People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research

IBN KHALDOUN UNIVERSITY OF TIARET

Dissertation

Presented to:

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
DEPARTEMENT OF MATHEMATICS

in order to obtain the degree of:

MASTER

Specialty: Computer Engineering

Presented by:
Belahouel Ossama et Hamri Akram

On the theme:

Reinforcement learning and Path Planning for the navigation of Mobile Robots

Defended publicly on / /2024 in Tiaret in front the jury composed of: :

Mr MEBAREK Bendaoud	Professor	Tiaret University	Supervisor
Mr MAASAKRI Mustapha	MCA	Tiaret University	Chairman
Mr MOSTEFAOUI Kadda	MCB	Tiaret University	Examine

Remerciements

With deep pride and gratitude, we extend our sincere thanks to everyone who supported and encouraged us throughout our academic and research journey.

First and foremost, we thank ***Allah Almighty****, whose blessings and guidance made this work possible.*

Our heartfelt thanks go to our beloved families, whose unwavering moral and financial support has been a cornerstone of our progress.

We would also like to thank our classmates and friends for their continued encouragement, collaboration, and motivation.

To all of you, we dedicate this modest work as a token of our appreciation and respect

Belahouel Ossama

Remerciements

All praise is due to Allah, through whose grace and guidance this humble work has been completed.

We would like to express our deepest gratitude to all those who contributed to the success of this project, whether directly or indirectly.

Our heartfelt gratitude goes to our beloved families, especially our parents, for their endless support, encouragement, and patience.
We also thank our friends and colleagues for their assistance, motivation, and positive presence throughout this journey.

May Allah reward you all, and may He grant success and goodness to everyone.

Hamri Akram

Table of contents

List of figures:

List of tables:

List of abbreviations:

AI: *Artificial Intelligence*

ML*: Machine learning*

DL*: Deep Learning*

RL*: Reinforcement Learning*

DRL*: Deep Reinforcement Learning*

ANN*: Artificial Neural Networks*

CNN*: Convolutional Neural Networks*

MDP*: Markov Decision Process*

POMDP*: Partially Observable Markov Decision Process*

DP: *Dynamic Programming*

TD*: Temporal Difference*

DQN*: Deep Q-Network*

Abstract

Wheeled mobile robots, which are present in several fields of activities nowadays, are machines equipped with perception, reasoning, and action capabilities to navigate autonomously and safely in their environments. This autonomous navigation skill requires a combination of hardware and software resources to perform basic tasks such as path planning, obstacle avoidance, and motion control with respect of the current navigation situation

Reinforcement learning is one of the intelligent methods adopted to address the challenges of autonomous navigation in dynamic environments. It is a technique based on the interaction between an agent whose goal is to learn an action policy and its environment.

In this thesis, this work focuses on integrating artificial intelligence, especially the Q-Learning algorithm, into the field of mobile robotics in order to enable robots to make intelligent decisions while on the move in environments with obstacles. The goal is to improve the autonomy and adaptability of robots by learning from experience, without the need for explicit programming for each task

Keywords: Mobile *robotics, Path planning, Obstacle avoidance , Reinforcement learning, Autonomous navigation , Dynamic environments , Artificial intelligence , Q learning algorithm*

ملخص

الروبوتات المتنقلة ذات العجلات ، الموجودة في العديد من مجالات الأنشطة في الوقت الحاضر ، هي آلات مجهزة بقدرات الإدراك والتفكير والعمل للتنقل بشكل مستقل وآمن في بيئاتها. تتطلب مهارة التنقل المستقل هذه مزيجا من موارد الأجهزة والبرامج لأداء المهام الأساسية مثل تخطيط المسار وتجنب العوائق والتحكم في الحركة فيما يتعلق بحالة التنقل الحالية

التعلم المعزز هو أحد الأساليب الذكية المعتمدة لمواجهة تحديات الملاحة المستقلة في البيئات الديناميكية. إنها تقنية تعتمد على التفاعل بين الوكيل الذي يهدف إلى تعلم سياسة العمل وبيئتها.

في هذه الأطروحة ، يركز هذا العمل على دمج الذكاء الاصطناعي ، وخاصة خوارزمية كيو التعلم ، في مجال الروبوتات المتنقلة من أجل تمكين الروبوتات من اتخاذ قرارات ذكية أثناء التنقل في البيئات ذات العقبات. الهدف هو تحسين استقلالية الروبوتات وقدرتها على التكيف من خلال التعلم من التجربة ، دون الحاجة إلى برمجة صريحة لكل مهمة

الكلمات المفتاحية : الروبوتات المتنقلة ,تخطيط المسار , تجنب العوائق , التعلم المعزز , الملاحة المستقلة , البيئات الديناميكية , الذكاء الاصطناعي , خوارزمية Q-Learning

General Introduction

General Introduction

Artificial intelligence (AI) occupies a prominent place in current technological development, in particular in complex fields such as autonomous robotics. Among the most promising approaches to AI, Reinforcement Learning (RL) is emerging as a powerful method to allow artificial agents to learn to interact with their environment autonomously, based solely on the experience gained. Unlike supervised learning, which requires labeled data, reinforcement learning is based on the concept of trial-and-error learning. An agent learns to make decisions by receiving rewards (or penalties) based on his actions, in order to maximize a long-term cumulative reward.

In the context of mobile robotics, this approach makes it possible to design robots capable of learning complex behaviors without being explicitly programmed for each task. Thanks to reinforcement learning, robots can adapt to changing environments, improve their autonomy, and solve problems based on their own experiences. This ability to adapt is particularly useful in areas such as autonomous navigation, object manipulation, exploration of unknown environments or human-computer interaction.

Among the best known and most widely used algorithms in the field of reinforcement learning, Q-Learning stands out for its simplicity and efficiency. This is a model-free algorithm that is based on learning a function called the Q function. This function evaluates the quality of a given action in a given state, thus allowing the agent to select the optimal actions to maximize the reward. Q-Learning updates the value of Q iteratively, according to the famous Bellman equation, integrating both the immediate reward and the expected future value. This method has proven its effectiveness in discrete and deterministic environments, and constitutes a solid basis for more complex extensions such as Deep Q-Learning, which combines Q-Learning with deep neural networks.

However, the application of Q-Learning in robotics raises several challenges. Real robotic environments are often continuous, noisy, and partially observable, which complicates learning. In addition, the space of states and actions can be very vast, making exhaustive exploration difficult. Despite these limitations, numerous researches have shown that Q-Learning can be adapted and improved to meet the requirements of robotics. Techniques such as state

discretization, guided exploration, or the use of approximation functions make it possible to overcome some of these constraints.

The growing interest in Q-Learning applied to robotics is also motivated by the ease of implementation of the algorithm on embedded hardware with low computing power. It is particularly well adapted to simulation scenarios, but more and more hybrid approaches today make it possible to transfer the skills acquired in simulation to real physical environments. This paves the way for concrete applications, ranging from domestic robots to industrial robots, including autonomous drones and intelligent vehicles.

The present thesis is part of this promising context. It aims to explore, design and evaluate reinforcement learning strategies for autonomous robots using the Q-Learning algorithm. The objective is to demonstrate that this algorithm, despite its theoretical simplicity, can allow a robot to learn effective and robust behaviors in complex environments. We will focus on modeling problems in order to optimize learning performance, improve convergence speed, and extend the capabilities of Q-Learning to more realistic robotic tasks. In short, reinforcement learning, and more particularly Q-Learning, represents an innovative and relevant approach to equip robots with adaptive decision-making capabilities. This thesis aims to contribute to the advancement of research in this field, by combining theoretical rigor and practical experimentation, in order to open new perspectives towards more intelligent, autonomous and efficient robots.

To achieve this objective, the thesis is structured as follows:

- The first chapter provides a general overview of mobile robotics through a series of definitions, its historical development, and the main architectures used.
- The second chapter is dedicated to machine learning techniques, detailing how they work as well as the most common algorithms.
- The third chapter focuses specifically on reinforcement learning, with particular emphasis on the Q-learning algorithm.
- The fourth and final chapter presents the proposed approach, along with the implementation and analysis of the results obtained.

CHAPTER I

OVERVIEW OF MOBILE ROBOTICS

1 Introduction

Mobile robotics is a dynamic field focused on designing, building, and programming robots capable of moving and operating autonomously or semi-autonomously in various environments. These robots are equipped with sensors, actuators, and advanced algorithms to perceive their surroundings, navigate obstacles, and perform tasks. Applications range from industrial automation and warehouse logistics to healthcare, agriculture, and space exploration. Key technologies include localization, mapping, path planning, and machine learning for decision-making. Mobile robotics integrates principles from mechanical engineering, computer science, and artificial intelligence. Its ongoing advancements are driving innovation, improving efficiency, and enabling robots to tackle complex real-world challenges.

The word ‘robot’ was first introduced in the real world in 1920 through the play ‘Rossum’s Universal Robots’ written by the Czech Karel Capek [1]. Robots become intelligent and autonomous after the implications of computer software and cybernetics in the field of robotics science [2].

2 History Mobile Robotics

Artificial intelligence and mobile robotics have always been interconnected. Even before the 1956 Dartmouth College Conference, where the term "artificial intelligence" was coined, it was recognized that mobile robots could perform interesting tasks and learn. William Grey Walter built a pair of mobile robots in the early 1950s that were capable of learning tasks such as obstacle avoidance and phototaxis through instrumental conditioning, by altering charges in a robot's capacitor, which controlled its behavior Early pioneers in artificial intelligence, such as researchers, became interested in robotics almost immediately after the 1956 Dartmouth Conference. In the late 1950s, Minsky, together with researchers, attempted to build a ping-pong-playing robot. Due to technical difficulties with the hardware, they eventually built a robot that could catch a ball using a basket instead of the robot's gripper At Stanford, Nils Nilsson developed the mobile robot **SHAKEY** in 1969. This robot featured a visual rangefinder, a camera, and binary tactile sensors, and it was connected to a DEC PDP-10 computer via a radio link. SHAKEY's

tasks included both obstacle avoidance and object movement within a highly structured environment. All obstacles were simple, uniformly colored blocks and wedges. SHAKEY a list of formulae representing the objects in its environment and, using a resolution-maintained theorem prover called "STRIPS," it determined plans of action that it then executed.

Although the reasoning system worked correctly, SHAKEY often encountered problems in generating the symbolic information needed for the planner from the raw data obtained from its sensors.

An example of a typical run for SHAKEY might involve the robot entering a room, locating a block, being instructed to move the block onto a platform, pushing a wedge against the platform, rolling up the ramp, and pushing the block up. However, SHAKEY never completed this sequence as a single operation. Instead, it performed several independent attempts, each with a high probability of failure. While it was possible to piece together a movie showing all the steps, the process itself was highly unreliable.

Also at Stanford, John McCarthy began a project in the early 1970s to build a robot capable of assembling a color television kit. Again, the hardware proved to be the most challenging part, as accurately inserting components into printed circuit boards was difficult. Many researchers interested in robotics during the early days of artificial intelligence shifted their focus away from the hardware aspects of robotics and concentrated instead on the software and reasoning components of control systems.[1]

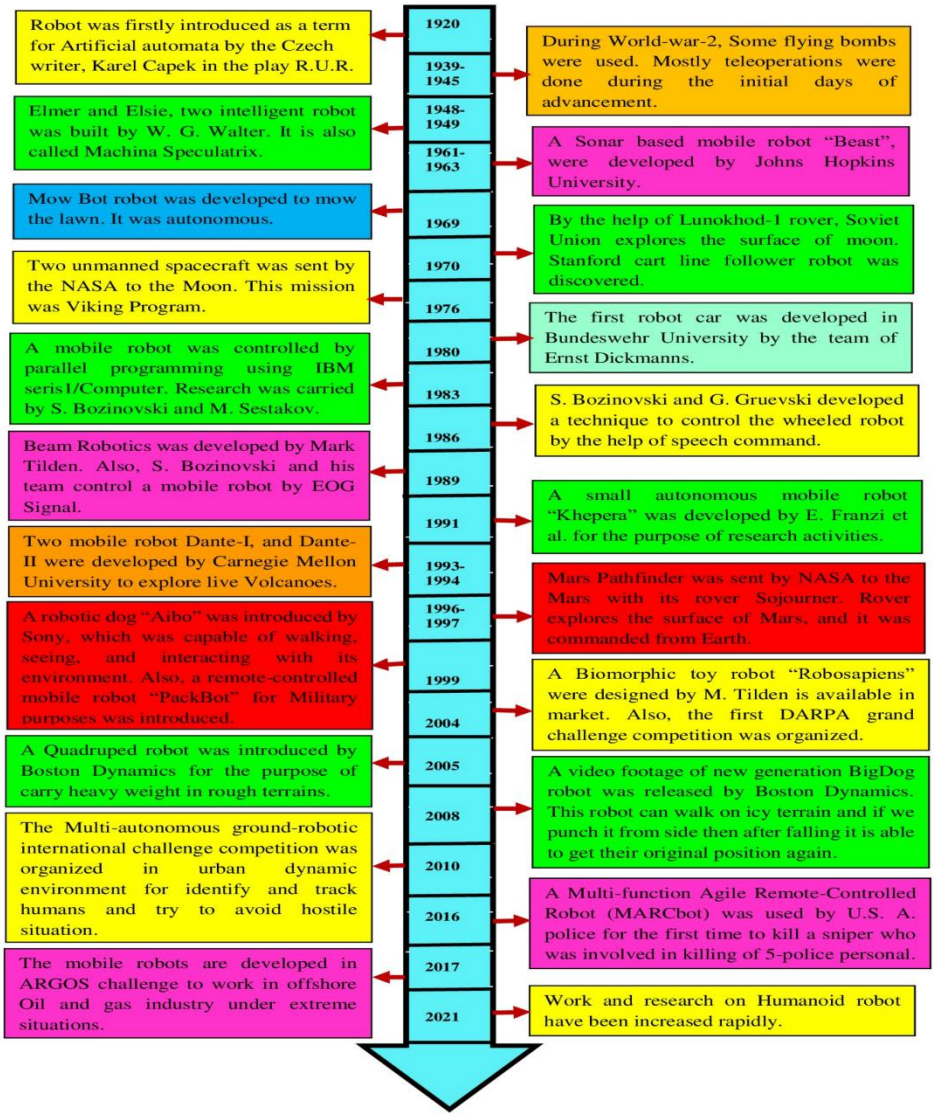


Figure 1:Some important and major events in the field of robotics science in the last 100 years

3 Definition Mobile Robotics

3.1 Origin of terms [3]

Robot was first used in 1921 by researcher in his play R.U.R. (Rossum Universal Robots), the word robot comes from the Czech "robota" which means chore, compulsory work the term robotics was first used by Asimov in 1941.

3.2 Mobile Robotics

An automatic machine equipped with memory and a program, capable of replacing humans to perform certain tasks.[2]

Mobile robots are autonomous or semi-autonomous machines capable of moving around. Unlike industrial robots (which are designed to operate within a fixed workspace), mobile robots can freely navigate and move around their environment.

The branch of robotics that develops robots which move around an environment. These robots employ an array of sensors, control mechanisms, and actuators to engage with their environment, enabling them to conduct tasks, including exploration, cartography, transportation, or service provision. Mobile robots are employed in the industrial, commercial, medical, and military sectors, as well as in research and education.[3]

4 Types of Mobile Robots

The classification of robots today aims to provide an understanding of the current state of robotics and its potential in the coming years. We can agree that there are three main categories of robots:

4.1 Humanoid Robots

This is undoubtedly the most well-known category of robots, largely due to the promotion by science fiction. It includes all anthropomorphic robots, those whose form resembles human morphology. The resemblance to human beings is sometimes so striking that, in Japan, a humanoid robot presented the evening news on June 18, 2014. This type of robot is commonly referred to as an Android.

4.2 Industrial Robots (Manipulators)

Most of these robots are stationary. When they are not, they are typically mounted on rails. This category includes manipulation robots, such as "Pick and Place" robots, welding robots, and painting robots used in the automotive industry. They currently represent the majority of robots.

4.3 Explorer Robots (Mobile)

In general, this category includes both humanoid robots capable of moving within their environment and wheeled mobile robots, as well as any other robots capable of locomotion. Wheeled mobile robots, known as UGVs (Unmanned Ground Vehicles), include robots powered by wheels or tracks.

They are generally used for exploration, which is why they are also called rovers (wanderers). The most famous examples are NASA robots, such as Curiosity and the Mars Rovers. The Curiosity robot, for instance, was sent to Mars for exploration and terrain analysis.[4]

5 General presentation of mobile robots

Unlike the industrial robot which is generally fixed, the mobile robot is equipped with means that allow it to move in its workspace. Depending on his degree of autonomy or degree of intelligence, he can be endowed with means of perception and reasoning. Some are able, with reduced human control, to model their workspace and plan a path in an environment that they do not necessarily know in advance.

Currently, the most sophisticated mobile robots are essentially oriented towards applications in variable or uncertain environments, often populated by obstacles, requiring adaptability to the task. Fig. (1.1) illustrates the structure of such a robot**.**

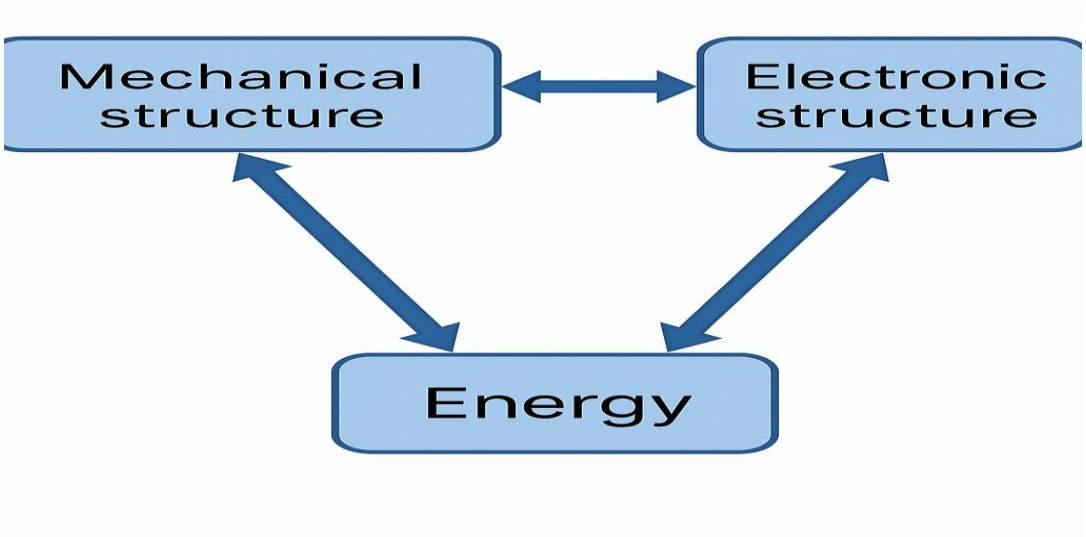


Figure 2:Structure of a mobile robot

Mobile robots have a special place in robotics. Their interest lies in their mobility which opens up applications in many fields. Like manipulator robots, they are intended to assist humans in difficult tasks (transport of heavy loads), monotonous or in a hostile environment (nuclear, marine, space, firefighting, surveillance ...) [5].

The particular aspect of mobility imposes a technological and methodological complexity which is generally added to the problems encountered by manipulating robots. Solving these problems requires the use of all available resources both at the technological level (sensors, motor skills, energy) and that of information processing through the use of artificial intelligence techniques or particular processors. The autonomy of a mobile robot is its ability to adapt or make decisions in order to carry out a task despite a lack of preliminary or potentially erroneous information. In other use cases, such as that of planetary exploration vehicles, autonomy is a fundamental point since the remote control is then impossible due to the duration of the information transmission time.

5.1 Advantages of mobile robots

Here are the advantages of mobile robots:

- ✓ Flexible and re-deployable. One major advantage of mobile robots is their flexibility in being deployed in various environments and adapted to different jobs. In opposition to traditional fixed industrial robots confined to assembly lines, mobile robots can navigate freely and reconfigure themselves based on changing needs. This lets them handle diverse material handling, delivery, inspection, or cleaning operations across multiple facilities.
- ✓ Productive and efficiency wins. Mobile robots excel at repetitive, tedious tasks that are labor-intensive for humans. By automating these processes, businesses can significantly up productivity while reducing costs associated with manual labor.
- ✓ Facing the danger. In industries involving dangerous environments like mining, construction, or chemical processing, mobile robots can take over risky operations. This minimizes the need for human workers to enter potentially dangerous areas, improving safety standards.

- ✓ Mobile robotics = optimization. Mobile robots can optimize material flow, reduce travel times, and improve overall operational efficiency. This results in leaner, more cost-effective operations that drive business growth and competitiveness.

6 Obstacle Detection and Localization

The perception of its environment is the foundation of any autonomous system. Without proper understanding and interpretation of its surroundings, a robot cannot make accurate decisions. This section describes the various tools available to a robot for detecting surrounding obstacles and the methods for localizing itself. The goal is to develop a model, more or less simplified, of the interactions between the robot and its environment. This step is crucial for the navigation of an autonomous mobile robot.

To achieve this, a robot is equipped with proprioceptive sensors that provide information about itself and exteroceptive sensors that gather information about its surrounding

6.1 Obstacle Detection and Mapping

Sensors providing information about the external environment are categorized into two types: passive and active [10]. Passive sensors gather and analyze energy from the environment, such as light, while active sensors generate energy and capture its reflection after interacting with the external environment. This principle underpins rangefinders, widely used to create real-time maps of a robot's environment. Scanning laser rangefinders are frequently employed for indoor navigation due to their high performance.

The principle of these rangefinders involves calculating the round-trip time of a light pulse. A low-power infrared wave emitted by a laser diode is reflected off the first object encountered and returns to the sensor's detector. The round-trip time determines the object's distance. A motorized rotating mirror scans a range of angles, parallel to the ground plane. These devices are precise and resistant to temperature variations, making them valuable for mobile robotics in low-to-moderate speed applications.

Ultrasonic sensors use sound waves at frequencies inaudible to humans (20–200 kHz) and measure the round-trip time of waves reflected by obstacles, similar to laser rangefinders. Animals like bats and dolphins use this echolocation method. Unlike laser rangefinders,

ultrasonic sensors are better at detecting thin elements like chair legs or fences due to their unfocused waves. However, they have a shorter range and are less effective in anisotropic media like air. One disadvantage of ultrasonic sensors is the significant divergence of the ultrasonic beam, resembling a cone rather than a focused beam. This divergence makes obstacle localization less precise. These sensors are primarily used for short-distance measurements (a few centimeters to a few meters), are sensitive to temperature variations, and their measurement frequency depends on the maximum detection range. However, they are less expensive than laser rangefinders and are often used indoors in confined navigation spaces.

Passive sensors, like stereo vision systems, use environmental energy directly. By recognizing primitives between two images, they can estimate an object's position and orientation to calculate depth. However, this requires at least two cameras, or more, for improved robustness. Omnidirectional vision is also highly beneficial in mobile robotics, enabling simultaneous monitoring of all surroundings. A camera faces a parabolic or hyperbolic mirror, producing a distorted image that complicates distance measurement. However, the panoramic view offers advantages for dynamic obstacle avoidance in structured environments. Vertical lines become radial, and horizontal lines become arcs. Vision-based systems are widely developed across various fields but remain reliant on environmental factors like lighting and contrast.

6.2. Multi-Sensor Data Fusion and Mapping

Mobile robot localization combines data from various exteroceptive and proprioceptive sources. Typically, this involves reconciling odometry measurements with absolute localization methods, such as:

- ✓ Recognizing and calculating distances from known-position beacons.
- ✓ Matching real-time maps with preexisting database maps.
- ✓ External robot localization via environmental sensors (e.g., GPS).

The simplest approach is to periodically recalibrate the robot’s position using absolute localization data to correct the odometer-based state. However, this method does not utilize all measurements simultaneously and ignores uncertainties in both odometry and absolute localization.

Measurement uncertainties, inherent to all technologies, arise from the measurement principle or technological imperfections. For example, laser rangefinders exhibit systematic errors averaging 15 mm and statistical errors around 5 mm. Combining multiple measurement methods involves weighted averaging based on the confidence level of each source.

For static obstacles, recurring measurements at a specific acquisition frequency improve precision through Kalman filtering. This statistical filter reduces un certainties with each new measurement and is widely used in robotics for obstacle-relative localization [11].

To plan a robot's movements, it is essential to model the environment using relative obstacle positions. This modeling involves creating local maps of navigable and non-navigable spaces, assuming the robot's absolute position is known. Two primary types of local environment representation are:

- **Geometric maps**, constructed from rangefinder data to recognize simple shapes (walls, corners).
- **Occupancy grids**, which discretize the environment into grid cells marked as accessible or inaccessible.

6.3 Localization

Localization tools fall into two categories: **dead reckoning** and **absolute localization** [10]. Dead reckoning involves integrating velocity or acceleration data from proprioceptive sensors (odometers, inertial units). These methods are independent of the environment but suffer from precision issues due to temporal drift. Errors accumulate over time, degrading accuracy.

The simplest and most commonly used dead reckoning method is **odometry**, which estimates relative positions by integrating instantaneous displacements based on wheel rotations. Although simple and widespread, odometry is quickly imprecise due to wheel slippage and terrain quality, leading to significant errors over long distances without regular recalibration.

For challenging terrains, like Mars exploration, traditional odometry is infeasible. Alternative methods like **visual odometry** evaluate movement using distinctive image points captured by onboard cameras, improving relative localization accuracy.

Absolute localization relies on identifiable elements in the environment (beacons) with known positions. These beacons may be:

- ✓ **Passive**, reflecting robot-emitted signals (e.g., laser, infrared).
- ✓ **Active**, emitting detectable signals.

GPS systems offer meter-level precision for outdoor navigation, enhanced by differential GPS for higher accuracy in fixed-base and mobile setups. However, GPS efficacy diminishes in urban or forested areas due to signal obstruction by structures.

Vision-based methods complement GPS for urban environments, as demonstrated by Cappelle's 3D model matching or Yang's visual odometry algorithms [12].

7 Conclusion

Mobile robotics represents a constantly evolving field, at the crossroads of artificial intelligence, mechanics and computer science. This chapter made it possible to explore the historical foundations, the key definitions, the types of mobile robots as well as their distinctive characteristics. Unlike stationary industrial robots, mobile robots are distinguished by their ability to dynamically interact with often uncertain or changing environments, thanks to a variety of sensors, location algorithms, obstacle detection and data fusion.

CHAPTER II

AUTONOMOUS NAVIGATION USING ARTIFICIAL INTELLIGENCE TECHNIQUES

1 Introduction

Autonomous navigation using artificial intelligence (AI) techniques represents a major advance in many fields, including autonomous vehicles, drones, and maritime or aerial systems. Thanks to AI, these systems can perceive their environment, make decisions in real time, and adapt to changing conditions without human intervention. This ability to learn and improve independently makes it possible to increase safety, efficiency and reduce human errors.

2 Artificial intelligence (AI)

Artificial intelligence (AI) was born in the 50s when computer pioneers explored whether computers could "think". This field aims to automate human intellectual tasks and encompasses various approaches, including machine learning, but also methods without learning, such as predefined rules. Initially, symbolic AI, based on explicit rules, dominated and was effective for well-defined logical problems, such as chess. However, it has shown limits in the face of more complex challenges, such as image recognition or translation, leading to the development of new approaches

2.1 Defining

Artificial intelligence (AI) refers to the simulation of human intelligence in machines designed to perform tasks that typically require human cognition, such as learning, reasoning, problem-solving, and perception. It encompasses a wide range of techniques, including machine learning, deep learning, and rule-based systems. AI systems can be trained to recognize patterns, make decisions, and improve over time with data. While some AI relies on predefined rules, others learn autonomously from experience. Ultimately, AI aims to create systems capable of performing complex tasks efficiently and adaptively. Marvin Lee Minsky [2] defines artificial intelligence as “the construction of computer programs that perform tasks that are, for the time being, more satisfactorily accomplished by human beings, because they require high-level mental processes such as perceptual learning, memory organization, and critical reasoning”.

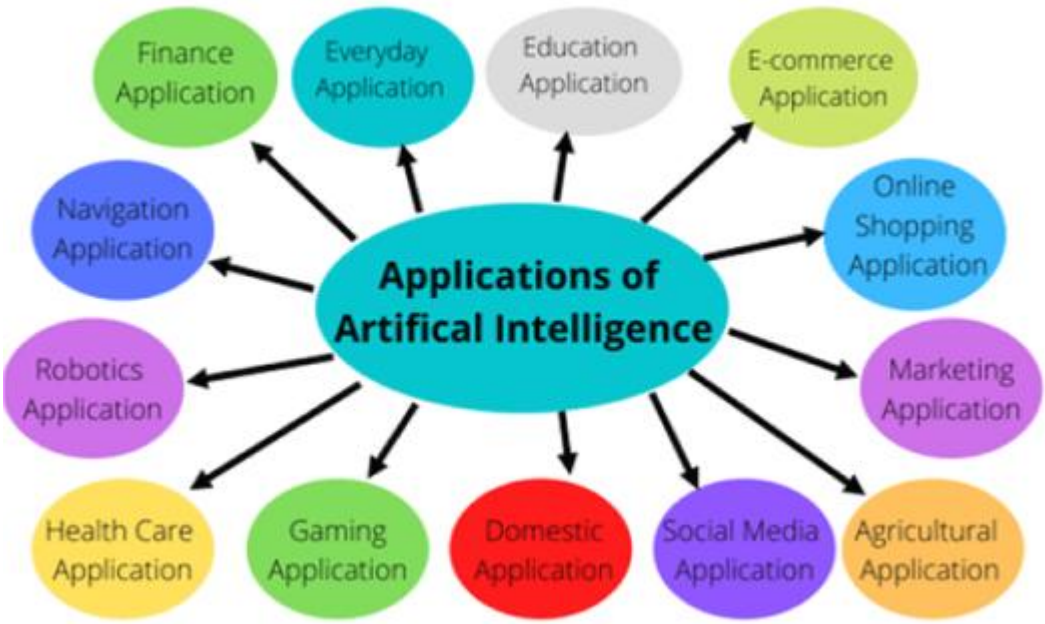


Figure 3:The different applications of artificial intelligence

2.2 Artificial intelligence techniques

A. Machine learning

Machine learning (or Machine Learning, ML) is a branch of artificial intelligence that allows systems to learn from data to perform tasks without being explicitly programmed. It is based on algorithms that identify patterns in the data and improve with experience, the main concepts and techniques are as follows:

1 Supervised Learning

The model learns from labeled data, where inputs and outputs are known, with the aim of predicting results for new data. For example, in classification, it predicts categories (such as spam detection), while in regression, it predicts continuous values (such as price prediction). Commonly used algorithms include linear regression, decision trees, support vector machines (SVM) and neural networks.

2 Unsupervised Learning

The model works with unlabeled data to discover hidden structures or patterns, in order to explore and organize the data. For example, clustering groups similar data (such as customer segmentation), while downsizing simplifies the data while retaining its essence (such as

principal component analysis, ACP). Commonly used algorithms include K-means, DBSCAN and autoencoders.

3 Reinforcement Learning

The model learns by interacting with an environment and receiving rewards or penalties, with the objective of maximizing long-term rewards. This approach is used in various fields such as video games, autonomous robots and trading systems. Commonly used algorithms include Q-learning and Deep Q-Networks (DQN)

4 Semi-Supervised Learning

This technique combines labeled and unlabeled data to improve model performance, which is especially useful when labeling the data is expensive or difficult to achieve.

B. Distinction between artificial intelligence, automatic machine deep

1 Learning Artificial Intelligence (AI)

Artificial intelligence (AI) is the broadest field, aimed at creating systems capable of performing tasks that require human intelligence, such as reasoning, learning and perception. It includes all techniques, whether they are based on rules, algorithms or learning models. Examples of applications include expert systems, robots and virtual assistants**.**

2 Automatic Learning (Machine Learning, ML)

Machine Learning (ML) is a subfield of AI that focuses on the creation of models capable of learning from data to make predictions or decisions. It uses algorithms to identify patterns in the data and improve with experience. Examples of applications include classification, regression and clustering.

3 Deep Learning

Deep Learning (DL) is a subfield of Machine Learning that uses artificial neural networks with several layers, hence the term "deep". It specializes in the processing of complex data, such as images, text and sounds, and requires large amounts of data as well as significant computing power. Examples of applications include image recognition, machine translation and self-driving cars.

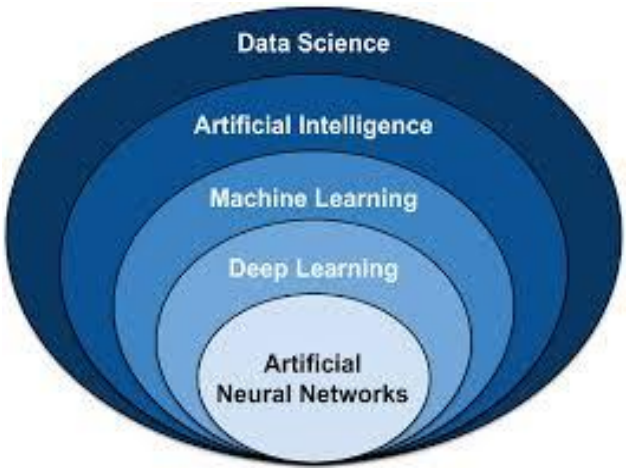


Figure 4:The relation between IA, ML and Deep Learning

C. AI Techniques in Robot Navigation

Several AI techniques can be used in robot navigation, including particle swarm optimization (PSO), ant colony optimization (ACO), genetic algorithm (GA), neural networks (NNs), fuzzy logic system, and deep reinforcement learning (DRL).

Fuzzy Logic Technique: Eight rule-based fuzzy controllers can be used for path following and obstacle avoidance for mobile robots, while gradient method-based Takagi Sugeon fuzzy controllers can tune various membership function parameters to acquire the optimal result for robot navigation.

Similarly, the Khepera simulator with fuzzy logic-based agents can be employed to control robots. The behavior of every agent, including sensor value, robot position, and heading angle, can be controlled by defining the sets of fuzzy rules. A memory system can be included to further increase the system's efficiency by enabling the robot to identify alternative routes when it gets trappedA fuzzy logic controller can be used for path following depending on the orientation and position errors. The control of two wheels independently using the controller can provide longitudinal and lateral control of the robot. Fuzzy logic controllers can also be utilized for sensor-based mobile robot navigation in indoor environments. Fuzzy controllers can be optimized by combining the RL and GA methods. Fuzzy logic with visual landmark recognition can be used for obstacle avoidance. The path following and control problem of

autonomous mobile robots can be solved using an ultrasonic range finder by combining GA and fuzzy controller.

NNs: NNs can be used to solve several robot navigation problems, including defining schedules and identifying the shortest route for traveling. For instance, multilayer feed-forward artificial NN can be combined with the Q reinforcement method for effective path planning.

Similarly, a multilayer NN controller and proportional integral derivative (PID) can be utilized to design an Arduino microcontroller-based direct current (DC) motor for controlling speed in robots. NN architecture can also be used for designing an automatic steering controller for an autonomous mobile robot, and to develop a collision-free path in a dynamic environment.

A biologically inspired NN can be employed to develop a wall following robot, while a hybrid NN can be used for efficient robot navigation. Goal-seeking and obstacle-avoidance behaviors can be realized in robots using NN.

The trajectory tracking problem in robot navigation can be solved using the adaptive NN PID controller, while a combination of the first-order Sugeno-fuzzy inference model and adaptive neuro-fuzzy inference system (ANFIS) can be utilized for coordinating several robots and path planning.

Two different NN controllers can be employed for path following and controlling robots. Additionally, Hopfield NN can be used for path planning and obstacle avoidance in complex environments. Multilayered NN and recurrent neural network (RNN) can be utilized for designing intelligent navigation systems for mobile robots and solving path following and localization problems, respectively. The RNN assists the robot in autonomously navigating an unknown environment.

Moreover, a type-2 fuzzy neural network (IT2FNN) can be employed to effectively address the obstacle avoidance and orientation stabilization of wheeled robots. IT2FNN possesses three layers, including the output, hidden, and input layer, and four inputs. Angular and linear velocities of the robot are the outputs of the robot. Dynamic nonholonomic robots can be controlled using NN.

GA: GAs can be used to solve path planning problems in both dynamic and static environments. The navigation path length of robots in a cluttered space can be optimized using the Petri-GA technique.

A fuzzy controller combined with GAs can be used for the guidance of robots in a static and dynamic environment and for optimizing the navigation path length. GA can be utilized to select the most suitable membership function parameters from a fuzzy inference system to control a robot's steering angle in a partially unknown environment.

The optimal path for a robot can be identified using GA and fuzzy logic, while effective path planning of several robots can be achieved using an improved GA, which can guide robots efficiently from the origin to the destination without any collision.

Motion control can be realized by implementing a genetic-fuzzy controller (GA-FLC) for tuning and optimizing the Gaussian membership function parameters. Additionally, multiple objective genetic algorithm (MOGA) and single fitness-based GA can be employed for path optimization of the robot and avoiding navigation problems in the dynamic environment, respectively.

PSO: The motion planning problem of a robot can be solved using multi-objective PSO, while the velocities of the left motor and right motor of the differential drive robot can be determined using a PSO-based optimal fuzzy controller.

The parallel met heuristic PSO (PPSO) algorithm can be used to address the global path-planning problem of robots. Moreover, an evolutionary-group-based PSO (EGPSO) for automatic learning of fuzzy systems can be utilized for wall following control and robot navigation.

DRL: Uncertainty-aware RL, double deep Q network (DDQN), asynchronous deep deterministic policy gradient (DDPG), fast recurrent DPG, and successor feature RL can be utilized for local obstacle avoidance.

Long short-term memory (LSTM) + DRL, asynchronous advantage actor-critic (A3C) + LSTM, and LSTM + proximal policy optimization (PPO) can be utilized for indoor navigation.

Additionally, PPO, parallel DDPG, parallel PPO, and collision avoidance with DRL can be employed for multi-robot navigation.

D. Autonomous navigation of mobile robots

Mobile robots are machines capable of moving around in their environment, often using wheels, paws, or other means of locomotion. They can be equipped with sensors to perceive their environment, navigation systems to move safely, and control systems to make decisions and perform tasks. Mobile robotics is a constantly evolving field, with applications more and more diverse and more and more advanced technology. It has the potential to transform the way humans interact with their environment and solve many practical problems that they face in everyday life. She concerns the design, construction and use of mobile robots have many practical applications, such as monitoring, maintenance, exploration, parcel delivery, security, and logistics. They can be used in various environments, such as factories, warehouses, construction sites of construction, agricultural fields, and outdoor spaces. Technological advances in the matter of sensors, data processing, control, and mechanics have made it possible to develop mobile robots that are increasingly autonomous and capable of adapting to varied situations.

Autonomous navigation of a mobile robot refers to the ability of a robot to move and to navigate in its environment without human intervention or assistance. This includes tasks such as planning a ride, obstacle avoidance, and the adjustment of the movement according to the inputs of the sensors. Autonomous navigation requires a combination of hardware and software, such as sensors to perceive the environment, a processing power to make decisions based on this perception, and actuators to execute the movement. This is a fundamental ability for the majority of the mob

The navigation of a mobile robot consists of determining an optimal trajectory enabling the robot to move from a starting point (an initial point) pi to a destination point (an endpoint) pf to reach a desired goal. It also involves seeking to move freely in the configuration space without colliding autonomously with obstacles close to the robot. There are two approaches to navigation techniques:

A- Planning of movement in the environment (workspace) and execution by servo-controlling the robot's movement to follow the desired instructions (planning-execution diagram);

B- Navigation by decomposition into a set of more reactive primitives. This corresponds to executing a series of sub-tasks (following a wall, avoiding an obstacle) to divide the overall task into a sequence of primitives. To achieve autonomous navigation, a mobile robot must implement several missions, the main functionalities of which are: [15]

- Motion planning
- Localization
- Path following
- Obstacle avoidance
- Parking

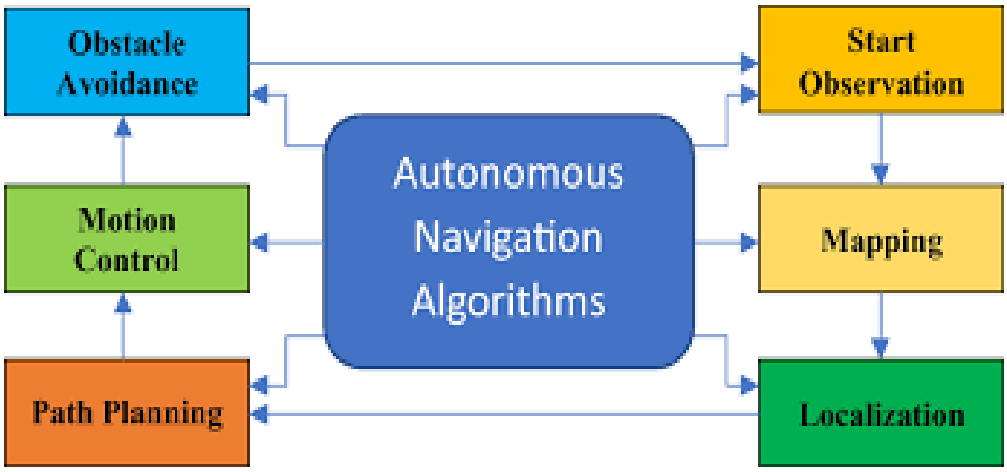


Figure 5:Autonomous navigation of mobile robots

3 Conclusion

In conclusion, although autonomous navigation based on artificial intelligence has the potential to transform many sectors, it remains necessary to continue developing robust technologies and to put in place appropriate regulatory frameworks to guarantee a smooth and secure transition toward a future where these autonomous systems will become commonplace.

CHAPTER III

REINFORCEMENT LEARNING

1 Introduction

Reinforcement learning is a method that enables an agent (such as a robot) to learn, through trial and error, the optimal action to take in each perceived situation in order to maximize a reward.

We will begin by introducing the core principles of reinforcement learning. Next, we will define the developmental approach to learning that we aim to apply in robotics. Finally, we will provide a brief overview of relevant theoretical tools.

2 Principles of reinforcement learning

A commonly used solution for implementing reinforcement learning involves an agent operating within an environment, making decisions based on its current state. The agent interacts with the environment by taking actions, and the environment provides feedback in the form of rewards (positive, negative, or zero).

More formally, reinforcement learning is a class of machine learning problems where the goal is to learn—through experimentation—how to act in different situations to maximize a numerical reward over time. The agent strives to optimize its cumulative rewards by exploring different actions and strategies (e.g., exploration, policy monitoring, etc.) within the environment. The nature and selection of these actions at any given stage of learning depend on the chosen algorithm. [17]

3 Definitions and concepts

3.1 Definitions

Reinforcement learning is generally operated in an interaction framework, illustrated in Figure (3.1); the learning agent interacts with an initially unknown environment and receives a representation of the state and an immediate reward in return. The environment produces an S_i state at each stage ***t*** by receiving the current state $\boldsymbol{s_i}$ the agent reacts with an action A_i that he then calculates and executes. The agent acts according to a policy $\pi (a_i \mid s_i)$, which represents the probability of taking an action a_i when he is in the state s_i (in a deterministic environment, $\pi (s_i) = a_i$), this action leads to a transition of the environment to a new state. The environment provides the new state t 1 s_{i+1} as well as a reward r_i, which indicates how good the new state is. The agent receives the new representation and the corresponding reward, and the whole process repeats. The agent's goal is to maximize the cumulative reward: max $\sum r_i$ (the rewards are often weighted to avoid exploding sums). The accumulated reward sum is called return R_i

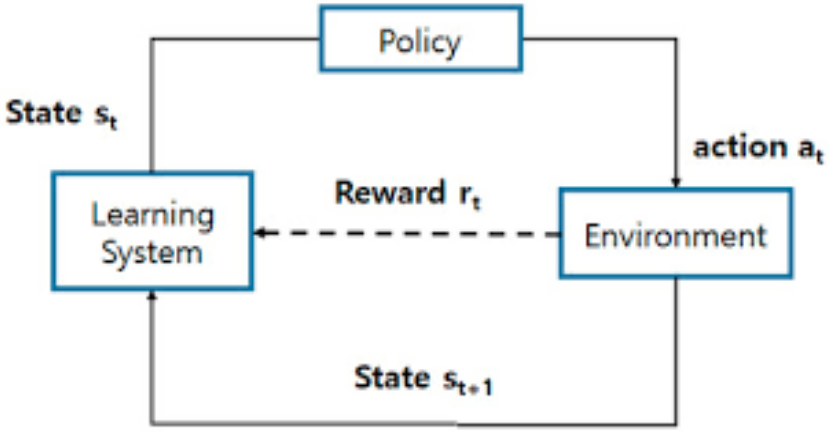


Figure 6:Interaction, agent-environnent

The reinforcement learning environment is usually formulated as a Markovian Decision Process (MDP), and the objective is to learn a control strategy to maximize the total reward that represents a long-term goal**. [17]**

3.2 Objectives of reinforcement learning

The goal of reinforcement learning is to find an optimal policy π* that associates states or observations with actions in order to maximize the expected return J, which corresponds to the expected cumulative reward. In a finite horizon model, we only seek to maximize the expected reward for the horizon H, that is to say for the next H steps (over time) h:

$$J = \mathbb{E}\left\{\sum_{h=0}^{H} R_{\mathrm{h}}\right\}$$

This setting can also be applied to model problems where we know how many steps remain. Alternatively, future rewards can be discounted with a discount factor γ (with $0 \leq \gamma < 1$) **[17]**

$$J = \mathbb{E}\left\{\sum_{h=0}^{H} \gamma^{\mathrm{h}} R_{\mathrm{h}}\right\}$$

3.3 Markovian Decision-making Processes

A Markovian Decision Process describes a sequential decision-making problem in which an agent must choose the sequence of actions that maximizes an optimization criterion based on a reward Formally, a MDP is a set M = { S,A,T,r,γ } where :

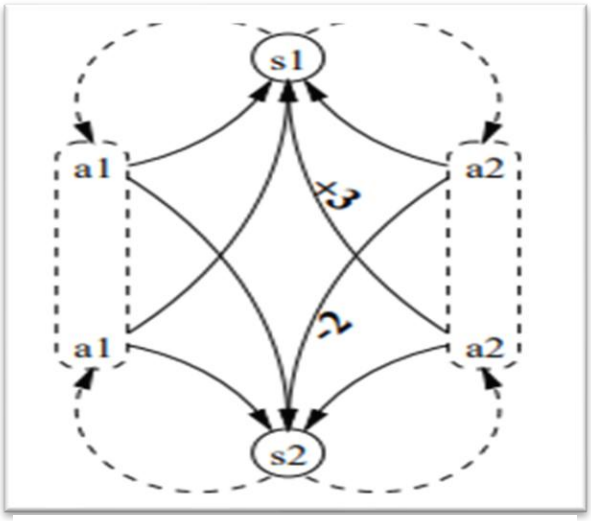


Figure 7:Example of MDP

$S = \{s_1, \ldots, s_n\}$ is a finite set of N states that represents the dynamic environment,

$A = \{a_1, \ldots, a_k\}$ k is a set of k actions that can be executed by an agent,

$T : S \times A \times S \rightarrow [0,1]$ is a transition probability function, or a transition model, where T (s ,a,s') represents the probability of state transition during the application of the action a∈A to the state s∈S leading to the state s' ∈ S , that is , $T(s, a, s') = P(s'|s, a)$,

$r : S \times A \rightarrow R$ is a reward function whose absolute value is bounded by Rmax or r (s,a) , represents the immediate reward obtained during the execution of the action a∈A in the state s∈S ,

$\gamma \in [0,1]$ is a discount factor (discount or devaluation).

Given a MDP M, the agent-environment interaction in Figure (3.2) proceeds as follows: either t∈ N is the current time, or s∈ S and a_t∈ A represent the random state of the environment and the action chosen by the agent at time t, respectively. Once the action is selected, it is sent to the system, which performs a transition: (s_{t+1}, r_{t+1}) P (.[S_t,A_t)

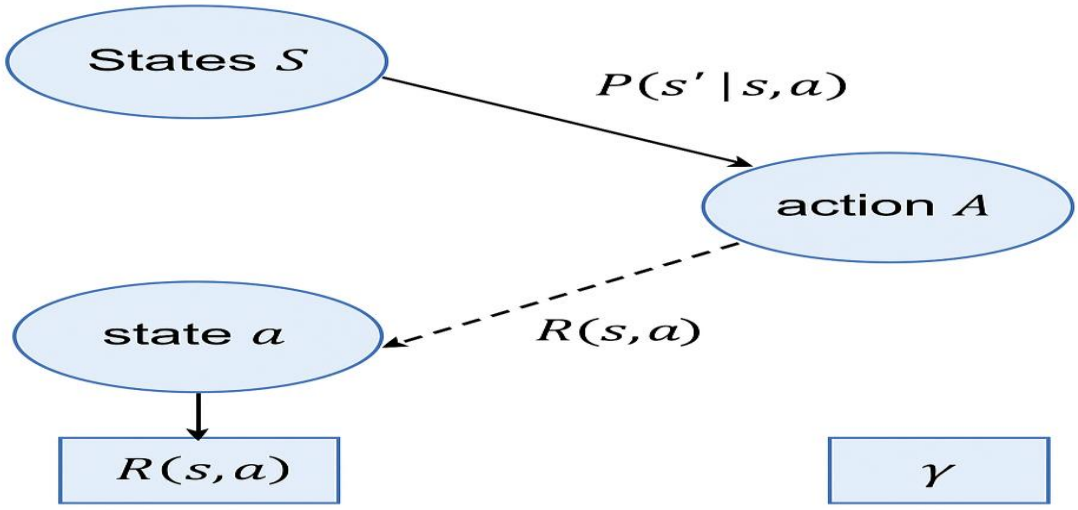


Figure 8:Diagram of a Markov Decision Model (MDP)

In particular, s_{t+1} is random and $P\ (s_{t+1} = s' / s_t = s\ , a_t = a) = T\ (s, a, s')$ is true for all s, s'∈ S and a∈ A. Moreover $[r_{t+1}/s_t, a_t\backslash] = r(s_t, a_t)]$. The agent then observes the next state S_{t+1}, and the reward r_{t+1}, chooses a new action a∈∈A and the process is repeated.

An important property of a MDP is that the process is Markovian, that is, the optimal action to be taken for a particular state does not depend on the history of actions and states that the agent has previously visited. The current state provides all the information the agent needs to act.

The Markov hypothesis implies that the sequence of state-action pairs specifies the transition model T:

$$\boldsymbol{P}\ (\boldsymbol{S}_{t+1} / \boldsymbol{S}_t, \boldsymbol{A}_t, \ldots\ldots, \boldsymbol{S}_0, \boldsymbol{A}_0) = \ \boldsymbol{P}\ (\boldsymbol{S}_{t+1} / \boldsymbol{S}_t, \boldsymbol{A}_t)$$

The state transition can be deterministic or stochastic. In the deterministic case, taking a given action in a given state always gives the same next state, while in the stochastic case, the next state is a random variable. The objective of the learning agent is to determine a theory of choice of actions to maximize the expected discounted total reward:

$$\boldsymbol{R} = \sum_{t=1}^{\infty} \gamma^t \boldsymbol{R}_{t+1}$$

If γ<1, then the rewards received far in the future have an exponentially smaller value than those received at the first stage **[17]**

3.4 Markov's property

Assuming that the current state and the current return depend only on the previous state and the action that has just been issued. This is a fundamental property, and must be respected by all PDM. It is the Markov property (we also say that the environment is Markovian). There is then no need for memory to make decisions at best: only knowledge of the current state is useful. The only trace of memory resides in the behavior learning performed by the agent **[18]**

3.5 Policy

The behavior of the agent is defined by a policy π:{S, A}→[1,0], which guides the agent probabilistically by specifying, for each state s , the probability of carrying out the action a (therefore π (s) = a). The goal is to find the optimal policy π * maximizing the long-term reward, we note it

$(s,a) \rightarrow \pi\,(s,a) = Pr\,[a_t = a/s_t = s]$. We therefore seek to solve an optimal control problem, where reinforcement learning is not defined by a certain class of algorithms, but by the problem it seeks to solve. That of the optimal control. The return received by the agent from time t is defined by the following sum: $\boldsymbol{R}_t = \boldsymbol{r}_t + \boldsymbol{r}_{t+1} + \cdots .. + \boldsymbol{r}_{t+k-1}$

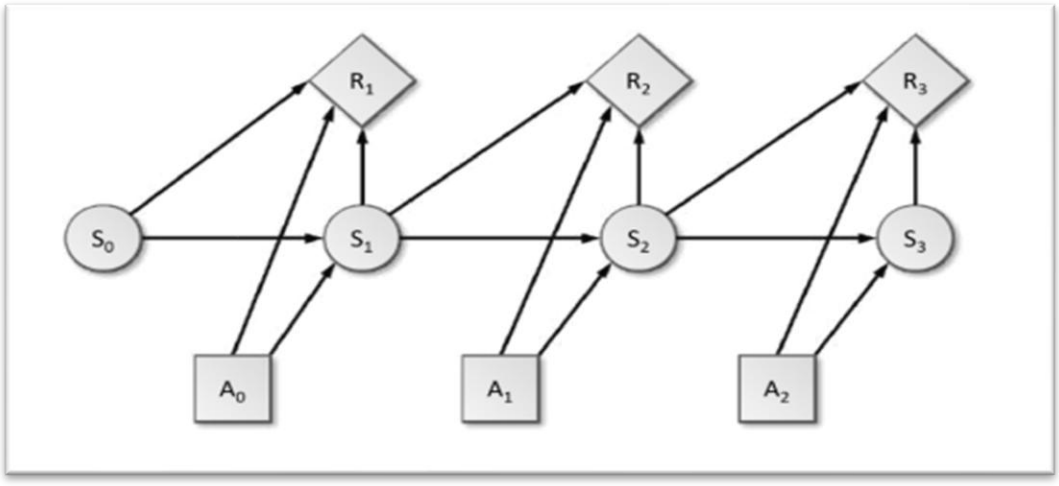


Figure 9:Decision network of a finite MDP

3.6 Other models

From the Markov model, variants are defined that serve both in pattern recognition and in planning (such as MDPs). The general goal is to guess the present or future state of a system. Here are some derived Markovian models:

- **HMM** In a Hidden Markov Model, the state of the system is not known. On the other hand, we have an observation that is linked by probabilistic laws to the states. We cannot be sure of a state with a perception of the outside world, but a series of perceptions can refine a judgment. It is the main pattern recognition tool derived from Markov models.
- **PDM** (see their detailed presentation in this report)
- **MMDPS** (Multiple CDMS) are a variant of MDPs adapted to the case of multi-agent systems, as are DEC-CDMS (decentralized CDMS)and Markov games. These three models are presented in the appendix.
- **SMDP** The so-called Semi-Markovian model aims to improve time management, considering that the passage in a state can be of variable duration (according to stochastic laws).
- **POMDP** is a mixture of HMMs and MDPs, these Partially Observable MDPs add the idea that an agent only has a partial perception of his environment, so that he only knows an observation and not a complete state [23]

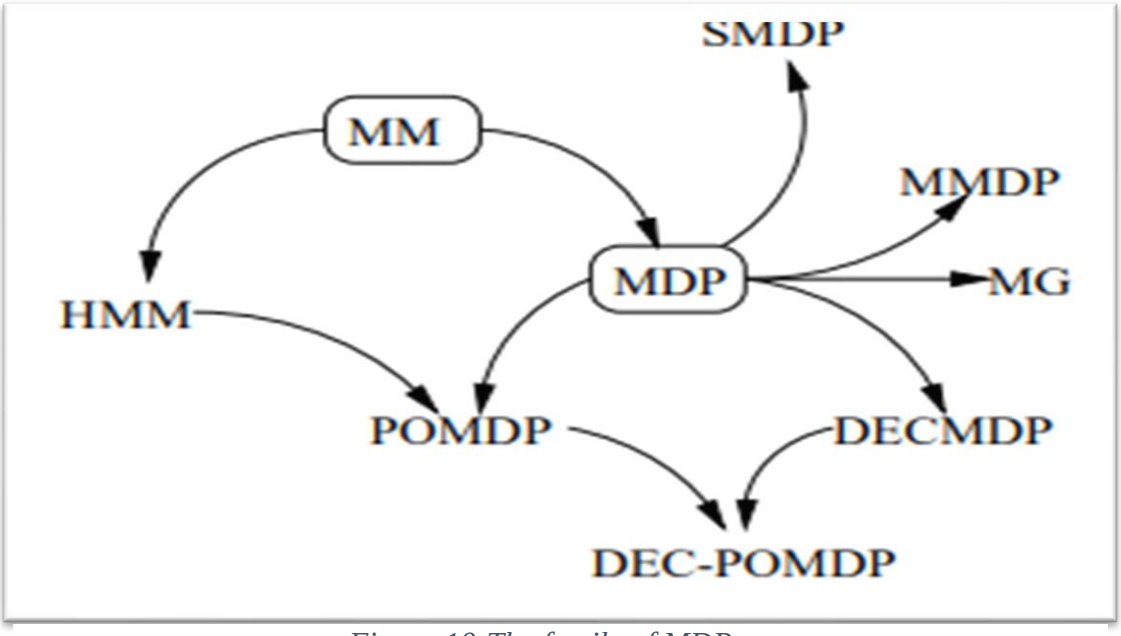


Figure 10:The family of MDP

4 Tabular methods

The first methods presented are the tabular methods where the CDM is finite and the dimensions of the state and action spaces are small, that is to say that the value function and the Q function can be represented in the form of tables (Table (3.1) and Table (3.2)).

The objective of tabular methods is to fill these tables, because once they are known **[17]**

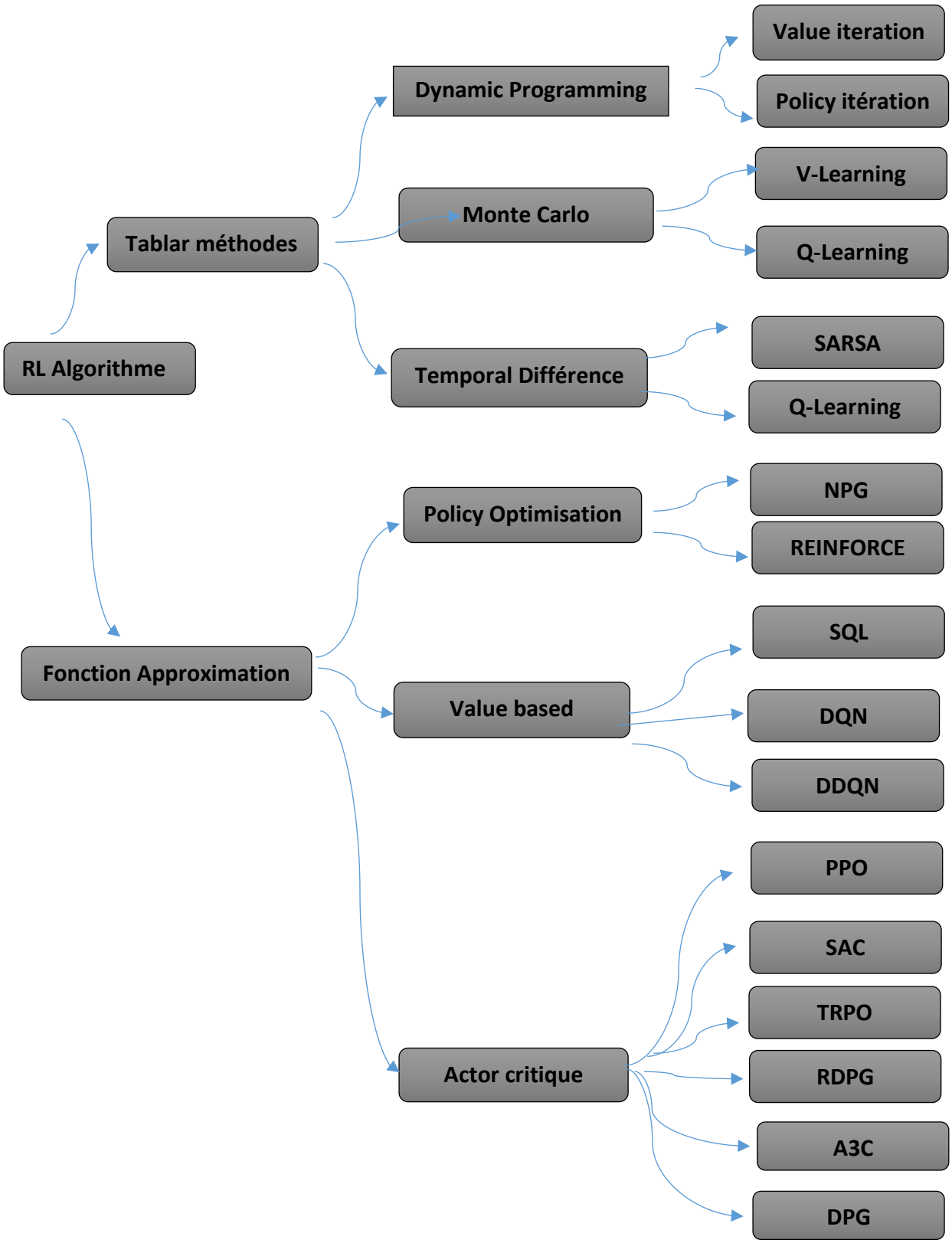
state	**value**
s_0	15
s_1	20
…	…
s_n	...

Table 1: Value function

		Actions			
		high	**bottom**	**left**	**right**
State	s_0	-1	1 2	0	15
	s_1	3	2 0	4	14
	…	…	…	…	…
	s_n	…	…	…	…

Table 2: Function Q

We can divide the tabular methods into 3 categories: dynamic programming, Monte Carlo methods ,time difference reinforcement learning and TD learning.



Classification of the main reinforcement learning algorithms

4.1 Dynamic Programming

The term dynamic programming (DP) refers to a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as a Markov decision process (MDP). Classical DP algorithms are of limited utility in reinforcement learning both because of their assumption of a perfect model and because of their great computational expense, but they are still important theoretically. DP provides an essential foundation for the understanding of the methods presented in the rest of this book. In fact, all of these methods can be viewed as attempts to achieve much the same effect as DP, only with less computation and without assuming a perfect model of the environment. [19]

4.2 Monte Carlo Methods

Monte Carlo methods are ways of solving the reinforcement learning problem based on averaging sample returns. To ensure that well-defined returns are available, here we define Monte Carlo methods only for episodic tasks. That is, we assume experience is divided into episodes, and that all episodes eventually terminate no matter what actions are selected. Only on the completion of an episode are value estimates and policies changed. Monte Carlo methods can thus be incremental in an episode-by-episode sense, but not in a step-by-step (online) sense. The term “Monte Carlo” is often used more broadly for any estimation method whose operation involves a significant random component. Here we use it specifically for methods based on averaging complete returns [19]

4.3 Temporal-Difference Learning

The Temporal Difference TD methods are a combination of Monte Carlo methods and dynamic programming methods. Unlike Monte Carlo methods, TD methods do not need to wait until a return estimate is available (i.e. at the end of an episode) to update the value function. Instead, they use time errors and have to wait until the next time step. The temporal error is the difference between the old estimate and a new estimate of the value function, taking into account the reward received in the current example. These updates are carried out iteratively and, unlike dynamic programming methods, only take into account the sampled successor states rather than the complete distributions on the successor states. Like the Monte Carlo methods, these methods are model-free, since they do not use a model of the transition function to determine the value function and can learn directly from the raw experience

without a model of the dynamics of the environment. In this context, the value function cannot be calculated analytically but must be estimated from the sampled transitions in the MDP

5 Fundamental algorithms:

The TD algorithm (0)

The elementary reinforcement learning algorithm, called the "time difference" algorithm, is called TD. We note it here TD (0) for reasons that will appear when we present the eligibility traces. This algorithm is based on a comparison between the reward that we actually receive and the reward that we expect to receive based on the estimates constructed previously. If the estimates of the value functions in the states s_t and s_{t+1}, denoted V (s_t) and V (s_{t+1}), were accurate, we would have

$$V\ (s_t)\ =\ r_t\ +\ yr_{t+1}+\ y2r_{t+2}+\ y3r_{t+3}+\ldots\ldots$$

$$V\ (s_{t+1})\ =\ r_{t+1}\ +\ \gamma\ r_{22}+\ \gamma 2\ r_{t+3}+\cdots\ldots\ldots\ldots$$

So we would have: $V\ (st)\ =\ r_t\ +\ yV\ (s_{t+1})$

It can be seen that the time difference error δ_k measures the error between the effective values of the estimates V (s) and the values that they should have. The temporal difference method consists in correcting this error little by little by modifying the value of V (s_t)according to a Windrow-Hoff type equation, which is used in the field of neural networks:

$$V\ (s_t) \leftarrow\ V\ (s_t)+\ \alpha[r_t+\gamma\ V\ (s_{t+1})-\ V\ (s_t)]$$

This update equation makes it possible to immediately understand how the temporal difference algorithms combine the properties of dynamic programming with those of Monte Carlo methods. Indeed, it reveals the following two characteristics:

- ✓ As in dynamic programming algorithms, the estimated value of V (s_t)is updated as a function of the estimated value of V (s_{t+1}), there is therefore propagation of the estimated value to the current state from the estimated values of the successor states.
- ✓ As in the Monte-Carlo methods, each of these values results from a local estimation of the immediate rewards which is based on the experience accumulated by the agent over his interactions with his environment.

It can therefore be seen that the temporal difference methods and, in particular, TD (0), are based on two coupled convergence processes, the first estimating more and more precisely the immediate reward received in each of the states and the second approaching better and better the value function resulting from these estimates by propagating them step by step.

In the case of TD (0), the updates are made locally each time the agent makes a transition in his environment, based on information limited to his current state ST_t, the successor state S_{t+1} and the reward R_t received following this transition. A proof of convergence of the algorithm has been proposed by Dayan and Sejnowski.

On the other hand, it should be noted that, as TD (0) estimates the value function of each of the states of a problem, in the absence of a model of the transitions between the states, the agent is unable to deduce which policy to follow, because he cannot take a step forward to determine which action will allow him to reach the next state of greater value. This point explains that we prefer to resort to algorithms that work on a value function associated with state-action pairs rather than the state alone. [21]

The Sarsa algorithm

The form of the Bellman equation V = LV is not satisfactory to derive directly an adaptive resolution algorithm. For this, Watkins introduced the value function Q, whose data is equivalent to that of V when we know the transition function p. [6]

The SARSA algorithm is similar to the TD (0) algorithm except that it works on the values of the pairs (s,a) rather than on the value of the states. Its update equation is identical to that of TD (0) by replacing the value function by the action value function :

$$\boldsymbol{Q = (s_n, a_n) \leftarrow Q(s_n, a_n) + a[r_n + yQ(s_{n+1}, a_{n+1}) - Q(s_n, a_n)]} \ldots\ldots \mathbf{1.1}$$

The information necessary to carry out such an update while the agent is carrying out a transition is the quintuple (s_n,a a,r,, s a, a a) from which the name of the algorithm derives

Carrying out these updates implies that the agent determines with a step of looking forward what is the next action that he will perform during the next time step, when the action n a in the state s l will have led him to the state s_{n+1}

As a result of this implication, there is a close dependence between the question of learning and the question of determining the optimal policy. In such a framework, there is only one policy that must take into account both exploration and exploitation concerns and the agent is required to carry out this learning only on the basis of the policy that he actually follows. An algorithm such as SARSA is said to be in politics. The dependence that this induces between exploration and learning considerably complicates the development of proofs of convergences for these algorithms, which explains why such proofs of convergence appeared much later than for so-called out-of-policy algorithms such as Q-Learning, which we will now see **[21]**

Q-Learning in Reinforcement Learning (detailed approach)

Q-learning is a **model-free reinforcement learning algorithm** used to train agents (computer programs) to make optimal decisions by interacting with an environment. It helps the agent explore different actions and learn which ones lead to better outcomes. The agent uses trial and error to determine which actions result in rewards (good outcomes) or penalties (bad outcomes).

Over time, it improves its decision-making by updating a **Q-table**, which stores **Q-values** representing the expected rewards for taking particular actions in given states.

Key Components of Q-learning

1. Q-Values or Action-Values

Q-values represent the expected rewards for taking an action in a specific state. These values are updated over time using the Temporal Difference (TD) update rule.

2. Rewards and Episodes

The agent moves through different states by taking actions and receiving rewards. The process continues until the agent reaches a terminal state, which ends the episode.

3. Temporal Difference or TD-Update

The agent updates Q-values using the formula:

$$Q(S,A) \leftarrow Q(S,A) + \alpha(R + \gamma Q(S',A') - Q(S,A))$$

Where,

- **S** is the current state.
- **A** is the action taken by the agent.
- **S'** is the next state the agent moves to.
- **A'** is the best next action in state **S'**.
- **R** is the reward received for taking action **A** in state **S**.
- **γ** (Gamma) is the **discount factor**, which balances immediate rewards with future rewards.
- **α** (Alpha) is the **learning rate**, determining how much new information affects the old Q-values.

4. ϵ-greedy Policy (Exploration vs. Exploitation)

The ϵ-greedy policy helps the agent decide which action to take based on the current Q-value estimates:

- **Exploitation:** The agent picks the action with the highest Q-value with probability 1–ϵ. This means the agent uses its current knowledge to maximize rewards.
- **Exploration:** With probability ϵ, the agent picks a random action, exploring new possibilities to learn if there are better ways to get rewards. This allows the agent to discover new strategies and improve its decision-making over time.

How does Q-Learning Works?

Q-learning models follow an **iterative process**, where different components work together to train the agent:

1. **Agent**: The entity that makes decisions and takes actions within the environment.
2. **States**: The variables that define the agent's current position in the environment.
3. **Actions**: The operations the agent performs when in a specific state.
4. **Rewards**: The feedback the agent receives after taking an action.
5. **Episodes**: A sequence of actions that ends when the agent reaches a terminal state.
6. **Q-values**: The estimated rewards for each state-action pair.

Steps of Q-learning:

1. **Initialization**: The agent starts with an initial Q-table, where Q-values are typically initialized to zero.
2. **Exploration**: The agent chooses an action based on the ϵ-greedy policy (either exploring or exploiting).
3. **Action and Update**: The agent takes the action, observes the next state, and receives a reward. The Q-value for the state-action pair is updated using the TD update rule.
4. **Iteration**: The process repeats for multiple episodes until the agent learns the optimal policy.

Methods for Determining Q-values

1. Temporal Difference (TD):

Temporal Difference is calculated by comparing the current state and action values with the previous ones. It provides a way to learn directly from experience, without needing a model of the environment.

2. Bellman's Equation:

Bellman's Equation is a recursive formula used to calculate the value of a given state and determine the optimal action. It is fundamental in the context of Q-learning and is expressed as:

$$\boldsymbol{Q(s,a) = R(s,a) + \gamma max_a Q(s',a)}$$

Where:

- **Q(s, a)** is the Q-value for a given state-action pair.
- **R(s, a)** is the immediate reward for taking action **a** in state **s**.
- **γ** is the **discount factor**, representing the importance of future rewards.
- $max_a Q(s', a)$ is the maximum Q-value for the next state **s'** and all possible actions.

What is a Q-table?

The **Q-table** is essentially a **memory structure** where the agent stores information about which actions yield the best rewards in each state. It is a table of Q-values representing the agent's understanding of the environment. As the agent explores and learns from its interactions with the environment, it updates the Q-table. The Q-table helps the agent make informed decisions by showing which actions are likely to lead to better rewards.

Structure of a Q-table:

- Rows represent the states.
- Columns represent the possible actions.
- Each entry in the table corresponds to the Q-value for a state-action pair.

Over time, as the agent learns and refines its Q-values through exploration and exploitation, the Q-table evolves to reflect the best actions for each state, leading to optimal decision-making.

Implementation of Q-Learning

Here, we implement basic Q-learning algorithm where agent learns the optimal action-selection strategy to reach a goal state in a grid-like environment.

Step 1: Define the Environment

Set up the environment parameters including the number of states and actions and initialize the Q-table. In this each state represents a position and actions move the agent within this environment.

Step 2: Set Hyperparameters

Define the parameters for the Q-learning algorithm which include the learning rate, discount factor, exploration probability and the number of training epochs.

Step 3: Implement the Q-Learning Algorithm

Perform the Q-learning algorithm over multiple epochs. Each epoch involves selecting actions based on an epsilon-greedy strategy updating Q-values based on rewards received and transitioning to the next state.

Step 4: Output the Learned Q-Table

After training, print the Q-table to examine the learned Q-values which represent the expected rewards for taking specific actions in each state.

Advantages of Q-learning

- **Trial and Error Learning**: Q-learning improves over time by trying different actions and learning from experience.
- **Self-Improvement**: Mistakes lead to learning, helping the agent avoid repeating them.
- **Better Decision-Making**: Stores successful actions to avoid bad choices in future situations.
- **Autonomous Learning**: It learns without external supervision, purely through exploration.

Disadvantages of Q-learning

- **Slow Learning**: Requires many examples, making it time-consuming for complex problems.
- **Expensive in Some Environments**: In robotics, testing actions can be costly due to physical limitations.
- **Curse of Dimensionality**: Large state and action spaces make the Q-table too large to handle efficiently.
- **Limited to Discrete Actions**: It struggles with continuous actions like adjusting speed, making it less suitable for real-world applications involving continuous decisions.

Applications of Q-learning

Applications for Q-learning, a reinforcement learning algorithm, can be found in many different fields. Here are a few noteworthy instances:

1. **Atari Games:** Classic Atari 2600 games can now be played with Q-learning. In games like Space Invaders and Breakout, Deep Q Networks (DQN), an extension of Q-learning that makes use of deep neural networks, has demonstrated superhuman performance.
2. **Robot Control:** Q-learning is used in robotics to perform tasks like navigation and robot control. With Q-learning algorithms, robots can learn to navigate through environments, avoid obstacles, and maximise their movements.
3. **Traffic Management:** Autonomous vehicle traffic management systems use Q-learning. It lessens congestion and enhances traffic flow overall by optimising route planning and traffic signal timings.
4. **Algorithmic Trading:** The use of Q-learning to make trading decisions has been investigated in algorithmic trading. It makes it possible for automated agents to pick up the best strategies from past market data and adjust to shifting market conditions.

5 Personalized Treatment Plans: To make treatment plans more unique, Q-learning is used in the medical field. Through the use of patient data, agents are able to recommend personalized interventions that account for individual responses to various treatments.

5.2 The difference between SARSA and Q-Learning

The essential difference between SARSA and Q-Learning is at the level of the definition of the error term. The term $Q(s_{t+1}, a_{t+1})$ appearing in equation (1.1) has been replaced by the term $max_a(s_{t+1}, a)$ in equation (1.2). This could seem equivalent if the policy followed was gluttonous (we would then have $a_{t+1} = Argmax_a q\ (s_{t+1}, a)$. However, given the need to achieve a compromise between exploration and exploitation, this is usually not the case. It therefore appears that the SARSA algorithm performs the updates according to the actions actually chosen while the Q-Learning algorithm performs the updates according to the optimal actions even if it is not these optimal actions that the agent performs, which is simpler. [21]

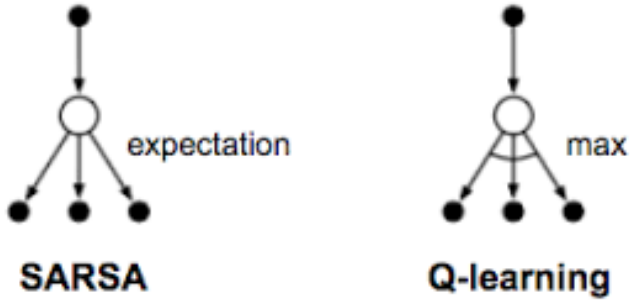


Figure 11:The backup diagrams for Q-learning and Expected Sarsa

5.3 The TD(λ), Sarsa(λ) and Q(λ) algorithms

The TD(0), SARSA and Q-Learning algorithms have the defect of updating only one value per time step, namely the value of the state that the agent is currently visiting. As it appears in Figure (3 .7), this update procedure is particularly slow. Indeed, for an agent having no a priori information on the structure of the value function, at least n successive experiments are required for the immediate reward received in a given state to be propagated to a state remote from the first of n transitions. While waiting for the result of this propagation, as long as all the values are identically zero, the behavior of the agent is a random walk

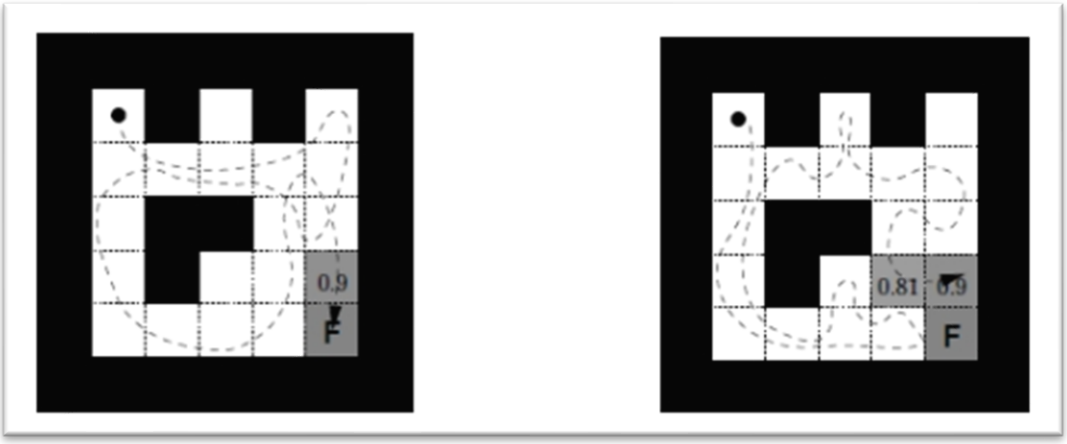


Figure 12:Navigation of an agent in a parts environment using the Q-Learning algorithm

Being initially zero, the propagation of non-zero values take place only once the agent has found the reward source for the first time and progresses only one step with each try of the agent.

One way of improving this state of affairs consists in providing the algorithm with a memory of the transitions carried out during an experiment in order to carry out all the possible propagations at the end of this experiment. This memory of the transitions carried out previously is called an eligibility trace. Thus, Sutton and Barto have proposed a class of algorithms called "TD(λ)" which generalize the TD(0) algorithm in the case where the agent has a memory of the transitions. Later, the SARSA and Q-Learning algorithms were generalized to SARSA(λ) and Q (λ), the second having been generalized in two different ways by two different authors. [21]

6 Conclusion

In this chapter, we have presented an overview of the deep learning algorithms, several techniques are used for solving the problem of reinforcement learning and the acquisition of optimal behavior in an environment Because of all these characteristics reinforcement learning is a method particularly suitable for robotics. This thesis therefore focuses on reinforcement learning for solve some problems of autonomous navigation of a mobile robot.

CHAPTER IV

IMPLEMENTATION RESULTS AND DISCUSSION

Introduction

In this chapter, we present the implementation of the robot model developed within the framework of this project, as well as the results obtained through different simulations. The main objective is to demonstrate the effectiveness of the reinforcement learning algorithm, in particular Q-learning, in the autonomous navigation of a robot in an environment containing obstacles. We first describe the robot components and the sensors used, before detailing the simulation scenarios, the development environment, as well as the Python libraries used. Finally, an analysis of the results from the different phases of exploration and exploitation is carried out, making it possible to evaluate the performance and the convergence of learning.

1 Simulation Environment

Visual Studio Code, often referred to as *VS Code*, is a free source code editor developed by Microsoft. It supports writing, editing, and running code in various programming languages such as Python, JavaScript, C++, HTML/CSS, and many more. It is known for being lightweight, highly customizable through extensions, and widely adopted by developers due to its powerful features like autocompletion, debugging, and project management.
In our project, we used VS Code to program the simulation using Python.

Python is a simple yet powerful interpreted programming language created in the late 1980s. Its clean and readable syntax makes it easy to learn, even for beginners. Python is widely used in various fields such as web development, data science, artificial intelligence (AI), automation, and more. One of its major strengths is its large community and the availability of numerous libraries.

In our case, we used it for AI-based learning with libraries that support this domain. Libraries used:

- ✓ Pygame:

Pygame is a Python library designed for developing 2D video games and multimedia applications. It allows the display of images and shapes, playing sounds and music, handling keyboard, mouse, and joystick events, and creating smooth animations. It is commonly used in educational projects as it does not require advanced graphics knowledge.

- ✓ sys: The sys library in Python interacts with the Python interpreter and the operating system to control the program's behavior. Common use include:

- ✓ sys.argv: reading command-line arguments
- ✓ sys.exit(): manually exiting a program
- ✓ sys.path: managing module import paths
- ✓ sys.version: checking the Python version being used in short, sys is a practical library for managing program execution and system parameters.
- ✓ random:

  The random library is a standard Python module used to generate random numbers and make random choices.
- ✓ Numpy as np:

  NumPy is a popular Python library for scientific and numerical computing. It allows for efficient manipulation of large numerical datasets, especially through arrays (also known as matrices).
- ✓ matplotlib.pyplot:

  Used to create graphs and data visualizations, matplotlib.pyplot is an essential tool for analyzing and presenting results in Python-based simulations.

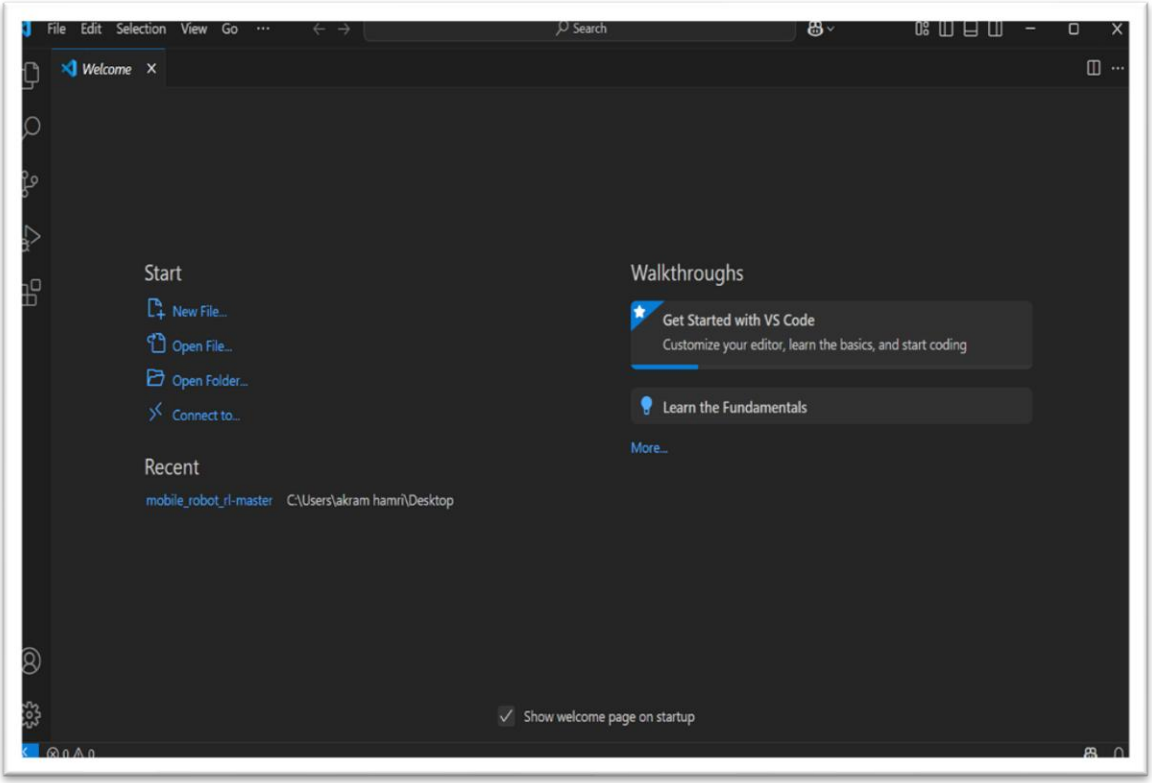


Figure 13:Visual Studio Code Welcome Visual Studio Interface Description

2 Implemented Algorithms:

2.1 Q-learning

Q-learning is a model-free reinforcement learning algorithm used to learn the value of an action in a particular state.

It finds the action-value function **Q** by interacting with the environment. Once the **Q** function is determined, we can achieve the optimal policy by selecting the action that provides the maximum expected utility (reward).

This is done by storing the rewards received in a two-dimensional table (Q-table), where in our case, there are 4 actions and 100 states.

State:

- **grid width**: environment width // grid size
- **grid height**: environment height // grid size

- **state**: grid width * grid height

Action:

- Right
- Left
- Up
- Down

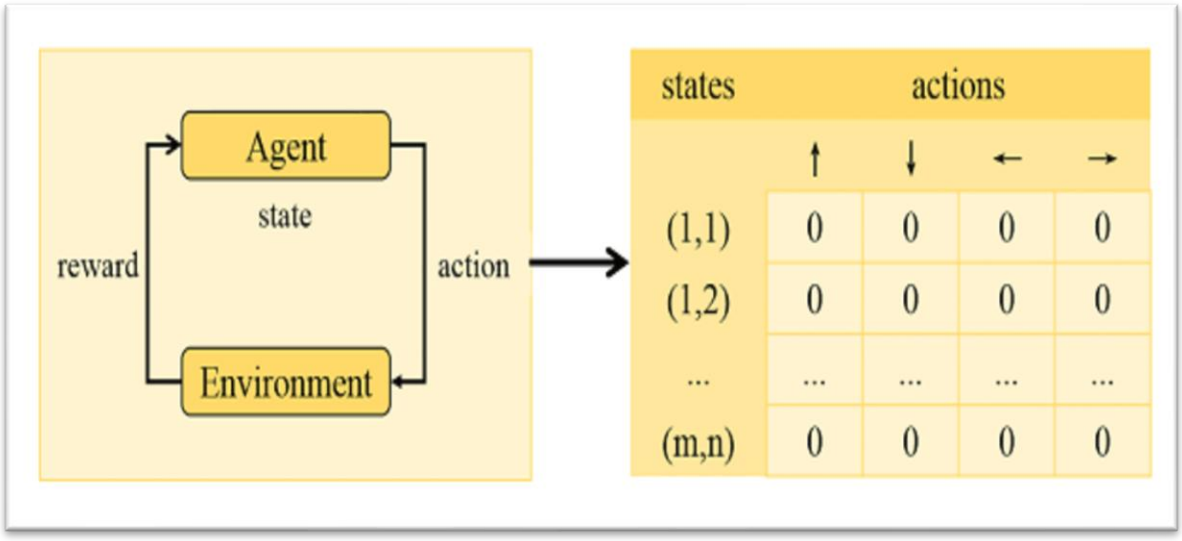


Figure 14:Form of Q table

Algorithm execution:

Initialize Q-table with zeros

Set learning rate (α), discount factor (γ), and exploration rate (ε)

While (Q-table not converged) {

Reset environment to a random initial state

For each episode (until goal state is reached) {

While current state ≠ goal state:

1. Choose action (a) for current state (s):

- With probability ε: Select random action (exploration)

- Else: Select action with max Q-value (exploitation)

2. Execute action (a), observe reward (r) and next state (s')

3. Update Q-table:

$Q(s, a) \leftarrow Q(s, a) + \alpha * [r + \gamma * \max(Q(s', a')) - Q(s, a)]$

4. Transition to next state:

$s \leftarrow s'$

}

}

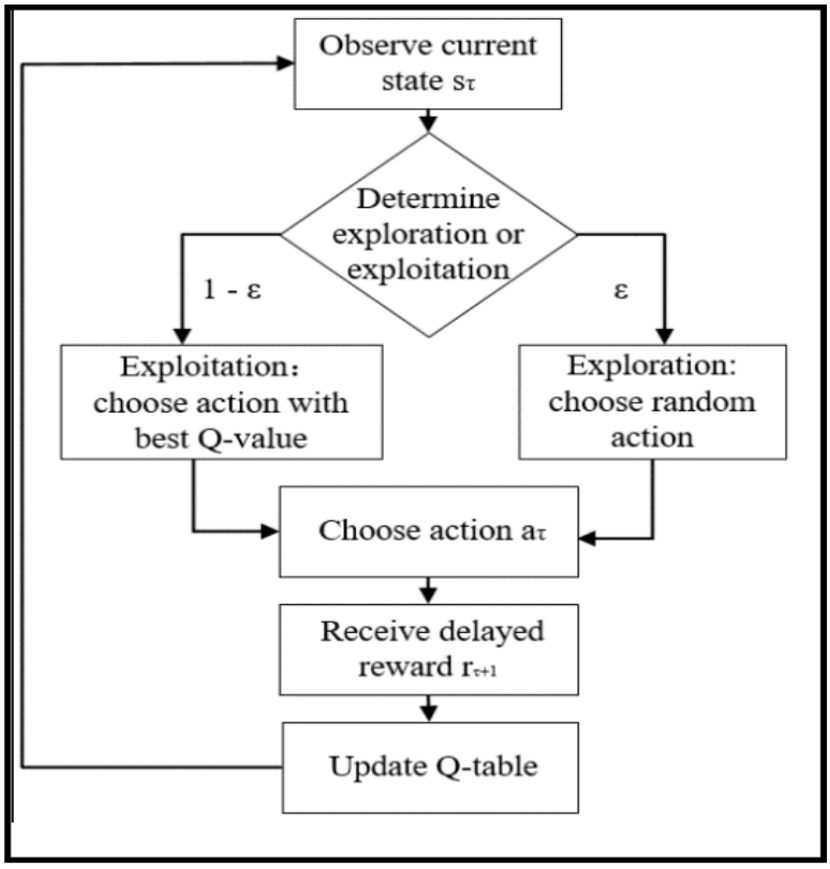


Figure 15:Q-Learning Flowchart

3 PRSENTATION OF THE REBOT MODEL

3.1 Sensors and Kinematics:

First, the robot gathers information through its sensors for navigation:

1. Front sensor
2. Rear sensor
3. Right sensor
4. Left sensor

Additionally, the robot moves from one state to another with a rotation angle of **90 degrees**.

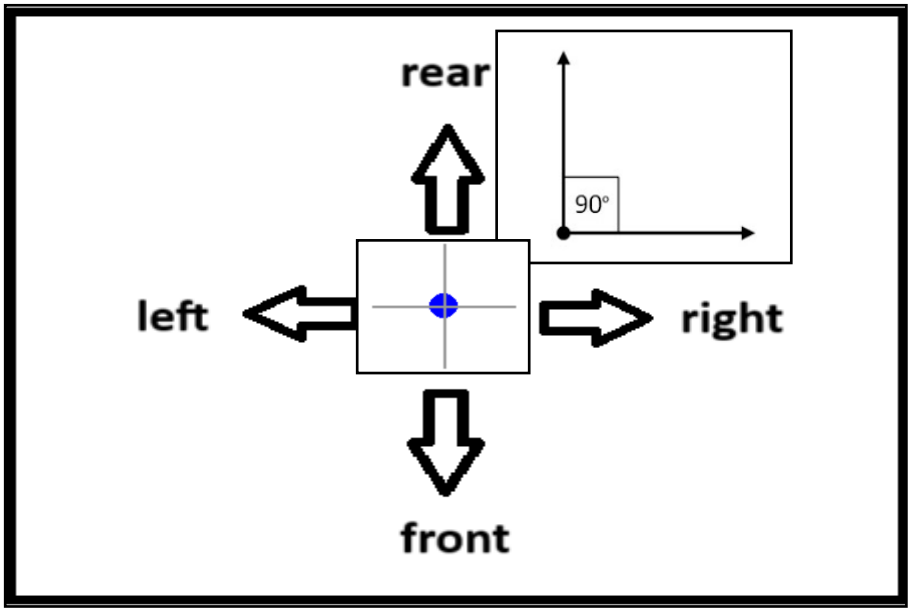


Figure 16:Robot Paths

The robot starts at its initial position and determines its current state (position in the grid). It chooses an action based on the value of -ε-:

- ✓ If random-unity < ε is exploration (random action)
- ✓ If random-unity> ε: exploitation (choosing the best move according to the Q-table)
    - It receives a reward:
- ✓ +100 if it reaches the target
- ✓ -100 if it collides

- ✓ -1 for each normal step

4 Environment modeling

4.1 Obstacles

There are two types of obstacles in this simulation:

- ✓ Circle
- ✓ Square

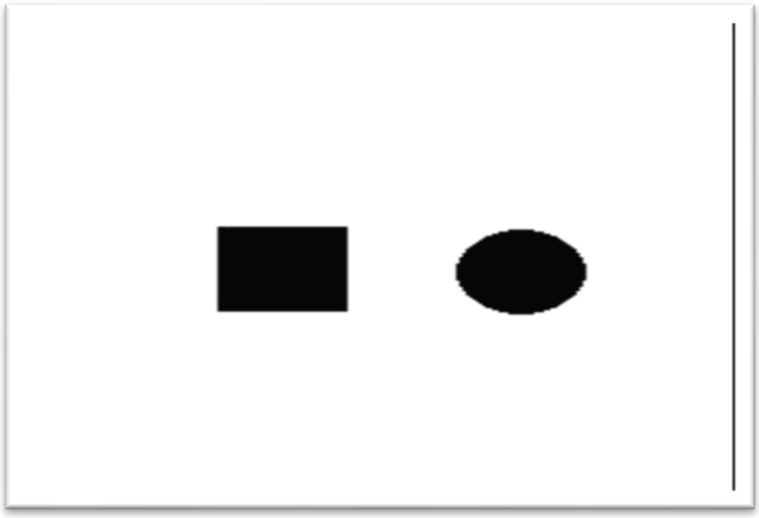


Figure 17:Type of obstacles

4.2 The interface description

In our program we have a main menu this menu have tow button

- ✓ Button one normal Learning
- ✓ Button tow Learning with episode

Figure 18:Main menu

When you press in the button (normal learning) one It shows you this interface

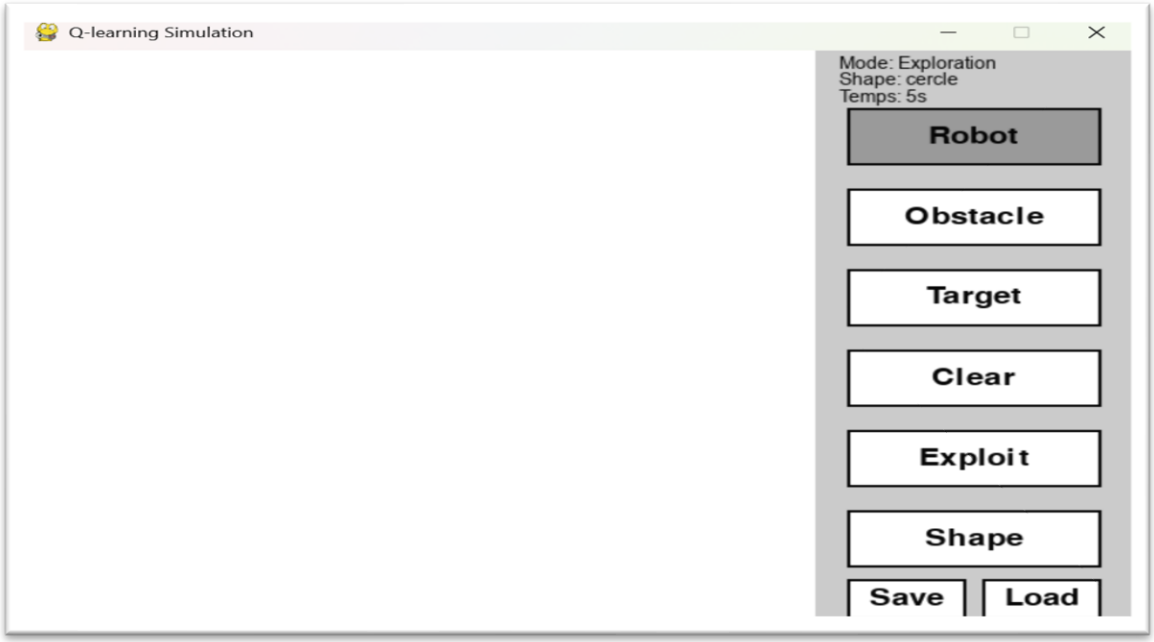


Figure 19:Interface of normal learning

- ✓ Button Robot: to place the robot
- ✓ Button Obstacle: to place the obstacle
- ✓ Button Target: to place the target
- ✓ Button Clear: to clear all environment
- ✓ Button Exploit: so that the robot just does the exploitation
- ✓ Button Shape: to change the form of obstacle
- ✓ Button Save: to save environment
- ✓ Button Load: to load environment

4.3 Simulation Scenarios

4.3.1 Exploration with Exploitation

The robot and obstacles are placed, then the goal is set. The robot starts navigating to reach the goal:

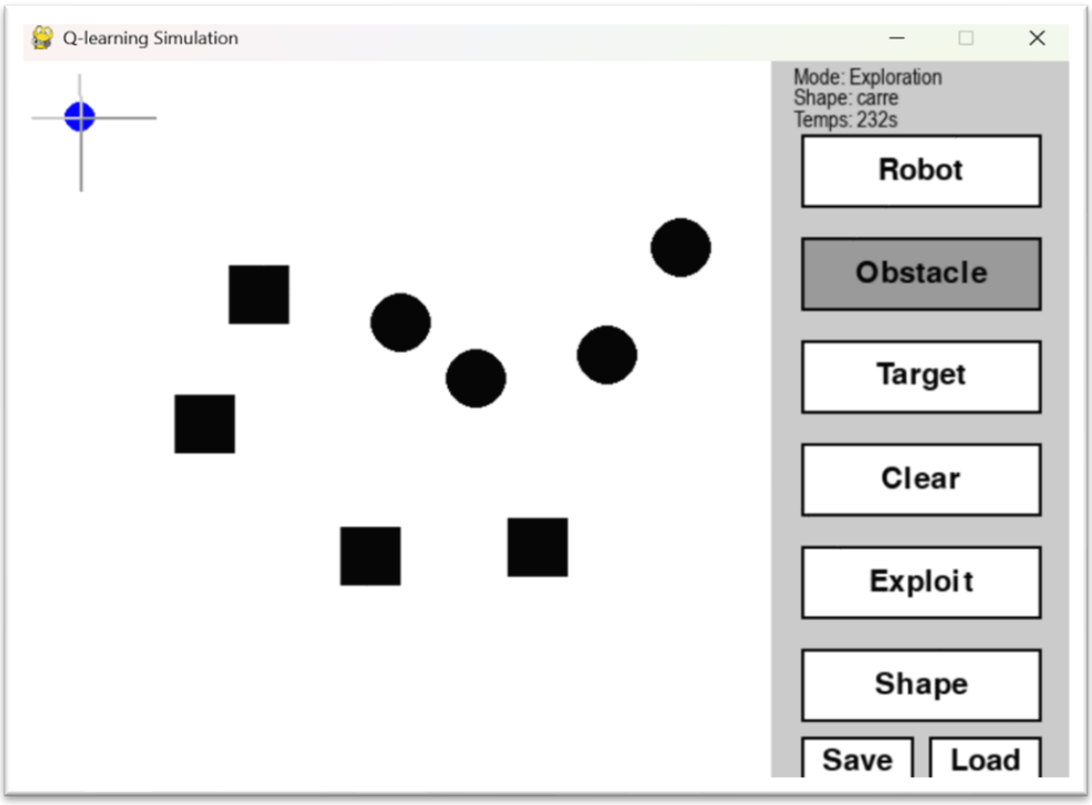


Figure 20:Before placing the goal

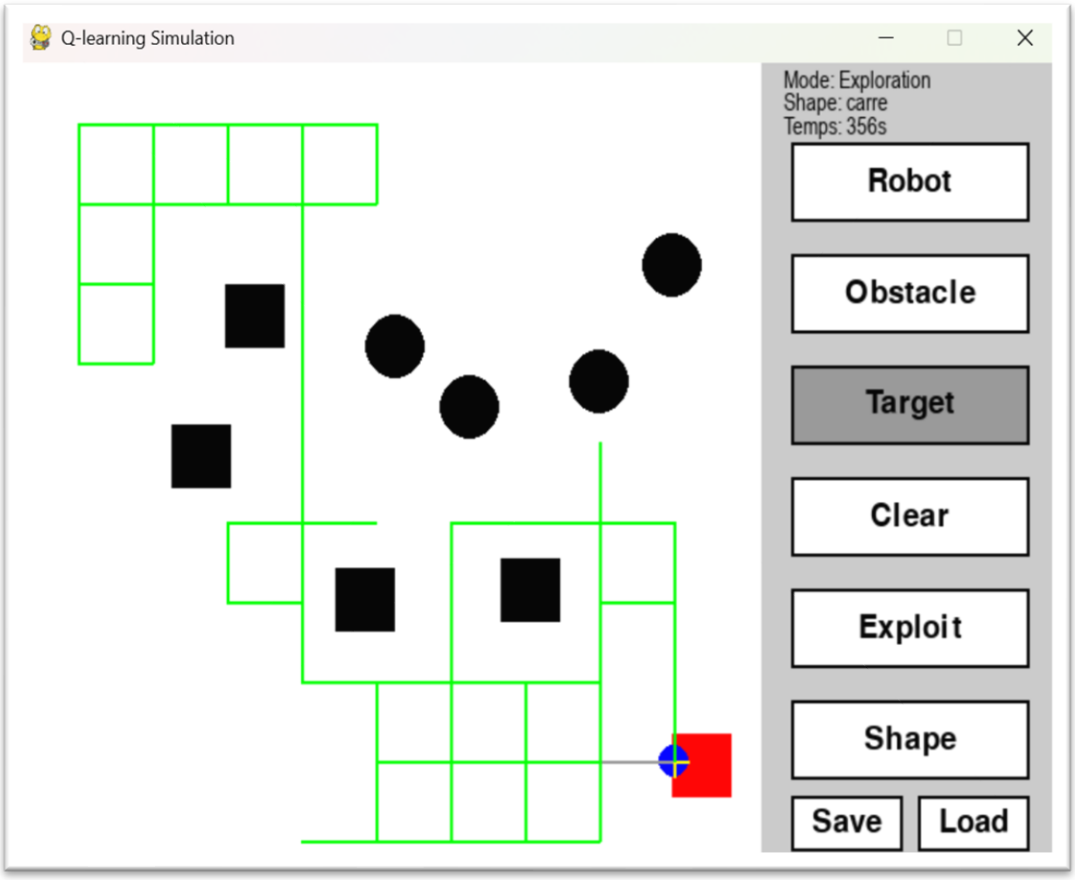


Figure 21:After placing the goal

4.3.2 Exploitation

a. Simple Exploitation

To do this, we perform at least one exploration phase first. Then, we apply exploitation, which can be repeated multiple times in the same environment (same obstacle and target positions) until the shortest path is found (maximum reward).

Below is the sequence of steps:

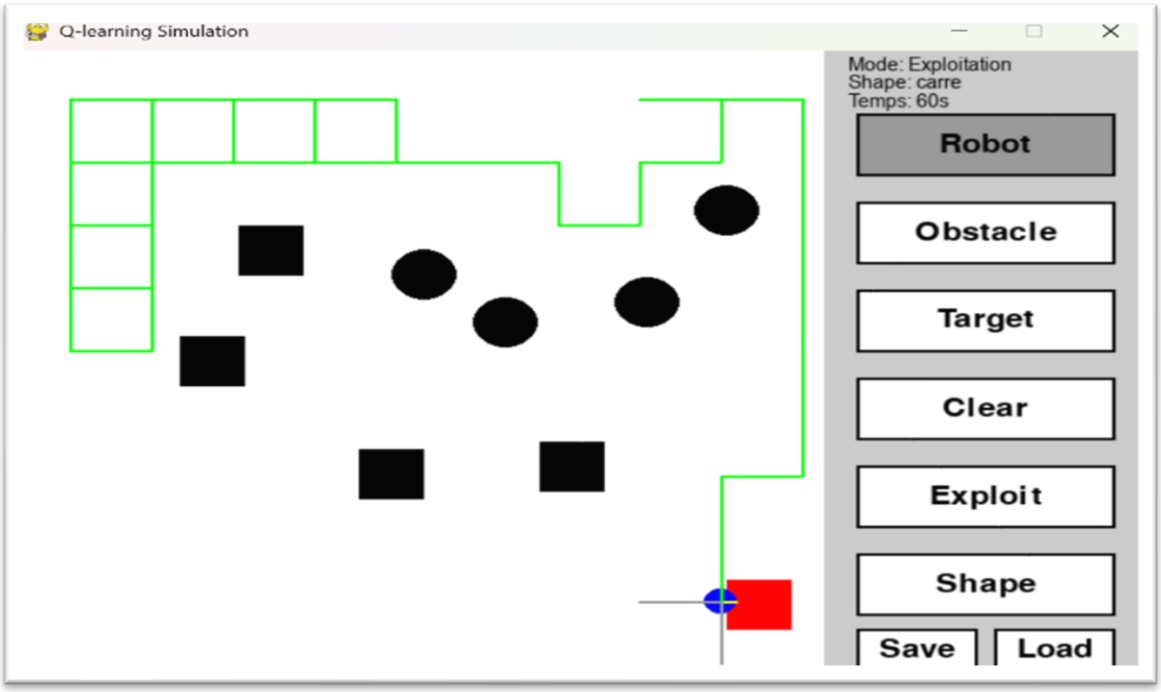


Figure 22:First exploitation after exploration

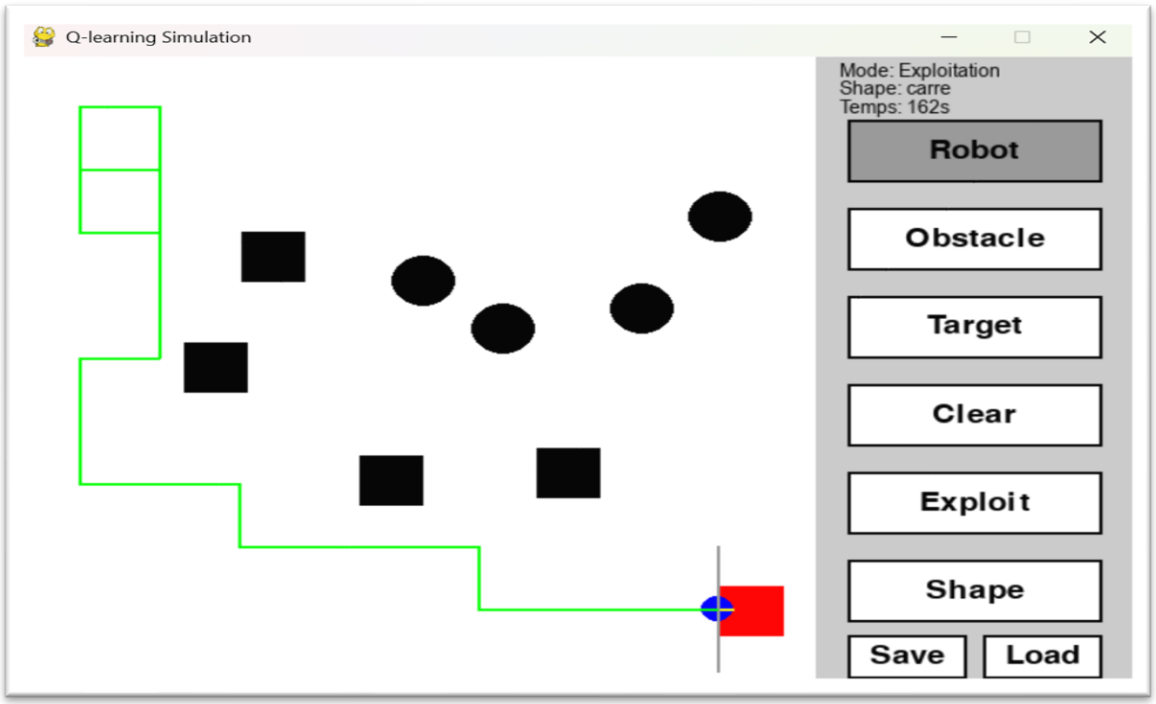


Figure 23:After a few trials (around 4 to 5))

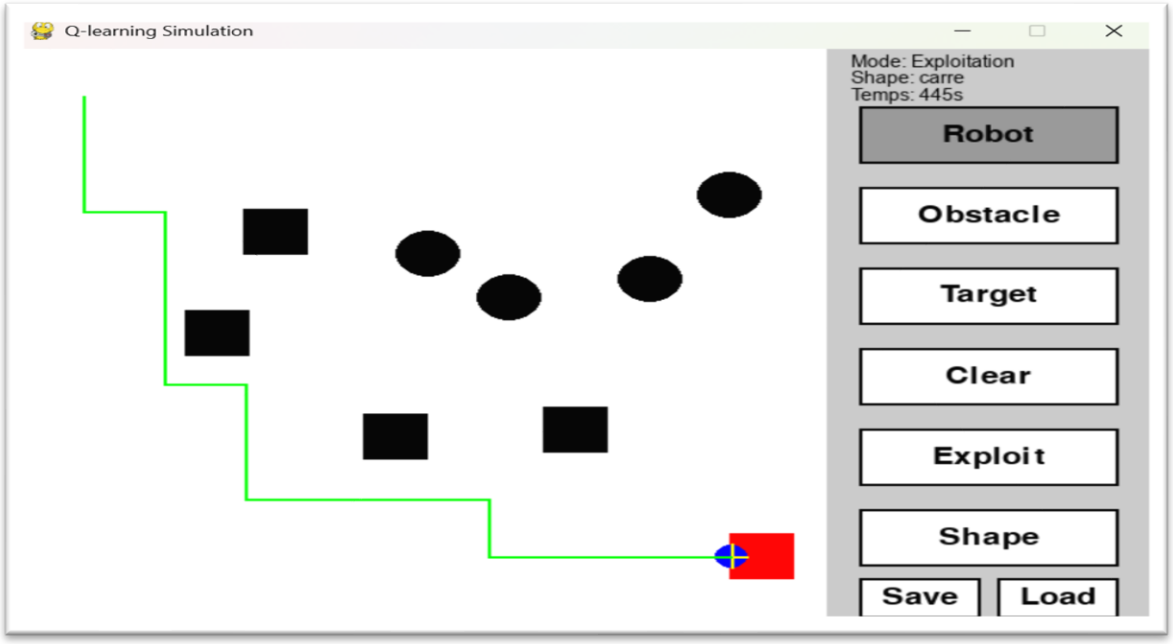


Figure 24:The shortest path (25 steps or more) is repeated

b. Exploitation with Episodes

We run a trial of 1000 episodes, which takes place in the background. Only the final state is shown when the goal is reached, along with a graph displaying the reward per episode.

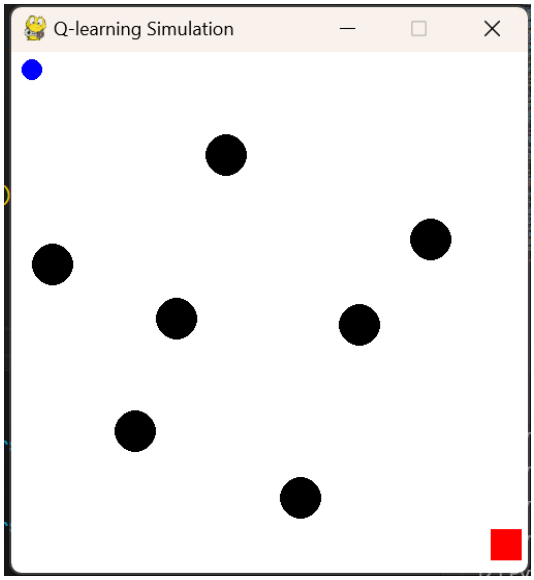


Figure 25:Determine the environment

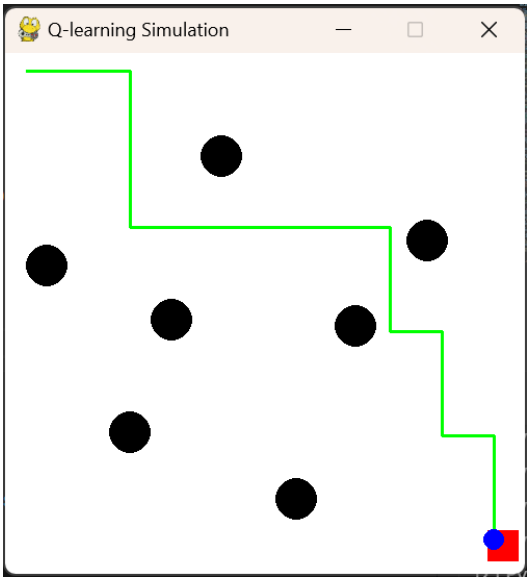


Figure 26:Reach the goal

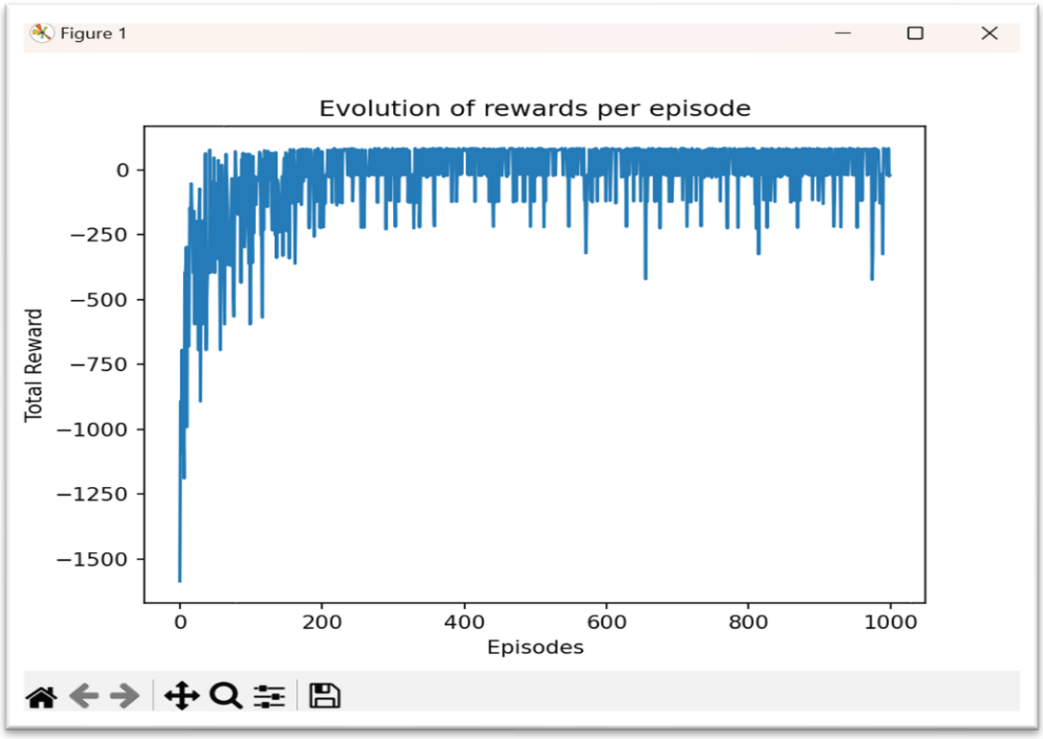


Figure 27:Graph of reward

In this graph, we observe the rewards over the episodes. We can see that the rewards are initially negative and tend to approach zero, which indicates that the robot's learning improves from one episode to another until it becomes approximately constant. This reflects the robot's convergence to the best path toward the target, with the highest possible reward**.**

On the program we before display the Q-table each time choosing an environment

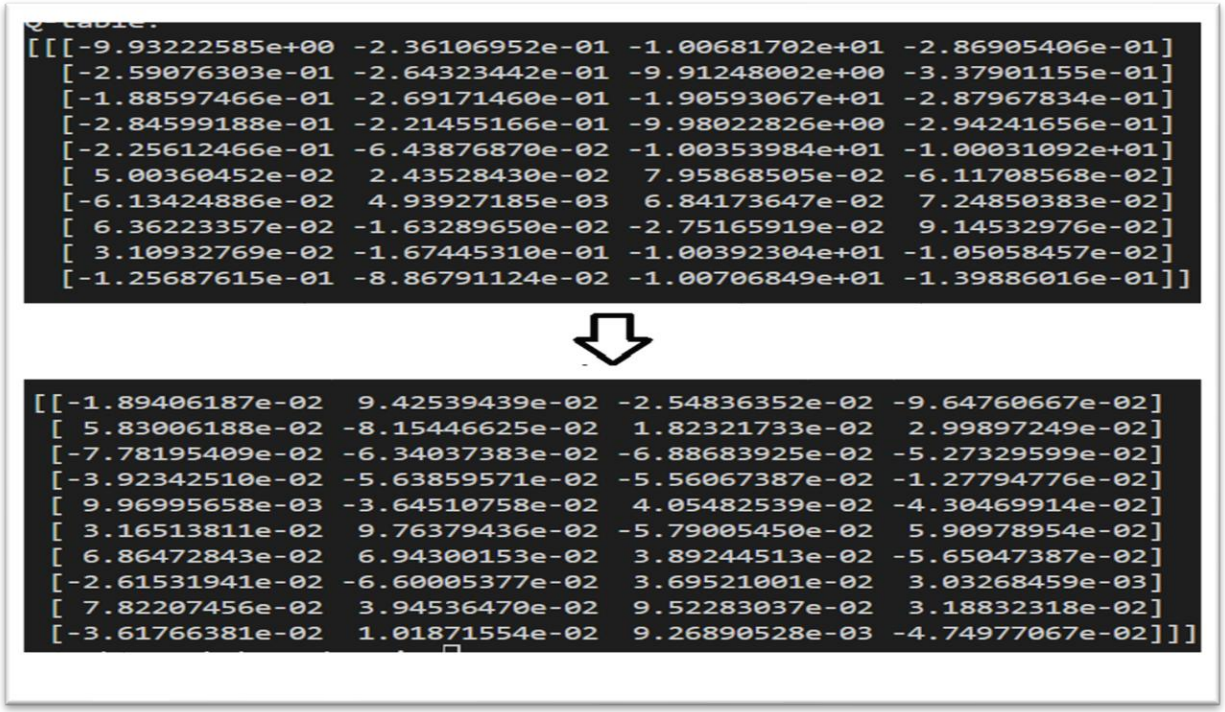


Figure 28:Q-table for an environment of our choices

5 The Q-learning process on the environment

We have chosen a specific placement of the robot, obstacles, and the target (as shown in the image above) to explain the execution of the navigation algorithm (Q-learning).

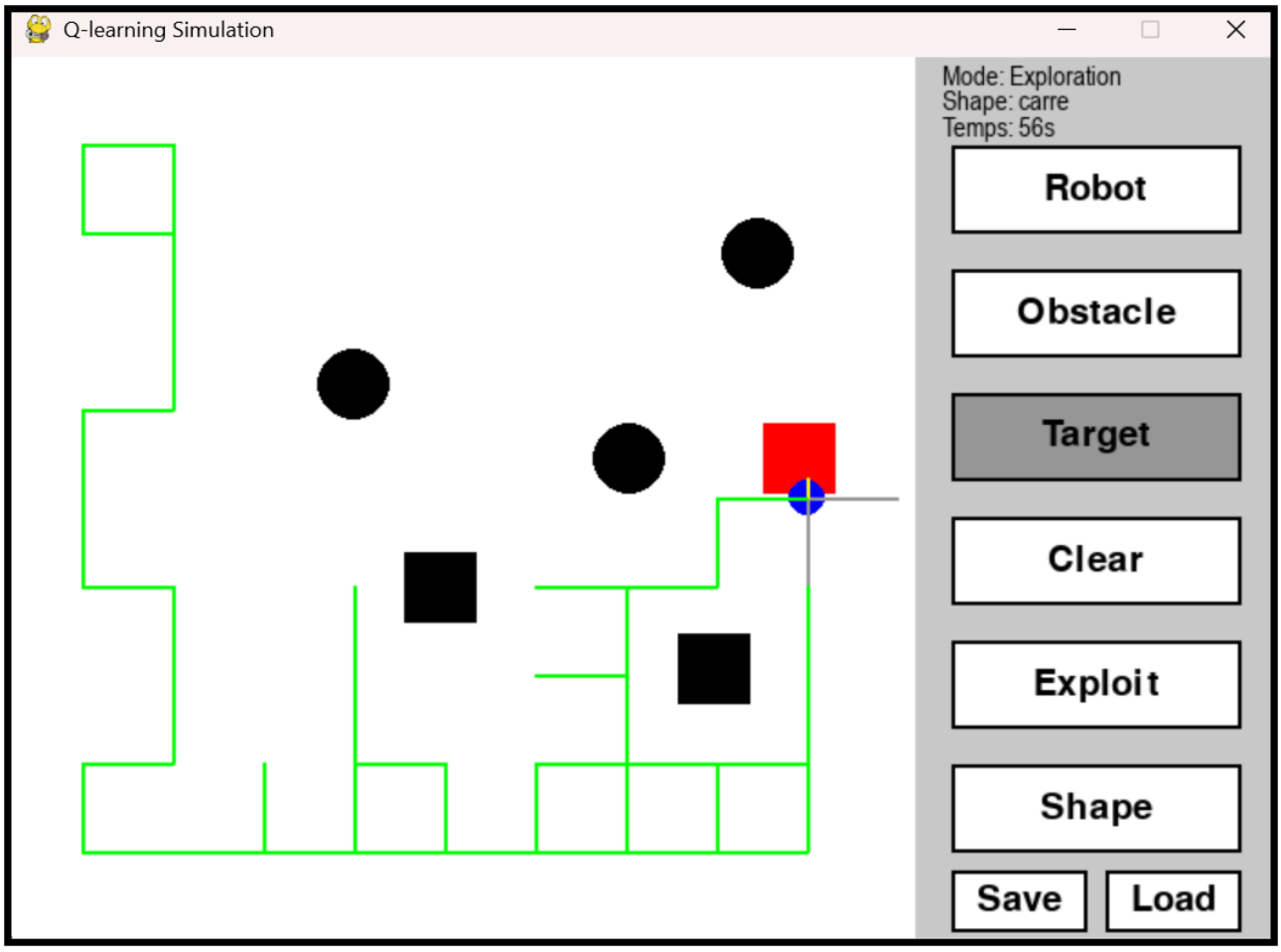
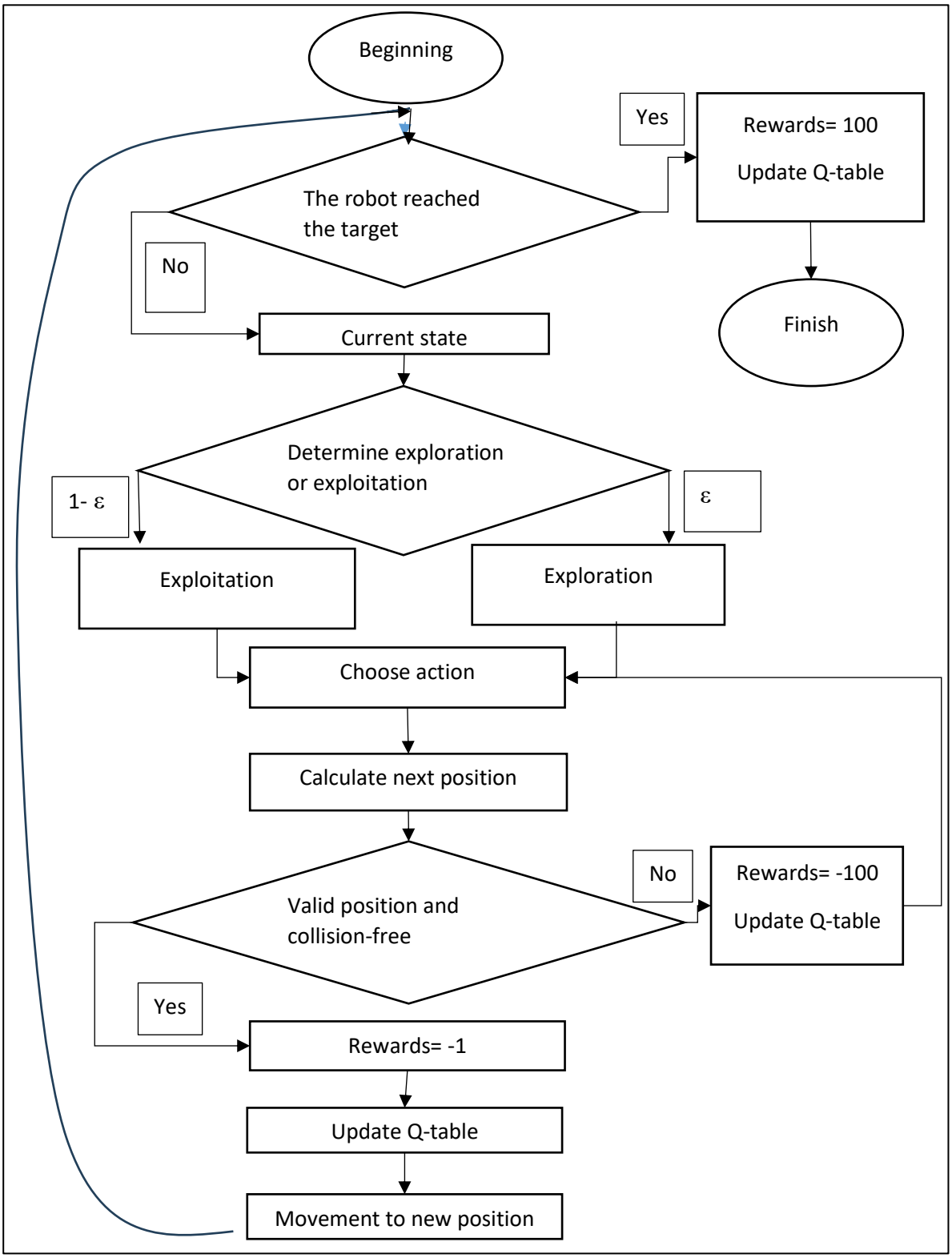


Figure 29:Navigation using the Q-Learning algorithm

In this case, the robot has two options. The robot makes a decision using epsilon (ε)-greedy strategy. In the application, ε = 0.8. This value is compared with a random number generated using a uniform distribution (between 0 and 1), resulting in:

- **Exploration (80%)**: The agent tries new actions to discover better long-term strategies (takes random actions).
- **Exploitation (20%)**: The agent uses its current knowledge (the Q-table) to make what it believes is the best choice.



robot navigation flowchart with Q-learning

6 Conclusion

This chapter highlighted the capabilities of a robot to learn to navigate effectively in an environment full of obstacles thanks to the Q-learning algorithm. The various simulations have demonstrated a progressive improvement in the robot's strategy, visible in particular by the optimization of the path taken and the increase in rewards. The use of adapted Python libraries, combined with a user-friendly development environment such as Visual Studio Code, facilitated the implementation and visualization of the results. This work thus confirms the relevance of reinforcement learning in autonomous navigation systems, while paving the way for future improvements such as the integration of more complex algorithms or adaptation to dynamic environments.

General Conclusion

At the end of this study, we demonstrated the relevance and potential of reinforcement learning, in particular Q-Learning, in the field of autonomous robotics. This approach, based on the principle of learning by interaction with the environment, allows robots to acquire optimal behaviors without explicit supervision, by maximizing a reward function through experience.

Our work has highlighted the advantages of Q-Learning, in particular its simplicity of implementation, its robustness in discrete environments, as well as its ability to generate effective action policies from autonomous exploration. We have applied this method to concrete robotic scenarios, thus demonstrating that a robot can, through progressive learning, learn to solve complex tasks such as navigation, obstacle avoidance or decision-making in a dynamic environment.

However, this research has also revealed several inherent limitations of Q-Learning when applied to continuous, noisy or large-dimensional environments. The size of the space of states and actions, the compromise between exploration and exploitation, as well as the speed of convergence represent as many challenges that had to be overcome, in particular through techniques such as discretization, the use of intelligent exploration strategies, or the use of approximation functions.

Despite these obstacles, our experimental results confirm that Q-Learning can be a powerful tool for developing autonomous behaviors in robots, provided that its implementation is intelligently adapted to the specific robotic context. In addition, this research paves the way for promising prospects: Deep Q-Learning, which combines Q-Learning and neural networks, would make it possible to significantly expand learning capabilities in more complex and more realistic environments.

Finally, this thesis contributes to the efforts of the scientific community to bring the theory of machine learning and robotic practice closer together. By integrating the Q-Learning algorithm into real robotic systems, we are participating in the creation of more intelligent machines, capable of adapting, learning and evolving without constant human intervention. This work thus marks an important step towards more autonomous, adaptive and collaborative robots, which in the future will be able to integrate harmoniously into our daily lives, our industries, and even our living spaces.

Bibliographic:

[1] U. Nehmzow, Mobile Robotics: A Practical Introduction, 2003

[2] Dictionnaire Hachette, Hachette Éducation, 2013. ISBN : 978-2-01-281493-6.

[3] O. Khatib, “Real-Time Obstacle Avoidance for Manipulators and Mobile Robots,” Int. J. Robot. Res., vol. 5, no. 1, pp. 90–98, 1986.

[4] Fédération International de Robotique, Robotics – World Robotics, 2006.

[5] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, Introduction to Autonomous Mobile Robots, 2nd ed., MIT Press, 2011.

[6] B. Siciliano and O. Khatib, Eds., Springer Handbook of Robotics, 2nd ed., Springer, 2016.

[7] R. R. Murphy, Introduction to AI Robotics, 2nd ed., MIT Press, 2019.

[8] S. Thrun, W. Burgard, and D. Fox, Probabilistic Robotics, MIT Press, 2005.

[9] G. A. Bekey, Autonomous Robots: From Biological Inspiration to Implementation and Control, MIT Press, 2005.

[10] A. Pruski, Robotique mobile, la planification de trajectoire, Paris: Hermès, 1996.

[11] L. Jetto, S. Longhi, and D. Vitali, “Localization of a wheeled mobile robot by sensor data fusion based on a fuzzy logic adapted Kalman filter,” Control Eng. Pract., vol. 7, pp. 763–771, 1999.

[12] C. Cappelle, M. El Badaoui El Najjar, D. Pomorski, and F. Charpillet, “Détection, suivi et géolocalisation d'obstacles…,” Conf. Int. Francophone Automatique (CIFA'08), Bucarest, Roumanie, 3–5 sept. 2008.

[13] Villani, Mission Intelligence Artificielle, Mar. 2018.

[14] I. Zara, L'intelligence artificielle : principes, outils et objectifs, Mémoire de Master, 2019.

[15] M. Nadour, Navigation Visuelle d’un Robot Mobile, Thèse de Doctorat, 14 oct. 2020.

[16] M. Wazan, Deep Learning: Principles, Concepts and Methods.

[17] A. Benmakhlouf, Navigation Intelligente Autonome Pour Robot Mobile, Thèse de Doctorat, 5 oct. 2023.

[18] A. Dilmi, Apprentissage par renforcement en utilisant les réseaux de neurones artificiels, Mémoire de Master, juin 2012.

[19] L. Cherroun, Navigation Autonome d'un Robot Mobile par des Techniques Neuro-Floues, Thèse de Doctorat, 22 mai 2014.

[20] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2018.

[21] S. Filippi, Stratégies optimistes en apprentissage par renforcement, Thèse de Doctorat, 24 oct. 2010.

[22] O. Buffet, Apprentissage par renforcement dans un système multi-agents, DEA, 11 juil. 2000.

[23] L. Cherroun, "Using Q-Learning and Fuzzy Q-Learning Algorithms for Mobile Robot Navigation in Unknown Environment," *Article*, mars 2012.