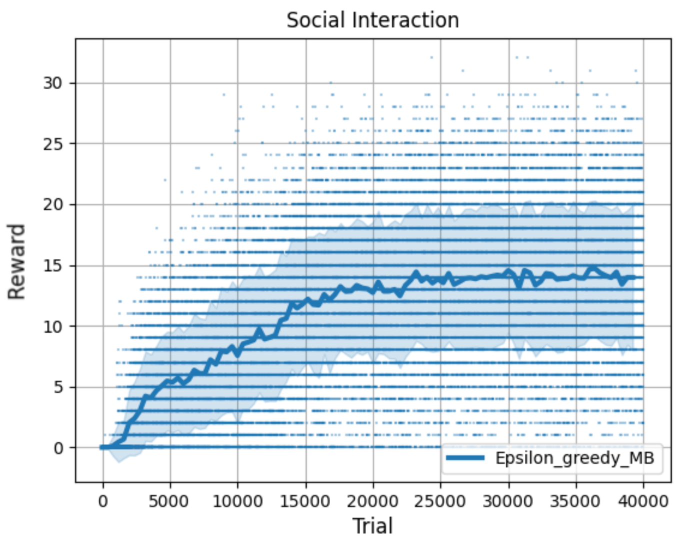


**Figure 4.5:** *Learning performance of $\epsilon$-greedy Model-Based in the Social Interaction task, with easy difficulty human parameters: $s_H = (1, 0, 0)$, $F_H = 0$, $P_H = 0$, $L_H = 0$, $O_H = 0$, $r_H = 0$. The algorithm parameter are exposed in Picture 4.1*

Socially speaking, this model of human is stationary until the maximum level of social interaction is reached, after which it moves slowly in one-step increments and never loses focus. It never randomly changes neither orientation nor direction of movement, which is why in this context the social interaction task reaches near-optimal policy after approximately 30000 trials, with about a 15-point average reward per episode.

Although the reward on the scale may seem low, in reality an average of 15 rewards means that for as many as 15 timesteps contemporarily, the human was moving in the required direction and the level of interaction was maximum. Indeed, it should be remembered that the preceding timesteps are useful for the agent to move in such a way as to go into the human's field of view, and then subsequently enact the social actions useful for raising the social interaction parameter. In this context, an average 15 reward, is an optimal score.

Next, we launched simulations with different classes of humans, which we named *Medium* and *Hard* where respectively these difficulty levels indicate how difficult it is to complete the social interaction. In the *Hard* human case for example, we set the parameter $L_H = 0.5$ and this indicates the fact that in 50% of the cases the human will get distracted, and therefore it will be necessary to start the whole social interaction process again. Clearly, it is extremely difficult to succeed in engaging such a Human, and therefore we expect the learning curves to remain trending toward 0 in this context. In any case, for a view of the parameters for the above classes of humans, one can look at the Table 4.1.

| | $s_H$ | $F_H$ | $P_H$ | $L_H$ | $O_H$ | $r_H$ |
|---|---|---|---|---|---|---|
| **Easy** | $(1, 0, 0)$ | 0 | 0 | 0 | 0 | 0 |
| **Medium** | $(0.5, 0.5, 0)$ | 0.2 | 0.5 | 0.2 | 0.3 | 0.3 |
| **Hard** | $(0, 0, 1)$ | 0.5 | 0.7 | 0.5 | 0.5 | 0.5 |

**Table 4.1:** *Easy, Medium and Hard human parameters. In order: speed, failing rate, pointing need, losing attention, orientation change rate and random rate movement.*

The next step is to compare the learning curves of our agent for the social interaction task with different classes of humans. The result can be observed in Picture 4.6.

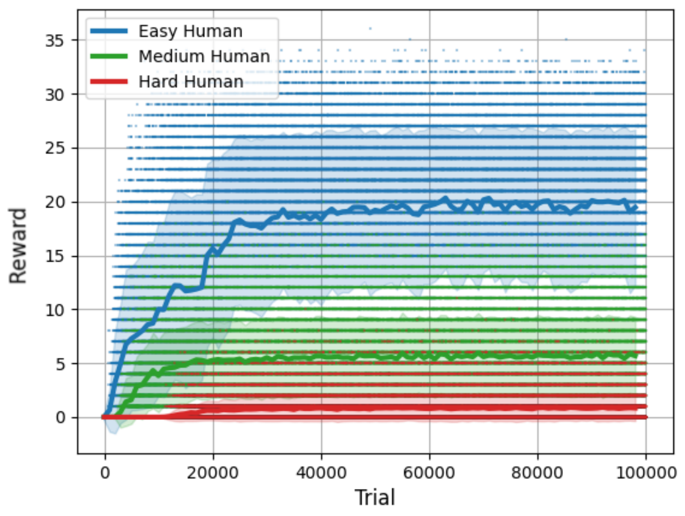
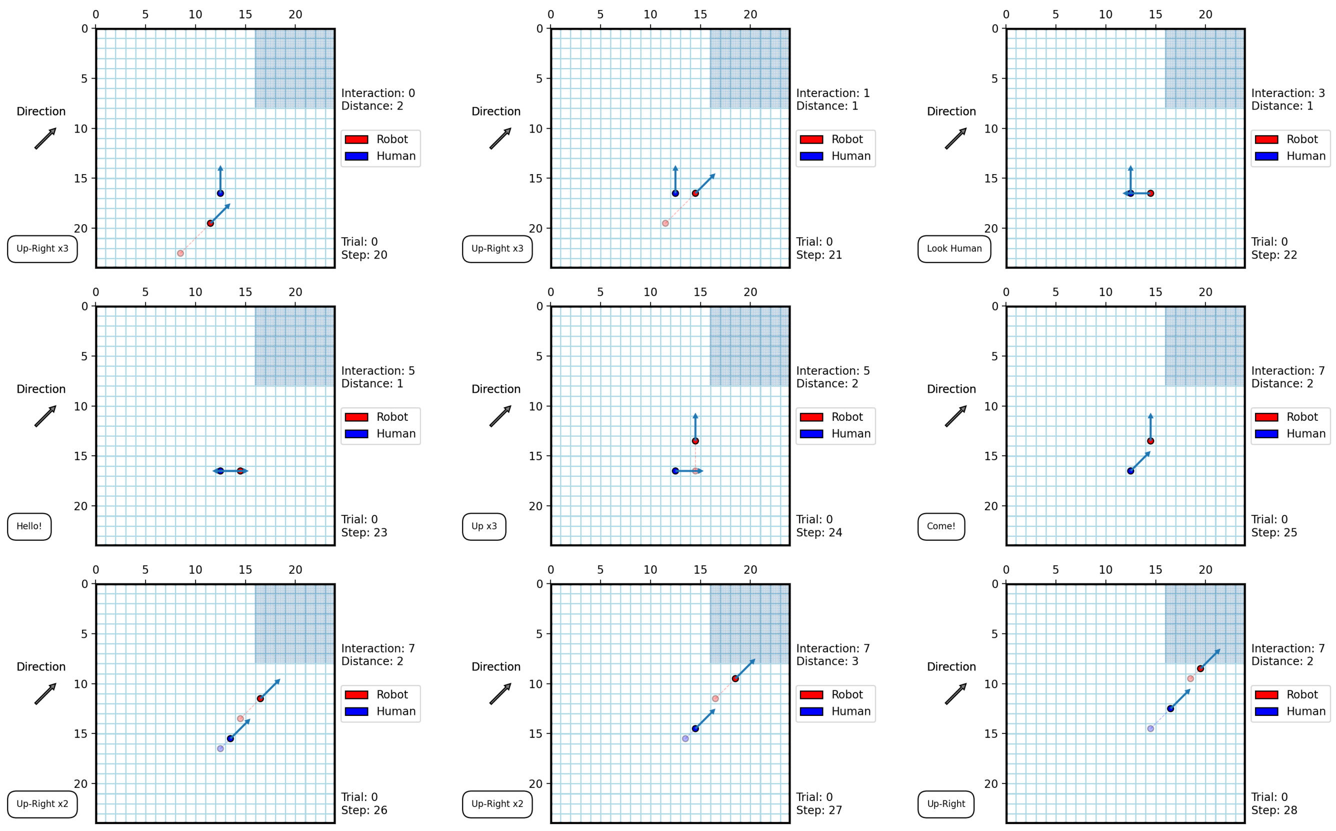


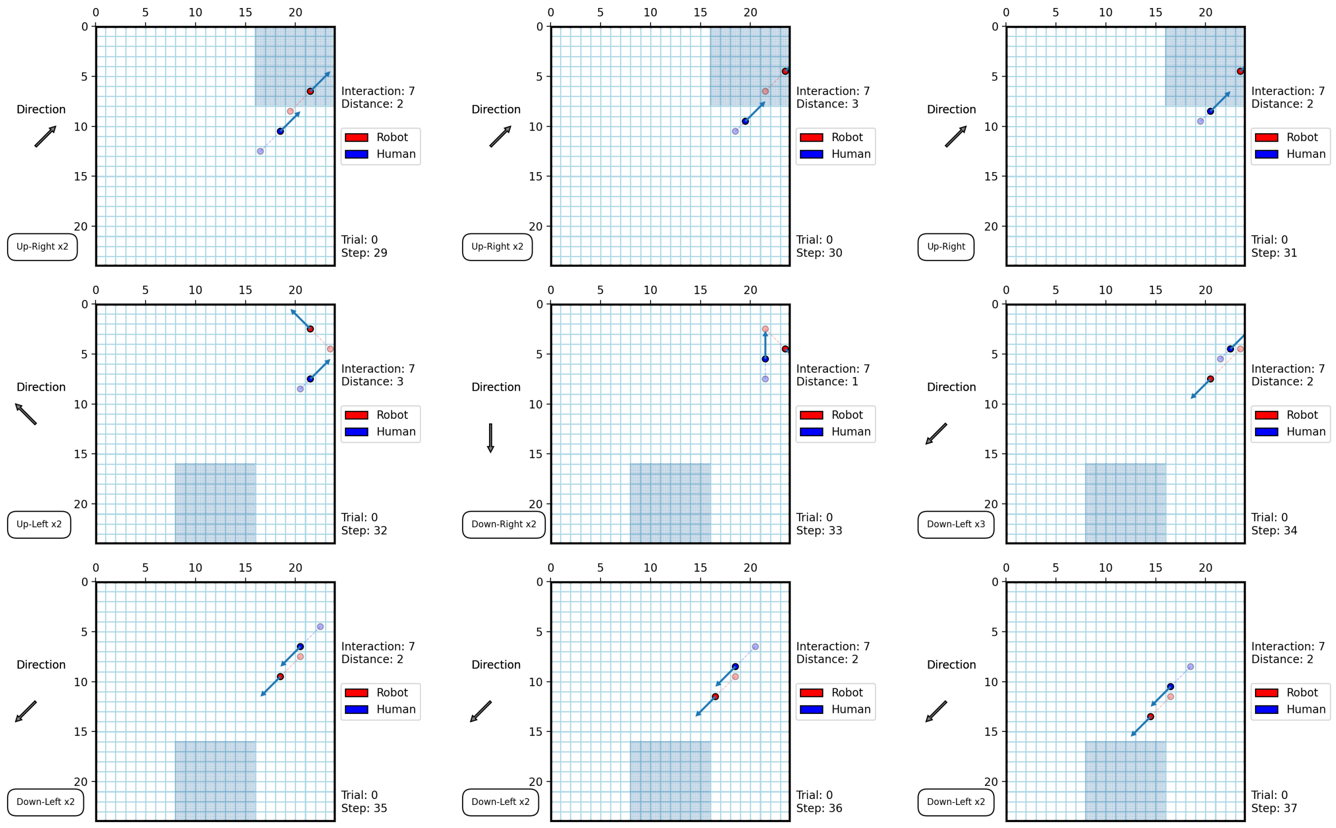
**Figure 4.6:** *Learning performance of the Social Interaction module with different humans, over a number of* 100000 *trials.*

As could be imagined, the best performance is achieved with the easy human, mediocre with the medium human, and very bad with the hard human. In particular, an average of 1 reward per trial is achieved in the latter: out of 40 time step, the agent managed to go in the required direction with the highest level of social interaction only once. However, if we consider the fact that this type of human moves randomly and changes orientation once in two times, distracting himself completely, the result is not even that bad. This type of outcome serves as a valuable tool for assessing the efficacy of coding and modeling within the realm of human class representation. In the event of disparate results, it would indicate potential errors in the code, the architectural framework of the learning process, or the modeling of the human class.

## 4.4 Combined tasks

A proof for evaluating the learning architecture that combine tasks could be the actual dynamics of the agent, i.e., the series of actions performed in different states of the environment.





In these visualizations we can see the results of our learning architecture, in which we are combining the knowledge acquired after the agent has been trained on the three tasks. The robot constantly tries to reach the human in order to go in his facing field; when it succeeds, it start interacting socially with the action `HELLO`. The agent then performs the social action `COME`, making the parameter *Interaction* reaching the maximum value, thus meaning the model of the human considered does not need the action `POINTING`. When this happens, the robot starts to walk in the required direction, leading the human towards the goal area. Once the human touches this area, the rewarded zone changes and moves from the right-top corner of the environment to the bottom-center. The *Interaction* value is still the maximum, and thus the human continues to follow the robot, which is going once again towards the goal.

This example shows that our learning architecture that combines navigation and interaction problems has demonstrated to work in the *Visit the lab* scenario, with excellent results.

### 4.4.1 Human adaptability

One interesting result we propose concerns the adaptability of the agent with respect to different classes of humans. What happens, for example, if we train our agent to interact socially with one class of human, and then test the acquired knowledge with another class of human? To do this, we can simply save the $Q(s, a)$ obtained after a training phase with one human, and then go and exploit it with another class of human.

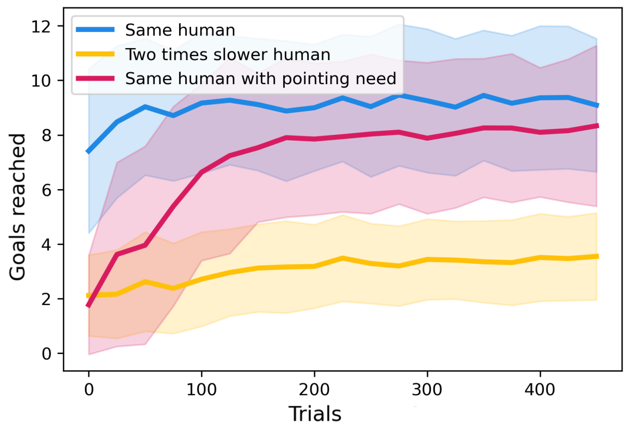


**Figure 4.7:** *Study of the human adaptability with our learning architecture. On the y-axis, the average number of goal areas reached: in blue, exploiting the learned Q-values with the same human, in yellow with two times slower human, and in red the same human but with pointing need action, i.e., $P_H = 1$.*

In Figure 4.7 we can see the adaptability of our agent when it comes into contact with a human with different parameters than those with which it was trained. In particular, the plot shows that our agent succeeds in adapting well when it interfaces with a human that needs the action `POINTING`, but this is not true in the case where it faces a human twice as slow. This can be demonstrated by considering the average number of times the human manages to reach the rewarded area: in the former case we have an average of 8 rewards obtained, while in the latter case the average reaches about 4. As to why the agent has more difficulty adapting to this type of human than to the other is still unclear, but one idea might be as follows.

Let us put ourselves in the agent's shoes: we have been trained to interact socially with a model of a human who needs the actions `LOOKING_H`, `HELLO`, `COME` in order to achieve the highest level of social interaction. We note, however, that by acting in this way, no reward is achieved with the new model of human. The agent thus feels motivated to explore new actions, and can try experimenting with new actions to see if he can gain any reward. As can be seen in the figure, around the 200th trial the agent seems to have figured out the winning strategy for relating to the new human model.

Much more complicated discussion for the twice-slowest human, where clearly the agent has failed to adapt. Indeed, in this context, the agent must learn to move slowly once the level of social interaction reaches the maximum! Otherwise, it will lose engagement and have to start all over again. Among all 25 dynamic actions, only 8 moves at one-step, and so a possible explanation for this phenomenon could be related to the fact that the agent is struggling to understand that it should only focus on these last actions. Furthermore, in this scenario, the agent needs to unlearn actions it has previously acquired, which is more challenging than adding a new action to the sequence.

## 4.5 Experiment

In order to run our algorithms in real-life, a map of the environment containing 41 Markovian states was extrapolated. To derive the transition probability matrix, the robot was tasked with walking the table for approximately 10 hours. Focusing on the probability transition matrix and evaluating its entropy, an interesting phenomenon has been noted:

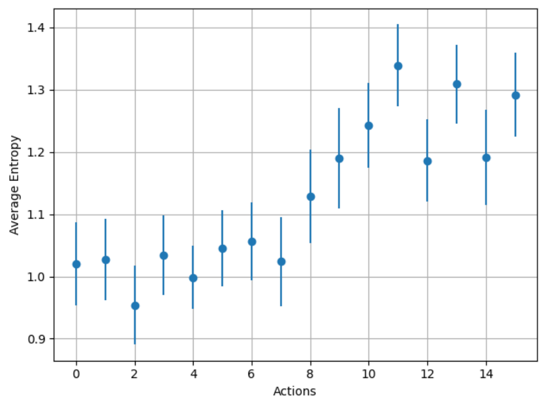


**Figure 4.8:** *Average Shannon entropy of transition probabilities for the available set of actions, over the encoded actions. The values have been computed by using the formula* $H = -\sum (P(s') \log_2 P(s'))$.

Shannon entropy is a measure of the uncertainty or information content in a probability distribution, and it quantifies how disordered the outcomes of a random process are. In this context, we computed the entropy associated with the distribution of successive states of the system $s'_i$ considering each state-action pair $(s_i, a_i)$; we then plotted these values as a function of the actions $a_i$. Recall that in this context, there are 16 actions for the Turtlebot: the first 8 are slow, while the last 8 are fast. We can see that the entropy tends to increase for the fast actions, but this makes perfect sense: a greater distance travelled is naturally associated with a greater number of final states of the system once the pair $(s, a)$ is fixed, which is why obviously the calculated entropy tends to increase. Subsequently, we executed the *Visit the lab* simulation using the experimentally derived probability transition matrix.

Remarkably, within a matter of minutes, we successfully trained a Model-Based agent to navigate the map we generated.

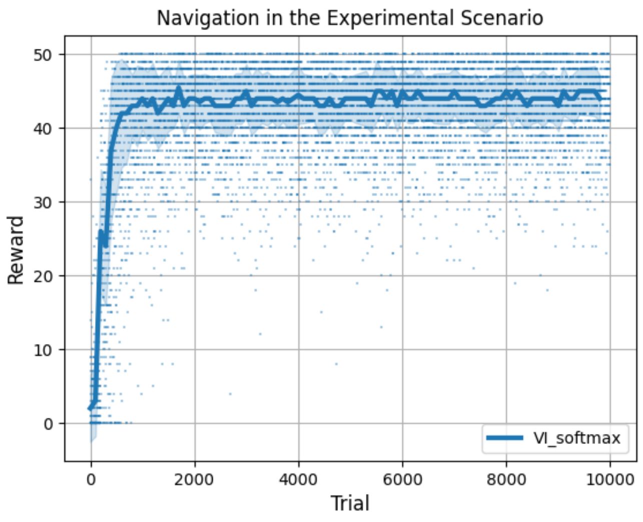


**Figure 4.9:** *Performance of Model-Based agent on the experimental scenario. The structure of the environment is the same, but now the probability transition matrix is completely different since it has been extrapolated from the running Turtlebot3.*

We employed the Value iteration softmax algorithm because of its good performance in the tests that were and, after approximately 2000 episodes, we achieved a good policy that allows for interesting reward results.

After training, we took care to save the Q-tables learnt by the agent so that the knowledge needed to reach the rewarded zones would be available. We then employed the Turtlebot to leverage this acquired knowledge and assessed its performance in terms of cumulative reward over the navigation time.

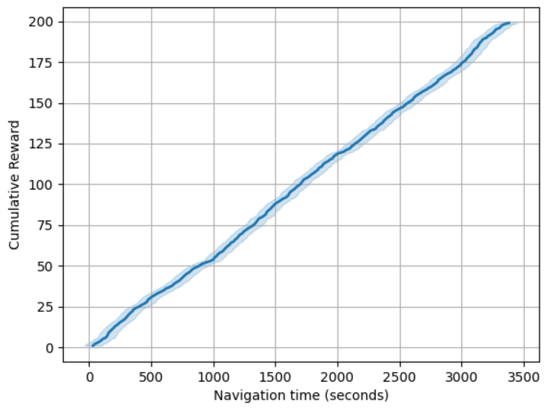


**Figure 4.10:** *Cumulative reward over time of the Turtlebot exploiting knowledge. Displayed datas are referred to the mean of cumulative rewards after three hours of exploitation. Each time the Turtlebot finds a rewarded area, the goal changes and it must find the new rewarded area. The cumulative reward linearly grows over time, since the agent seems it has learnt the quasi-optimal policy.*

Interestingly, in Figure 4.10 the dependency between cumulative reward and navigation time is not perfectly linear, but this variation does not pose an issue. In fact, the points where the first derivative of the function is greater, i.e. where the curve grows faster, are simply associated with situations where once the Turtlebot has found the reward area, the next extracted rewarded area is physically very close to the robot. Consequently, having already been trained, Turtlebot immediately finds the new rewarded area and the obtained curve ends with greater slope in that instant of time.

Again, the steady increase in the cumulative reward shows that excellent results were also achieved experimentally, with the Turtlebot solving the navigation problem perfectly. Due to time constraints, it was not possible to also implement the social interaction module, but in the continuation of the project this will certainly be done.

In addition, a dynamic image exposed via a projector was also developed, inherent to the Markovian states of the system with an associated heat-map of Q-values. This type of tool is useful for both debugging and various demonstrations to be performed in the laboratory. A photo of the scenario can be seen in Figure 4.11.

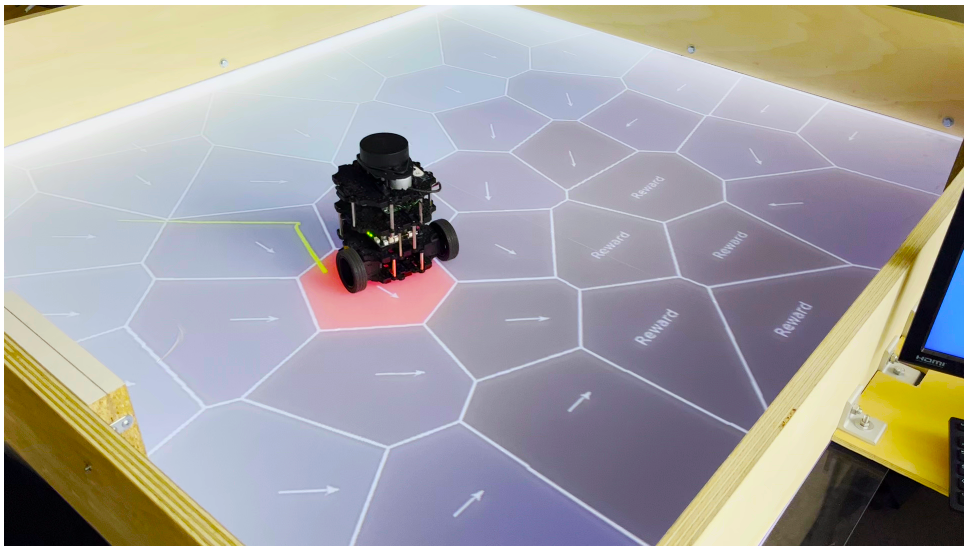


**Figure 4.11:** *Photo of the Turtlebot over the designed heat-map. Each color of the state is associated with the amount of reward it is possible to get arriving in the state. Arrows are given by* $\text{argmax}_a Q(s,a)$*, i.e., the action that brings to the maximum expected reward in that state. Red state is given by the localization found with the SLAM algorithm. Yellow line represent the trajectory of the robot.*

Chapter 5

# Conclusions

The first Chapter of this thesis focused on reinforcement learning theory, formalizing MDPs and the most important algorithms that were used in this thesis project. Model-Based and Model-Free algorithms were addressed, function approximations method were rapidly discussed, and the theory of HRL was briefly mentioned.

Within the second Chapter we focused on the ELSA Project. The main research rationale was discussed, with great attention given to the purpose of the project and the methodologies that will be used. Nonetheless, we focused on ISIR's contribution to the ELSA Project, namely the *Visit the lab* framework: a scenario in which a human has to take an informational tour inside a lab by seeing different rooms, accompanied by a robot. This robot will obviously not be programmed; rather, it will have to learn both how to navigate properly, and how to interact socially with the human. MDPs associated with the three implemented modules were formalized: *Navigation*, *Social interaction*, *and Go to human*, explaining the definition of the states, actions, rewards attributed to the system. In this context, our aim was to build and implement a useful architecture to bring these different modules together, so as to naturally unify the knowledge gained after performing offline RL algorithms on the modules.

In third Chapter, the materials and methods used were addressed: we focused on the learning architecture, and the experimental architecture used to launch an experiment with Turtlebot. It was explained how to derive an environment map, and the general architecture of the experiment was explained, using ROS. Nonetheless, it was explained how to derive the matrix of probability transitions using a robot.

Finally, the fourth Chapter showed the experimental results that were conducted both through simulations and on experimentally collected real-life data. The results that were shown demonstrated great successes regarding the learning architecture presented, and excellent results regarding the experiment with the Turtlebot. In this last part, the focus was only on the navigation problem, but in the short future it is thought feasible to address the social interaction problem as well.

In conclusion, this thesis has delved into the realm of human-robot interaction, harnessing the power of reinforcement learning to empower autonomous agents with the ability to navigate and manage social situations effectively. Through a designed architecture, we aimed to create agents capable of seamless interactions with humans, capturing their attention and guiding them to specific goal areas.

Our evaluation encompassed simulation trials involving simulated humans with varying characteristics, studying also the the agent's adaptability in different scenarios. Additionally, we ventured into a real-world experiment employing the Turtlebot robot, emphasizing navigation in a table scenario.

The results achieved are promising, as they clearly validate the efficacy of our learning architecture in seamlessly integrating knowledge across diverse tasks. The successful navigation problem tackled with the Turtlebot further underscores the potential of this approach in real-world applications.

In summary, this thesis represents a substantial contribution to the rapidly evolving domain of human-robot interactions. By harnessing RL techniques, we have empowered agents with the capability to comprehend and actively participate in social scenarios, marking a substantial advancement towards more sophisticated and intuitive human-robot collaborations.

## Future directions

Because the project is only at the beginning of the journey, there is still plenty of time to continue this work to the best of our ability. The next stages are already well outlined: it is first important to have the Turtlebot deal with a social navigation problem, possibly with a human drawn with the interactive screen, or interacting with another Turtlebot. Then, it will necessary to introdue RL functions approximation methods, so that it will be possible to build more general models of agents policies. The next step will regard the actual, real, human-robot interaction with a human-size robot such as PR2 or TIAGo; it will not be trivial but for sure it is possible to make a robot learn how to communicate and interact with a human.

In essence, this project's trajectory is marked by a series of well-defined stages, each designed to propel us towards a more comprehensive understanding of social affordances in human-robot interaction. With time on our side, we are poised to explore the depths of robotics and reinforcement learning research and contribute substantially to the field's evolution.

# Appendix A

# Manhattan Distance

The Manhattan distance, also known as the taxicab distance, is a metric used to measure the distance between two points in a grid-based space, such as the environments that have been developed during this thesis work. It is named after the layout of streets in Manhattan, where the shortest path between two points follows a grid-like pattern, similar to how a taxi might navigate through city streets (Krause 1973).

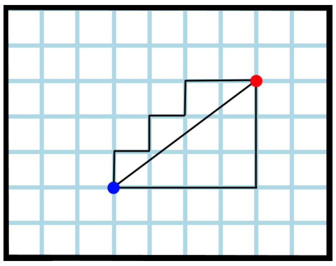


**Figure A.1:** *Example of two Manhattan distances with respect to the diagonal line provided by Euclidean distance between the blue and the red point in the Gridworld environment.*

More generally, known two n-dimensional vectors $\vec{x}$ and $\vec{y}$, the way to compute the Manhattan distance is straightforward:

$$d_M(\vec{x}, \vec{y}) = \|\vec{x} - \vec{y}\|_M = \sum_{i=1}^{n} |x_i - y_i| \tag{A.1}$$

whereas if we apply this formula on the above picture, the taxicab distance between the blue and the red point would be computed simply as $d_M = |x_1 - y_1| + |x_2 - y_2|$, where the considered vectors represent the coordinate of the points in an arbitrary coordinates system.

For its simplicity, the Manhattan distance has various applications in fields like computer science and data analysis, such as:

- **Grid-Based Navigation**: In robotics and computer games, the Manhattan distance is used to determine the shortest path between two points on a grid, where movement is restricted to horizontal and vertical steps.
- **Clustering Algorithms**: The Manhattan distance is used in clustering algorithms, such as k-means, to measure dissimilarity between data points (Sinwar and Kaushik 2014).

# Appendix B

# LIDAR sensor

LIDAR (Laser Imaging, Detection, and Ranging) is a remote sensing technology that uses laser light to measure distances and create detailed three-dimensional maps of objects and environments (Taylor 2019). It can provide accurate distance measurements for creating maps and enabling navigation for robots and autonomous systems: equipping a moving robot with such a sensor allows it to avoid obstacles in the scenario by continuously scanning its surroundings at 360°.

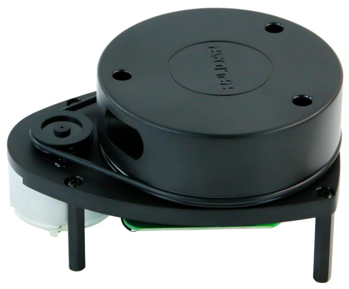


**Figure B.1:** *Picture of the RPLIDAR A1M8 Laser Scanner Sensor that was equipped to the TurtleBot.*

Usually, the functioning of a LIDAR sensor starts with the pulses laser emission in different directions. These pulses are typically in the infrared range with a wavelength $\lambda$ included between 700 $nm$ and 1 $mm$, thus not not visible to the human eye. If an obstacle has been encountered, the laser pulse will hits an object that will reflects the light back towards the sensor.

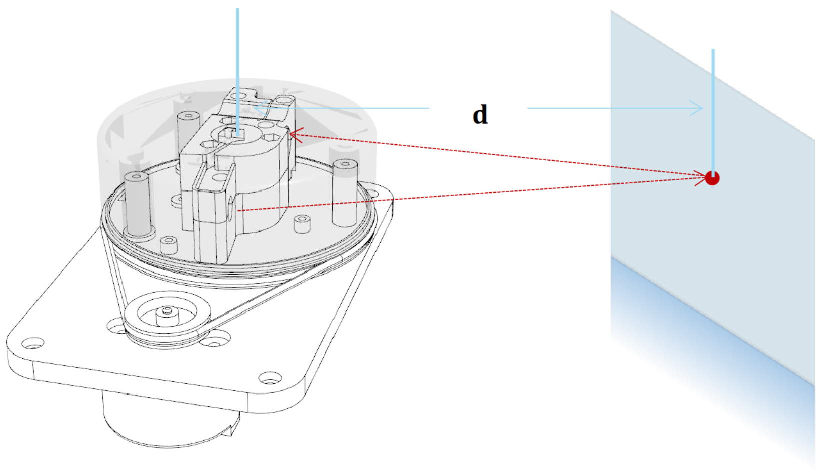


**Figure B.2:** *The RPLIDAR A1 Working Schematic. Credit: [Shanghai Slamtec Co., Ltd.]*

Considering the figure B.3, let $(x_1, y_1)$ be the 2-D coordinates of the point from which the laser has been emitted from the LIDAR, and $(x_2, y_2)$ the coordinates in which the light has been then received. The red point in the wall has coordinates $(x_3, y_3)$. We then have the formation of a triangle connecting these three coordinates points, of which we can define two angles:

- $\theta_1$, the angle associated with the vertex of the triangle receiving the laser;
- considering the angle at the vertex emitting the laser, $\theta_2$ is its supplementary.

In this framework, the goal is to find the coordinate of $(x_3, y_3)$, that using simple trigonometry are given by:

$$x_3 = \frac{(y_1 - y_2) + x_2 \tan(\theta_2) - x_1 tan(\theta_1)}{\tan(\theta_2) - \tan(\theta_1)}$$

$$y_3 = \frac{(y_1 \tan(\theta_2) - y_2 \tan(\theta_1)) + (x_2 - x_1) \tan(\theta_2) \tan(\theta_1)}{\tan(\theta_2) - \tan(\theta_1)}$$

Given that, the sensor repeats this process for multiple angles as it rotates, generating a set of distance measurements. These measurements are typically organized into a point cloud or another suitable format that represents the environment in 2D. Finally, the system's software processes the LIDAR data to identify objects, obstacles, and open spaces, enabling the robot or vehicle to make informed decisions about its path. Running the Turtlebot in our Gridworld experimental environment with the equipped LIDAR sensor, from the RViz tool the result is the following:

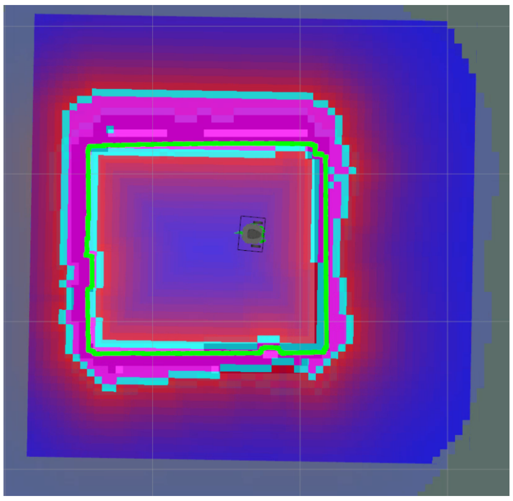


**Figure B.3:** *LIDAR measured distance on RViz during our experiments with the Turtlebot. The distance measured from the scanning sensor is the light-blue line in the inner square in which the Turtlebot is located.*

Appendix C

# IMOL 2023 Poster

Part of the work of this thesis project was exhibited at the 6th International Workshop on Intrinsically Motivated Open-ended Learning (IMOL) that took place on the 13rd-15th of September 2023 in Paris, France (IMOL 2023).

Within this poster, we focused on a specific architecture that was not explored in this thesis project: the passive Model-Free learner. The idea is that, while using a Model-Based algorithm that learns by going to build a model of transitions and rewards, we can go alongside it a Model-Free learner that learns in turn. To do this, in the learning process, all we need to do is to update two different Q-tables: one regarding the Model-Free, and one regarding the Model-Based. The whole process at any rate is orchestrated by the Model-Based agent, since this one chooses the action to be performed. Model-Free agents have the advantage of having a computational inference cost that is practically zero, but it requires many more trials to learn the optimal policy. In this framework, we show that a passive Model-Free observer learning from a Model-Based agent can bootstrap the Model-Free performance.

A complete view of the exhibited poster can be seen on the next page.

# Visit the lab scenario: navigation and social interaction in reinforcement learning

Augustin Chartouny [1] Keivan Amini [1] Mehdi Khamassi [1] Benoît Girard [1]

[1]Institut des Systèmes Intelligents et de Robotique, Sorbonne Université & CNRS, Paris

## Presentation of the task

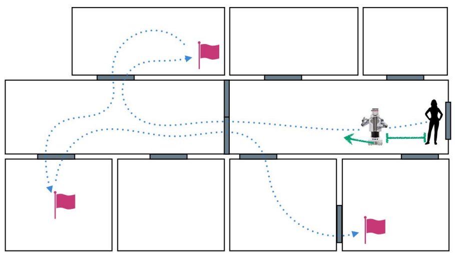


Figure 1. The goal of the robot is to learn how to bring different humans to several rooms.

## Research questions

- How should the robot learn the effects of its actions both in terms of navigation (moving from one room to another) and in terms of social interaction (attracting and keeping the human's attention)?
- How can the robot adapt to intra and inter-human behavioral variability?

## Our current approaches

We use model-based reinforcement learning with the agent learning separate models of navigation and of social interaction tasks. The navigation model focuses on finding the path to the goal room while the social system tries to get and keep the human's attention. Between the two systems, the agent learns an option to navigate to the human's visual field so as to get its attention.

### Tabular simulations

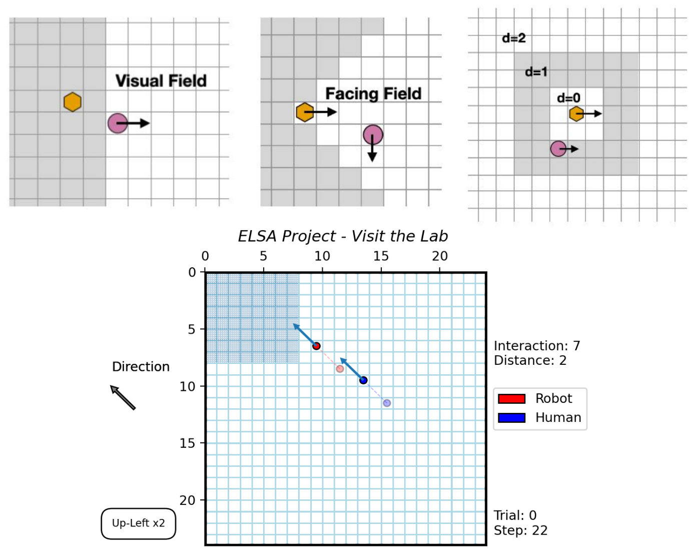


Figure 2. Simulated tabular version of the task. (Top) Social inputs for the agent to take into account during social interaction (visual field and facing field of the robot and of the human, human-robot distance). (Bottom) The robot (in red) tries to bring the human (in blue) to the checked goal area. The agent chooses between 24 actions to navigate (8 directions, 3 speeds) and a few actions for social interactions such as hello, come and indicating a direction.

### Real-world experiments

We started to run real-world experiments, with a TurtleBot3 navigating in a closed arena and a simulated human, second step before full Human-Robot Interaction.

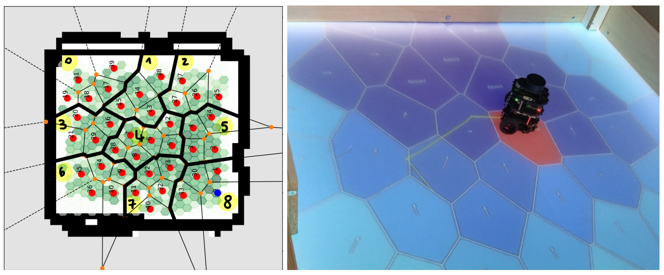


Figure 3. Navigation system tested on a real robot. (Left) Map of the environment, built by the robot using SLAM (left). (Right) Robot finding the rewarded areas, with a gradient of colors from dark blue (high) to white (low) representing the learned Q-values. This new environment brings more uncertainty on the outcome of the robot actions.

## Influence of MB exploration on a passive MF learner

The structure of the social task, with the robot having to make a sequence of actions to get human's attention makes it hard for a model-free agent to learn a good policy. In Dromnelle et al.'s [2] MF/MB arbitration method, the MB agent often takes the lead in the beginning. Thus, we study a passive MF observer learning from an MB agent and show that it can bootstrap the MF agent performance.

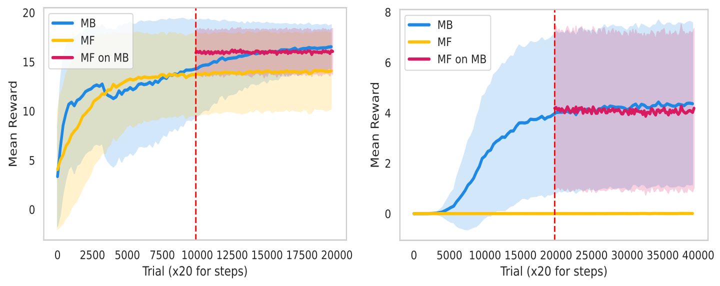


Figure 4. Comparison of performance on 10 runs of three $\epsilon$-greedy agents: MB, MF and MF which passively observes the MB for half of the experiment. (Left) Navigation task. (Right) Social task.

Nevertheless, the exploration method of the MB agent can drastically modify what the MF agent perceives and vice versa, which we illustrate in the navigation task.

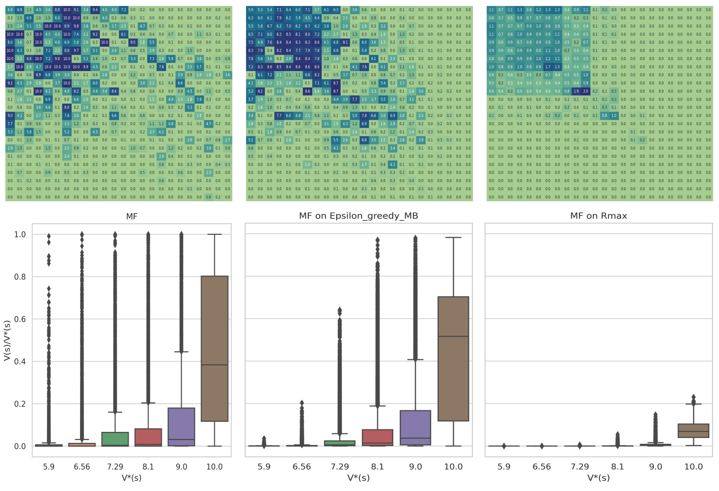


Figure 5. (Top) Q-map learned by MF agents in the navigation task. From left to right, MF alone, MF observing MB $\epsilon$-greedy, MF observing R-max [2]. (Bottom) Learned Q-values of the same three agents, on 10 runs and for all the goal locations, depending on possible values $V^*(s)$.

## Adaptability to different human behaviors

The final task of the robot is to lead the human to several rooms in a row. After learning a model of the environment with R-max and a model of one human with $\epsilon$-greedy, we compare how the agent adapts to the new human it interacts with.

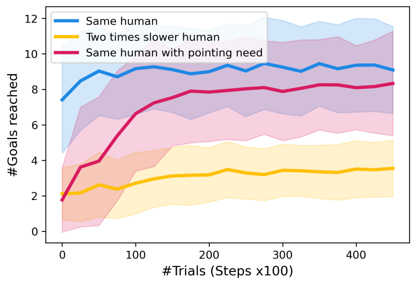


Figure 6. The agent adapts well to the introduction of a new necessary action to engage the human, but adapts poorly to a big change in the human speed. Future models could include intrinsically motivated model-based agents biased towards the reduction of the uncertainty on their models.

## Future directions

- Further adapt the task to real-world environments and to continuous domains.
- Explore how the robot should arbitrate between learning about the physical environment it is exploring or about the model of the human it is interacting with and how to react in the case of non-stationarity.
- Explore the idea of a human-general model and human-specific models when the robot faces unpredictable behaviors.

[1] R. Dromnelle, ..., B. Girard, M. Khamassi (2020). Conference on Biomimetic and Biohybrid Systems
[2] Brafman, R. I., & Tennenholtz, M. (2002). Journal of Machine Learning Research

Work funded by ANR ELSA IMOL 2023 contact: augustin.chartouny@sorbonne-universite.fr

# Bibliography

[1] Josep Aulinas et al. “The SLAM problem: a survey”. In: *Artificial Intelligence Research and Development* (2008), pp. 363–371.

[2] Franz Aurenhammer and Rolf Klein. “Voronoi Diagrams.” In: *Handbook of computational geometry* 5.10 (2000), pp. 201–290.

[3] Andrew G Barto and Sridhar Mahadevan. “Recent advances in hierarchical reinforcement learning”. In: *Discrete event dynamic systems* 13.1-2 (2003), pp. 41–77.

[4] Dictionary Cambridge. “Cambridge advanced learner’s dictionary”. In: *Recuperado de: https://dictionary. cambridge. org/es/diccionario/ingles/blended-learning* (2008).

[5] Arthur Charpentier, Romuald Elie, and Carl Remlinger. “Reinforcement learning in economics and finance”. In: *Computational Economics* (2021), pp. 1–38.

[6] Thomas G Dietterich. “Hierarchical reinforcement learning with the MAXQ value function decomposition”. In: *Journal of artificial intelligence research* 13 (2000), pp. 227–303.

[7] Hugh Durrant-Whyte and Tim Bailey. “Simultaneous localization and mapping: part I”. In: *IEEE robotics & automation magazine* 13.2 (2006), pp. 99–110.

[8] ELSA. *Effective Learning of Social Affordances project - web site*. 2023. URL: https://www.isir.upmc.fr/projects/elsa/?lang=en.

[9] James J Gibson. *The ecological approach to visual perception: classic edition*. Psychology press, 1979.

[10] Stevan Harnad. “The symbol grounding problem”. In: *Physica D: Nonlinear Phenomena* 42.1-3 (1990), pp. 335–346.

[11] IMOL. *International Workshop on Intrinsically Motivated Open-ended Learning - web site*. 2023. URL: https://imolconf2023.github.io/.

[12] Gareth James et al. *An introduction to statistical learning*. Vol. 112. Springer, 2013.

[13] B Ravi Kiran et al. “Deep reinforcement learning for autonomous driving: A survey”. In: *IEEE Transactions on Intelligent Transportation Systems* 23.6 (2021), pp. 4909–4926.

[14] Anis Koubâa et al. *Robot Operating System (ROS)*. Vol. 1. Springer, 2017.

[15] Eugene F Krause. “Taxicab geometry”. In: *The Mathematics Teacher* 66.8 (1973), pp. 695–706.

[16] Steven Macenski et al. “Robot Operating System 2: Design, architecture, and uses in the wild”. In: *Science Robotics* 7.66 (2022), eabm6074. DOI: 10.1126/scirobotics.abm6074. URL: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

[17] Elisa Massi et al. “Model-Based and Model-Free Replay Mechanisms for Reinforcement Learning in Neurorobotics”. In: *Frontiers in Neurorobotics* 16 (2022), p. 864380.

[18] Volodymyr Mnih et al. “Playing atari with deep reinforcement learning”. In: *arXiv preprint arXiv:1312.5602* (2013).

[19] Robotis. *Robotis website e-Manual*. 2023. URL: https://emanual.robotis.com/docs/en/platform/turtlebot3/features/.

[20] Arthur L Samuel. “Some studies in machine learning using the game of checkers”. In: *IBM Journal of research and development* 3.3 (1959), pp. 210–229.

[21] Deepak Sinwar and Rahul Kaushik. “Study of Euclidean and Manhattan distance metrics using simple k-means clustering”. In: *Int. J. Res. Appl. Sci. Eng. Technol* 2.5 (2014), pp. 270–274.

[22] Richard S Sutton. “Learning to predict by the methods of temporal differences”. In: *Machine learning* 3 (1988), pp. 9–44.

[23] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[24] Richard S Sutton, Doina Precup, and Satinder Singh. “Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning”. In: *Artificial intelligence* 112.1-2 (1999), pp. 181–211.

[25] Travis S Taylor. *Introduction to laser science and engineering*. CRC Press, 2019.

[26] Sebastian Thrun. “Probabilistic robotics”. In: *Communications of the ACM* 45.3 (2002), pp. 52–57.

[27] Sorbonne Universitè. *Institut des Systèmes Intelligents et de Robotique - web site*. 2022. URL: https://www.isir.upmc.fr/isir/.

[28] Christopher John Cornish Hellaby Watkins. “Learning from delayed rewards”. In: (1989).

[29] Martyna Weigl et al. “Grid-based mapping for autonomous mobile robot”. In: *Robotics and Autonomous Systems* 11.1 (1993), pp. 13–21.

[30] Chao Yu et al. “Reinforcement learning in healthcare: A survey”. In: *ACM Computing Surveys (CSUR)* 55.1 (2021), pp. 1–36.

[31] Yu Zhang and Qiang Yang. “A survey on multi-task learning”. In: *IEEE Transactions on Knowledge and Data Engineering* 34.12 (2021), pp. 5586–5609.