Deep Reinforcement Learning for Complete Coverage Path Planning in Unknown Environments

Omar BOUFOUS

School of Science

Thesis submitted for examination for the degree of Master of Science.
Espoo 29.09.2020

Thesis supervisor:

Prof. Mihhail Matskin

Thesis advisor:

Prof. Florian T. Pokorny

Aalto University
School of Science

Aalto University
School of Science

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi

Abstract of master's thesis

Author Omar BOUFOUS

Title of thesis
Deep Reinforcement Learning for Complete Coverage Path Planning in Unknown Environments

Degree Master of Science, ICT Innovation

Degree programme Autonomous Systems

Thesis advisor(s) Florian T. Pokorny, Alexandre Pascault

Year of approval 2020 **Number of pages** 60+2 **Language** English

Abstract
Mobile robots must operate autonomously, often in unknown and unstructured environments. To achieve this objective, a robot must be able to correctly perceive its environment, plan its path, and move around safely, without human supervision. Navigation from an initial position to a target location has been a challenging problem in robotics. This work examined the particular navigation task requiring complete coverage planning in outdoor environments. A motion planner based on Deep Reinforcement Learning is proposed where a Deep Q-network is trained to learn a control policy to approximate the optimal strategy, using a dynamic map of the environment. In addition to this path planning algorithm, a computer vision system is presented as a way to capture the images of a stereo camera embedded on the robot, detect obstacles and update the workspace map. Simulation results show that the algorithm generalizes well to different types of environments. After multiple sequences of training of the Reinforcement Learning agent, the virtual mobile robot is able to cover the whole space with a coverage rate of over 80% on average, starting from a varying initial position, while avoiding obstacles by using relying on local sensory information. The experiments also demonstrate that the DQN agent was able to better perform the coverage when compared to a human.

Keywords
Deep Reinforcement Learning, Autonomous Robots, Path Planning, Coverage, Control, Robot Vision

Acknowledgements

First of all, I would like to warmly thank Mr. Alexandre Pascault, my internship tutor and manager of the Innovation Lab, for the technical help he provided throughout the entire project, and his invaluable comments and guidance. He paid close attention to the progress of my work. This project would not have been possible without his support. I would also like to thank Mr. Aziz Amal, director of the Astek Innovation Lab for welcoming and supporting me throughout my internship. Special thanks to him for his great management of the remote work situation in the unusual context of the pandemic and all the additional challenges it brought.

I would like to thank all the members of the team; Mathis in particular, for helping me integrate the team of interns at the company and making me comfortable from the very beginning. I thank also all the other trainees of the Astek Innovation Lab for having maintained a pleasant and studious atmosphere at the Research Lab during the month we spent together.

I would like to express my greatest gratitude to my academic supervisors Florian T. Pokorny and Robert Gieselmann, for their answers to my questions and all support they have given me during the master thesis.

Finally, many thanks to my examiner Professor Mihhail Matskin for his interest in the master thesis project and for taking the time to give me feedback and suggestions to improve the quality of my work.

Contents

List of Figures

List of Tables

List of Abbreviations

AC	**A**ctor-**C**ritic
AI	**A**rtificial **I**ntelligence
ANN	**A**rtificial **N**eural **N**etwork
CNN	**C**onvolutional **N**eural **N**etwork
CCPP	**C**omplete **C**overage **P**ath **P**lanning
DRL	**D**eep **R**einforcement **L**earning
DNN	**D**eep **N**eural **N**etwork
DQN	**D**eep **Q**-**N**etwork
FFN	**F**eed **F**orward **N**etwork
ML	**M**achine **L**earning
MC	**M**onte **C**arlo
MDP	**M**arkov **D**ecision **P**rocess
NN	**N**eural **N**etwork
POMDP	**P**artially **O**bservable **M**arkov **D**ecision **P**rocess
RL	**R**einforcement **L**earning
RMSE	**R**oot **M**ean **S**quare **E**rror
RNN	**R**ecurrent **N**eural **N**etwork

Chapter 1

Introduction

This chapter introduces the problem that this thesis project addresses, the context of the problem, the goals of this study, and outlines the structure of the report.

1.1 Background

The use of robots in various fields such as in the military, for firefighting, or more commonly for domestic use has grown considerably in the past few decades. Recent and high-end robots include auto-navigation systems that introduce mapping technologies and different techniques for building and maintaining map-like representation of the workspace, instead of navigating in their environment by using a more or less ”random” approach while taking into account given constraints [1]. Robots that successfully navigate in their space, are able to so with the help of various types of sensors such as GPS, LiDARs, ultrasonic sensors and sometimes a camera as the primary means of detection and localization. In fact, a common approach to the navigation tasks, used for instance in some UAVs and sweeping robots, usually involves the use of measurement data to *(i)* localize the robot in the environment and *(ii)* identify relevant characteristics of the environment such as the position of obstacles or other objects that may be of interest. Therefore, the autonomous agent relies on the data it collects to try to extract useful information from the environment which serve as a basis for decision-making.

In general, the objective when designing autonomous and intelligent mobile systems is to develop smart robots capable of gaining and using knowledge and/or experience acquired from the analysis of collected data to improve their navigation techniques and their behavior. In this context, it is essential that robots learn not only from the processing of sensors input data, but also from the consequences of their actions. In this regard, trajectory and motion planning in static and known environments is a well-defined and known problem in the literature on robotics and control [2], [3]. It has been the subject of in-depth research in particular with a focus on the energy efficiency which is critical for robots; see, for example, [4], [5] and references therein. The introduction of Machine Learning techniques unlocked several technological barriers and is at the origin of significant advances in missions involving navigation or motion planning. Complete Coverage Path Planning is a particular case of those missions where an area needs to be covered entirely such as in cleaning, lawnmowing, demining or even painting. Unlike other activities that have been vastly automated, the majority of complete coverage operation are still nowadays accomplished manually. This is due to

the fact that there aren't solutions that completely fulfill the needs of the users.

In fact, some of the current strategies for controlling and guiding the robot to accomplish complete coverage are particularly demanding in terms of computation time [6] and are sensitive to noise and uncertainties in sensors data. Indeed, the control and planning of the trajectory often requires the exploration of a large an potentially infinite space of solutions. However, Machine Learning in general and Reinforcement Learning in particular which allows to represent robots as software entities, opened new possibilities to enhance the strategies of the control of robots. Motion planning can be performed for relatively complex situations, and acceptable trajectories can be derived in reasonable time.

1.2 Problem

In this thesis, the autonomous navigation problem in an unknown structured environment is first modeled a Markov Decision Process and the challenge is to solve it in the form of a discrete-time control problem using deep reinforcement learning so that the coverage task can be automated. The method should enable the agent to navigate from an arbitrary departure point and explore the entire space using solely local information provided by the embedded sensors informing the robot on its surroundings. In [7], an engine is developed to perform sensor-fusion and merge data coming from different types of sensors. Others techniques opted for rangefinders as in [8], but the use of cameras has increasingly become common [9], [10] in the literature. And in this work, given that the real robot is equipped with a stereo camera (see figures in the appendix), it is used as the main sensor to observe the space. The flow of 3D images is to be used to identify the vicinity of the robot and it will therefore serve as an input for the model to make decisions. The case study of the lawnmower is taken into consideration and we assume that the exact values of the state variables characterizing the real-time position of the robot can be obtained without noise. The real-time localization of the robotic system can also be supposed perfectly known as it can be obtained conveniently by either opting for an absolute position using technologies such as GPS or DGPS in outdoor environment for example, or choosing to track the movement with the help of an appropriate odometer as it has successfully been done in [11]. In this setting, the impact of the noise and uncertainties on sensor measurements and state observations of environment is not negligible but will be not be addressed in this work. This study was carried out with simultaneous map building and without offline path planning. This means that actions are taken on the fly and online.

In practice, before starting the coverage task it is hardly possible for the agent to have complete and perfect knowledge of the workspace it is placed in. For this reason, the main objective of this project is to design a motion controller to navigate a mobile robot, in an unknown environment such that the entire free space is explored using as few resources as possible, and is as little time as possible while avoiding collisions with obstacles.

Firstly, the space to be covered is represented with an occupancy grid map [12], which is a discrete map of the environment. Without loss of generality, we chose a rectangular shape and we divide the environment into cells with similar width and height and of comparablesize as the robot shown in Figure 1.1. Cells are

characterized by their type (or state) as well as their position as it was introduced in the preliminaries.

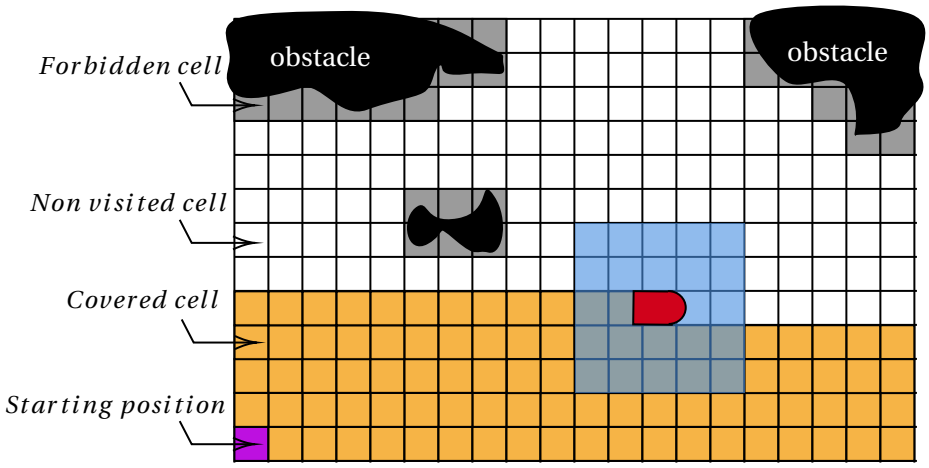


FIGURE 1.1: Top-view of a mobile robot performing complete coverage in an environment.

As the robot explores the unknown environment, a 2D occupancy grid map is generated using data extracted from the camera and the robot's odometry (for the robot position). Next, every cell inside the occupancy grid is classified. Each cell can be visited, non-visited, forbidden (or occupied), respectively depicted in orange, white and grey in Figure 1.1. Cells which are not in the range of the sensors are classified as unknown. Sensor measurements help identify obstacles by providing information on the state of the cells surrounding the robot, thus allowing the algorithm to update the map accordingly as the robot moves around.

In the base case scenario, the visibility of the robot at a given position is limited to the 25 cells that are immediate neighbors. This is shown in Figure 1.1. The area scanned by the robot is colored in blue and represents the neighbouring cells visible to the agent, included in the set $\mathcal{N}_i$.

1.3 Purpose

As mentioned earlier, studies tackling the problem of complete coverage path planning have not made the most of the progress in Deep and Reinforcement Learning especially regarding trajectory optimization and computational efficiency. For this reason, building autonomous robots making use of these available techniques deserves more attention, especially given the existence of plethora of problem formulations and possible improvements, as well as the applications to diverse robotic systems with limited memory, energy resources and computation capabilities. This is one of the key elements that motivate this thesis. The aforementioned studies treated different aspects of the coverage problem focusing for example on a cooperative solution involving several agents such as in [13] or [14]. More specifically, the most significant contributions to solving complete coverage tasks, that make use of the Deep Reinforcement Learning framework are rather recent and include for example [15] which focuses on an energy-efficient coverage under varying power constraints for a UAV, and [8], where the navigation task is formulated

as a POMDP and a DRL scheme was applied to derive a policy rapidly enabling the agent to reach a target in an unknown and complex environment. To the best of the author's knowledge, there are no studies that have applied a combination of DL and RL methodologies when dealing with CCPP in unknown environments for mobile robots. This is the second point that motivates this thesis. Moreover, as explained earlier, autonomous navigation in unknown environments from an initial position to a desired target without colliding with obstacles is a crucial task in robotics. But this work considers the special case navigation task of a coverage for a robot such as a lawnmower and aims at addressing this challenge thanks to a novel definition of the coverage problem. The work carried out during this thesis focuses on the problem of navigation in unseen and possibly dynamic environments, by means of sensor-based local and limited perception. This is the third element that motivates this thesis. Finally, in this thesis project, the purpose is to formulate the navigation problem in a way that makes it convenient to use DRL such that navigation strategies for the robot can be derived in reasonable time. This is a problem without an *a priori* model of the environment (model-free) and without a predefined strategy (off-policy) with the objective of designing an end-to-end online motion planner with the help of DRL.

Based on the objectives mentioned above, the aim of the master thesis can be summarized in the following research questions: how to perform complete coverage based on Reinforcement Learning and how efficient is it? Several underlying other questions are raised by this research question, such as how to model the environment? or how to avoid obstacles?

Note that the nature of this work is mainly exploratory, as it attempts to apply Deep Reinforcement Learning to automate coverage. This study does not claim nor strive to solve the problem of ground coverage. Instead, it provides an initial reflection upon which future work can be based.

1.4 Goals

In the long run, this study could serve as a first step and a basis for path planners that are based on RL and DRL especially those under computational or energetic constraints. This is particularly interesting for some type of applications and devices in robotics. As stated above, the goal is to be able to achieve sensor-based navigation in unknown areas with the help of the Reinforcement Learning framework in order to succeed in covering the ground with a good coverage rate which is evaluated with appropriately defined metrics. The comparative analysis and experiments allow to have an idea on how effective the deployed strategy is.

The simple problem formulation makes it possible to quickly generalize the solution concept to various domains with little changes or adjustments in the structure. This is made possible thanks to the mathematical tools used in the scope of this thesis.

1.5 Ethics and Sustainability

This work is in line with the Open Science principles, making sure any produced knowledge is accessible to everyone. Besides, this project was conducted thanks to free and open source tools and platforms. The data used for training the DRL model have been generated artificially. The code produced in the scope of this project, and associated with both the AI component and the simulation environment is online and available on the author's GitHub account[1,2] to share it with the scientific and research community and encourage collaboration. Moreover, the development of some components in this project required consequent computational resources (for instance, the training of the model often takes over 24h) which consume significant amounts of electricity. It goes without saying that their use has immediate and tangible consequences. Therefore, in order to minimize the carbon footprint of the entire project, special attention has been paid to the reasonable and intelligent management of said resources. For example, intermediate results are regularly saved during the training process so as to avoid restarting from the beginning when needed. From a moral and ethical perspective, this research project guarantees that it does not present results or theories that are not backed-up with scientific evidence. Nevertheless, the work carried out raises questions and concerns from an ethical point of view. Looking at it from a wider perspective, the proposed algorithms and the contributions exposed can be deployed in various systems as mentioned above. On the one hand, there is the issue of security since a human presence is often possible and even desirable for monitoring purposes when operating robotic systems. On the other hand, in a domestic environment, the use of such systems which implement sensors like a camera could violate the privacy of users. Therefore, this ethical question is of paramount importance when developing intelligent systems. In general, a responsible engineer or research scientist has the duty to ensure that the systems he participates in designing fulfill their mission without endangering the physical, social or moral integrity. This can only be done if the developed system is questioned in terms of its absolute usefulness and the benefit it brings to the user, as well as its sustainability. This can possibly be done as suggested in [16] by introducing safeguards upstream in order to define a framework for the proper functioning of the system. Nevertheless, one should be aware of the fact that every computer system is hackable, and as a consequence, it can be used for malicious and abusive purposes. It is a risk that comes with using this type of devices. This security aspect is critical for Intelligent Autonomous Systems which are very complex, but it is out of the scope of this master thesis, which is destined to present an approach to solve a technical and scientific problem in an academic context.

1.6 Research Methodology

In order to answer the research question, several elements must be taken into account. First, as exposed in [17], this project follows both the quantitative and

[1] https://github.com/omar-bfs/gym_maze
[2] https://github.com/omar-bfs/DQN

qualitative research methodology by applying a comparative and deductive approach. Given the nature of the studied problem, data is generated for the path planning part to simulate real world scenarios. Later, the data is processed and analyzed with mathematical and statistical tools, that are known to be reliable. This also allows the results to be reproducible. For the image analysis part, real images were used by the computer vision component, which backs up the validity of the method. Additionally, the results produced by the algorithms are designed and evaluated numerically. For this purpose, performance metrics are defined to set a baseline and an objective reference for comparison. And in order to assess the quality of the presented work, validation and verification are done with a study of the results in light of other methods as well as a discussion on the difficulty of deploying such a technique on a real physical system, particularly in terms of complexity.

Throughout this work, the link between theory and practice is constantly established whenever possible. This is particularly important since the program developed is an algorithm that is eventually destined to be used on physical entities. This also supports the validity of research. The training and testing data sets are created according to a specific distribution so that it corresponds to a certain reality. This gives consistency and reliability to the methodology and the results. Finally, conclusions are drawn from analysis of the results, as well as a comparative study.

1.7 Outline

The rest of the report is organized as follows; in chapter 2, we gives a brief overview of the Reinforcement Learning and the underlying concepts, in Markov Decision Processes in particular. The learning problem is introduced as well as the mathematical framework and tools used to implement the proposed Deep Reinforcement Learning algorithm. This section reviews other related technologies in the field of complete coverage path planning.

Chapter 3 defines the type of neural networks used in this study in relation with the Complete Coverage Planning problem as well as the assumptions made. This chapter also explains the choices made in terms of the structure of the proposed solution and the integration of different components.

Numerical results evaluating the performance of the proposed path planning technique are presented in chapter 4. Firstly, this section reveals intrinsic characteristics and the behavior of the agent before comparing it later to existing algorithms on the basis of chosen performance metrics. A second experiment is also presented in this part where the DQN competes against a human player to cover an area as rapidly as possible. This section is concluded with a discussion on the obtained results.

Finally, in chapter 5, we summarize this work and draw directions for future work, providing insights for possible improvements and potential approaches to extend this study.

Chapter 2

Background

Reinforcement Learning (RL) is a type of Machine Learning that is concerned with sequential decision making. A RL problem is composed of an agent and an environment. The RL agent interacts with the environment it is placed in and obtains in return a scalar reward as shown in Figure 2.1. More precisely, at each time step, the agent gets a representation S_t of the environment and given this representation, an action A_t is selected. The environment then transitions into a new state S_{t+1} and the agent is given a reward R_t as a consequence of its previous action A_t. This reward allows the agent to asses the quality of the action taken. The objective for the RL agent is to learn an optimal action policy, which is a mapping between the states and actions that maximize the expected cumulative reward.

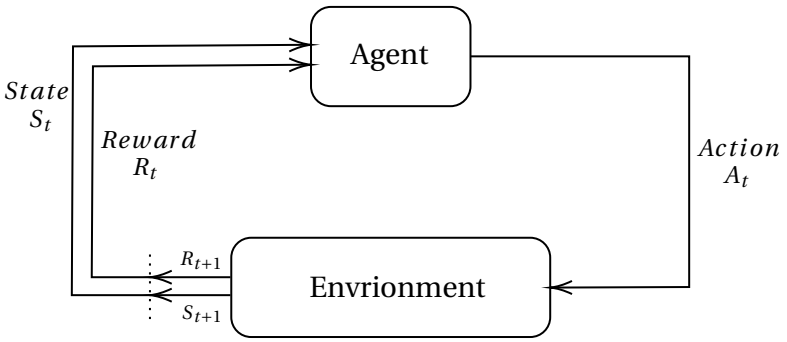


FIGURE 2.1: The agent–environment interaction in a Markov Decision Process.

RL is different from traditional methods of supervised learning since the agent doesn't know which is the best action to take at a given moment and needs instead to determine it based on a trial-and-error approach as explained previously. A Reinforcement Learning problem is usually formulated as an optimal control of a Markov Decision Process (MDP). Other aspects of RL such as the optimal policy, the value function and the Bellman optimality condition are discussed in this chapter.

2.1 Notations

Throughout this document, the set of real numbers is denoted by $\mathbb{R}$. Vectors are denoted by small letters whereas matrices are denoted by capital letters. The transpose of a matrix A is denoted by A^T. For $A \in \mathbb{R}^{n\times n}$, A_{ij} denotes the entry

in row i and column j. The i^{th} component of a vector x is denoted by x_i. $\mathbb{E}\{\cdot\}$ represents the expectation of its argument and the cardinality or number of elements of a finite set $\mathscr{S}$ is denoted $card(\mathscr{S})$.

2.2 Preliminaries

The path planning problem in an unknown environment with transitions between different states can be conveniently captured by a directed graph $\mathscr{G}(\mathscr{N},\mathscr{E})$ of order n ($n \geq 2$), where $\mathscr{N} = \{1,2,\ldots,n\}$ is the set of nodes representing the possible positions the system can be at and $\mathscr{E} \subseteq \mathscr{N} \times \mathscr{N}$ is the set of edges. A directed edge from node i to node j is denoted by $\varepsilon_{ij} = (i,j) \in \mathscr{E}$ and represents a link between neighboring positions which are connected, thus allowing the system to go from position j to position i. A graph is said to be undirected if and only if $\varepsilon_{ji} \in \mathscr{E}$ implies $\varepsilon_{ij} \in \mathscr{E}$. In this work, links are bidirectional given the motion planning problem considered. All positions that are directly connected to node i are said to be neighbors of position i and belong to the set $\mathscr{N}_i = \{j \in \mathscr{N} \mid \varepsilon_{ji} \in \mathscr{E}\}$. Hence, at every given position (x_i, y_i) the agent can observe the states included in the set $\mathscr{N}_i = \{j \in \mathscr{N} \mid \varepsilon_{ji} \in \mathscr{E}\}$. The graph is strongly connected since all accessible states of the workspace are reachable from any initial position, i.e., all states are recurrent. Therefore, the corresponding Markov chain is irreducible. This formulation is particularly suitable for the coverage problem given that the final objective is to visit all the nodes that can be reached. It should be noted that the graph representing the workspace is a priori non stationary as it evolves in time depending on the changes in the environment it is representing. Each node i is characterized by a unique and unchangeable tuple (x_i, y_i) which translates into the geographic position in the Cartesian 2D space. A variable s_i is used to describe at a given time the nature (or state) of node i (visited location or not, obstacle, starting position, etc.).

$$s_i(t) = \begin{cases} 1 & \text{if cell } (x_i, y_i) \text{ is a free cell,} \\ 2 & \text{if cell } (x_i, y_i) \text{ is an obstacle,} \\ 3 & \text{if cell } (x_i, y_i) \text{ is a visited cell,} \\ 0 & \text{if it's unknown.} \end{cases}$$

2.3 The Markov Decision Process

In practice, the RL agent is the emulation or software entity of the physical system and its control actions for decision making. For this reason, we use the terms *agent* and *robot* interchangeably throughout this report. In the RL framework, the agent is the brain responsible for learning and taking decisions, and as exposed earlier, the goal of RL is to control the actions of this agent through a suitable control action policy. The agent continuously interacts with an environment, in particular through the sensors and the actuators. A mobile robot operating in a real environment has to face many uncertainties and the actions it performs do not always result in the expected outcome. Moreover, sensors send back data that is often noisy which further complicates the precise characterization of the relationship between the two and its dynamics. This interaction process between the

agent and the environment and the feedback control loop shown in Figure 2.1 is modeled by a Markov Decision Process. The formalism introduced by Markov's Decision Processes makes it possible to incorporate the uncertainties in the interaction with the environment. MDP have been extensively studied by Bertsekas [18], White [19], Whittle [20] and Puterman [21] and are still today at the heart of RL theory. Given that they take into account the possible the uncertainties associated with real-world scenarios, the control policies obtained are usually significantly more robust than those obtained with deterministic methods.

At each time step, an action is selected which makes the system transition into a new state and resulting in a reward. The happens sequentially and leads to what is known as a *trajectory*, which is the sequence of states (and also actions and rewards).

For the coverage problem, at a given moment, the state of the system is determined by the the situation of the entire space t(covered, non covered areas, position of the robot and obstacles, or unknown areas). However, the current state of the environment is not completely observable, hence an assumption of complete observability of the corresponding MDP is not valid. This assumption is very important and has the advantage of allowing the calculation of trajectories based on the observable subset of the state space, which is by definition smaller than the entire set representing the free space, hence reducing the computational complexity of the online process of algorithm to take an action.

2.4 Theoretical Framework

2.4.1 Markov Process

Definition 2.4.1. A stochastic process describing the state S_t is called *Markov Chain* if and only if the conditional probability of S_{t+1} given $S_t, S_{t-1}, ..., S_1$ depends only on S_t, i.e.,

$$P[S_{t+1}|S_t] = P[S_{t+1}|S_t, ..., S_1]. \tag{2.1}$$

In other words, the future is independent of the past given the present. This means that knowledge of the present is sufficient to predict the future. However, this does not mean that the future is independent of the past.

Definition 2.4.2. A *Markov Decision Process* is a tuple $(\mathscr{S}, \mathscr{A}, \mathscr{P}, \mathscr{R}, \gamma)$

- $\mathscr{S}$ is a finite set of states
- $\mathscr{A}$ is a finite set of actions
- $\mathscr{P}$ is a state transition probability matrix, $p^a_{ss'} = P[S_{t+1} = s'|S_t =, A_t = a]$
- $\mathscr{R}$ is a reward function, $\mathscr{R}^a = \mathbb{E}[R_{t+1}|S_t = s, A_t = a]$
- γ is a discount factor, $\gamma \in [0, 1]$

In general, the transition probabilities from a Markov state s to a successor state s' are given by $p_{s,s'} = P[S_{t+1} = s'|S_t = s]$ and represented in the transition matrix $\mathscr{P}$ which defines transition probabilities from all states s to all successor states s'.

$$\mathscr{P} = \begin{bmatrix} p_{1,1} & \cdots & p_{1,n} \\ \vdots & \ddots & \vdots \\ p_{n,1} & \cdots & p_{n,n} \end{bmatrix}. \tag{2.2}$$

The reward function $\mathscr{R}^a = \mathbb{E}[R_{t+1}|S_t = s, A_t = a]$ informs us on the quality of the decision to take action A_t while being in state S_t. This is very important as it is based on this criteria that actions are selected and the overall action policy is improved. However, we are not only interested in the reward for one single state, but rather the total cumulative reward that obtained from being and thus covering a sequence of states. That is why we define the goal G_t.

Definition 2.4.3. The return G_t is the total discounted reward from time-step t.

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.3}$$

The value of receiving reward R after $k+1$ time-steps is $\gamma^k R$ and with $\gamma \in [0,1[$, this means that immediate rewards are valued more than delayed rewards, which is an important aspect when dealing with Markov Reward Processes in the context of Reinforcement Learning. In practice, this signifies that we try to maximize in priority the present reward since future ones are discounted and thus given less importance. Therefore, $\gamma \to 0$ gives the agent a "short-sighted" evaluation whereas $\gamma \to 1$ makes it possible for the agent to consider rewards in a "far-sighted" fashion.

One of the main reasons why the reward is discounted with time is because of the uncertainties on the future which come from the imperfect knowledge of the model of the environment. In fact, for the considered coverage problem, a model of the dynamics of the environment is not available and the possible changes that may occur in the free space (moving obstacles for instance) remain uncertain.
As we will see later in this work, the discount factor γ is one of the many parameters that have an influence on the convergence speed towards the optimal action policy during training.

This study is based, among other things, on an assumption consisting in considering the Markov process describing the environment as being partially observable. This is due to the fact that this thesis considers CCPP for workspaces that are unknown. We therefore define in the mathematical sense this concept which will be of use later in this report.

Definition 2.4.4. A POMDP is a tuple $(\mathscr{S}, \mathscr{A}, \mathscr{O}, \mathscr{P}, \mathscr{R}, \mathscr{Z}, \gamma)$

- $\mathscr{S}$ is a finite set of states
- $\mathscr{A}$ is a finite set of actions
- $\mathscr{O}$ is a finite set of observations
- $\mathscr{P}$ is a state transition probability matrix,

$$p^a_{ss'} = P[S_{t+1} = s'|S_t =, A_t = a]$$

- $\mathscr{R}$ is a reward function, $\mathscr{R}^a = \mathbb{E}[R_{t+1}|S_t = s, A_t = a]$.
- $\mathscr{Z}$ is an observation function, $\mathscr{Z}^a_{s'o} = P[O_{t+1} = o|S_{t+1} = s', A_t = a]$
- γ is a discount factor, $\gamma \in [0,1[$

2.4.2 Value Function and Bellman Equations

Definition 2.4.5. The *value function* $v(s)$ gives the long-term value of state s

$$v(s) = \mathbb{E}[G_t|S_t = s] \tag{2.4}$$

Simply put, this function represents how good is the state s for an agent to be in.

This value function can be decomposed into two parts *(i)* an immediate reward R_{t+1} and *(ii)* a discounted value of successor state $\gamma v(S_{t+1})$ as follows:

$$v(s) = \mathbb{E}\left[R_{t+1} + \gamma v\left(S_{t+1}\right) \mid S_t = s\right] \tag{2.5}$$

This equation called Bellman equation is crucial and it determines conditions for optimality in the evolution of the states.

Definition 2.4.6. A *policy* π is a distribution over actions given states,

$$\pi(a|s) = P[A_t = a|S_t = s]. \tag{2.6}$$

The policy determines the behavior of the agent.
Note that policies are time-independent.

Definition 2.4.7. The state-value function v_π of an MDP is the expected return starting from state s, and then following policy π

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] \tag{2.7}$$

Definition 2.4.8. The action-value function $q_\pi(s,a)$ is the expected return starting from state s, taking action a, and then following policy π.

$$q_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \tag{2.8}$$

2.5 Model Free Methods

Two classes of learning methods can be distinguished in Reinforcement Learning: model-free and model-based methods. Model-based methods assume a representation of the environment in which the agent evolves is available *i.e.*, the transition function between states and the reward function are known and the agent plans accordingly. In these cases, the environment can be represented by an MDP and when the problem is well defined and its corresponding MDP is completely known, solving it is straightforward and can be done using for example [18] based on the Bellman equations [22] leading to the optimal policy that maximizes the amount of reward the agent can expect to get. However, in most real world applications, the dynamics of the workspace (or the environment) the Reinforcement Learning agent operates in, are usually not fully defined nor characterized. When an agent is introduced in a new environment, how can it figure out - with no information given on the environment - the right thing to do? How can it find out which actions lead to the highest reward when a model of the environment is not known?

This is the goal of model-free methods that learn directly from actual experiences and the interaction of the environment.

Therefore, in this study case, the MDP is not given beforehand and the goal is to solve it. More specifically, the environment is assumed to be partially observable (*i.e.*, a POMDP). As a consequence, the equations governing the way the environment operates (especially when it is dynamic) are not fully known across the whole space.

In this hesis, the algorithm to be used is a model-free one which does not make use of the distribution of probabilities of transitions (Equation 2.2) between states associated with the MDP. The core idea of the algorithm is based on the trial-and-error paradigm.

2.6 Solving the RL problem

There exist several ways of solving the RL problem:

1. Learning the value function by looking at the interaction under some policy and learning from its result. For example, if the agent follows a random walk initially, how well does it do in the coverage and how much reward does it get?

2. Since is not necessary to learn the model of the environment, the agent can instead learn a policy directly using algorithms such as Q-learning or policy gradient as explained in the next part.

These two learning techniques are explained in more details in what follows. In both cases, the learning procedure takes place in two steps and consists in first evaluating the quality of a given policy by trying out things and evaluating how good the outcome is and later optimize to converge towards the best policy.

2.6.1 Policy Search and Value Function Based Methods

There exist multiple methods to solve the RL problem through optimal control of the process. The first one being policy search methods [23], such as REINFORCE algorithm. Policy search methods try to optimize directly the policies which are evaluated by constructing their Q-value functions (Equation 2.4). The algorithm is initialized with an arbitrary policy. Next, the corresponding value function of that policy is computed. Later, based on the previous value function, a new enhanced policy is derived. This procedure is repeated and ensures in this setting that a new policy is enhanced in comparison to the past estimated ones.

By contrast, value function based methods are based on the Q-value function itself (Equation 2.4), *i.e.*, the algorithm is initialized with an arbitrary Q-value function and an improved value is derived. This upgrade is performed with the goal of eventually reaching the optimal value function. This way, the algorithm can be trained off-policy (such as Q-Learning [24] and SARSA [25] for example). The main advantage of this operating mode is that the Q-values are a direct indicator of how good being in a given state is.

In policy search methods, one can easily obtain the associated value function with Bellman equations and for value function based methods, the optimal policy can also immediately be derived from the optimal value function. The two techniques rely on the same theoretical rules.

Besides, research has opened up the field to more sophisticated uses, in particular combining the two above-mentioned methods. In fact, a common procedure for solving RL problems include Actor-Critic Methods [26]. Actor-critic methods are a hybrid type of methods that share the most interesting features of the two previously mentioned methods by exploiting at the same time the value function and the policy function spaces. AC remain to this day the state-of-the-art schemes particularly suited for continuous space problems. They are known to be relatively fast compared to value iteration methods and are usually applied to simpler problems [27]. For example, the recent version of AC proposed by Mnih in [28] and called Asynchronous Advantage Actor Critic (A3C), deploys a distributed system involving multiple agents interacting in parallel with the environment and sharing their knowledge and experience to achieve convergence more rapidly. One of the drawbacks of this method lies in the complexity of its implementation and the difficulty in designing it with the appropriate parameters allowing the expected fast convergence.

In this work, we have finite input (state) and output (action) spaces, because for convenience, a choice was made to address this problem in discrete form, thus reducing the dimensionality and avoiding a possible combinatorial explosion.

2.6.2 Value Function Approximation

Using function approximators for representing and learning value functions in Reinforcement Learning makes it possible to generalize the learned model from seen states to unseen states more easily instead of storing the value function for each state (as it is the case in Q-Learning with a look-up table) [29]. This approach has the advantage of being scalable to practical Reinforcement Learning problems. In fact, for some problems such as the computer game Go, the state space is of size 10^{170}. Therefore, the usual approaches making use of tables storing values in large matrices are no longer practical because they do not scale-up especially in continuous (hence infinite) state spaces.

The goal is to build an estimate of the state-value function $v_\pi(s)$ (Equation 2.4.7), using a parametric function approximator $\hat{v}(s,\mathbf{w})$ where $\mathbf{w}$ is a vector of weights (of a neural network in our case). The objective is to use the function approximator $\hat{v}(s,\mathbf{w})$ that fits $v_\pi(s)$ with a limited number of parameters. The same reasoning applies to the action-value function $q(s,a)$ (Equation 2.4.8) approximated by $\hat{q}(s,a,\mathbf{w})$ across the whole state and action spaces. This method is very convenient since the approximation of the function can be used for states that have never been seen before.

There exist several types of value function approximators: linear combinations of features, decision trees, nearest neighbors, etc. Depending on the context, each one of them leads to different results and efficiency. In what follows, neural networks are used especially because they have the crucial property of being differentiable, which will be useful for computing the gradient and updating the network.

This doesn't change anything to the core concept of Reinforcement Learning exposed in the previous paragraphs, with the exception that the policy is now generated by a neural network as it is shown in Figure 2.2.

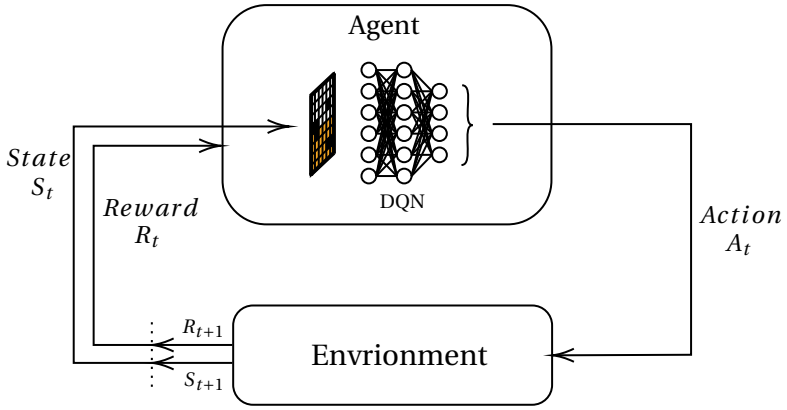


FIGURE 2.2: Reinforcement Learning with policy learnt via DNN.

2.7 Deep Neural Networks

The Neural Networks that are used in this thesis have witnessed considerable growth in the past decade. They are increasingly used in many fields and are at the origin of the greatest technological breakthroughs, such as the autonomous car, image recognition in healthcare or AI-based trading in finance. This section presents the core concept of Neural Networks (NN) in general as well as the particular case of Deep Neural Network (DNN). The basics of Convolutional Neural Networks (CNN) that are used in the final implementation are also exposed.

2.7.1 Artificial Neural Networks

Artificial Neural Networks (ANNs) are biologically inspired computer programs composed of a collection of virtual neurons. As biological systems, the neurons are connected together to form a network of nodes. In biological networks, synapses play a key role in the guiding process of electrical impulses through the network. The type of information circulating in neural nets is of scalar type, analogously to electrical signals emitted in a brain of a living being. In Artificial Neural Networks, neurons are the processing units. In fact, signals propagate and are transmitted from one neuron to the next. Each input connection is associated with a parameter (also called *weight*) and the overall network behavior is determined by these connections between the different neurons. The weights determine the strength of the signal on a link between two neighboring neurons.

Feed Forward Networks

Feed Forward Networks (FFNs) are a special type of neural networks in which connections do not form a cycle. Unlike Recurrent Neural Networks (RNNs) which are used for different types of sequential tasks and in which signals can propagate in both directions thanks to loops introduced in the network allowing them to have a

temporary and short term "memory", FFNs tend to associate or map inputs with outputs and are dynamic in the sense that their parameters (or weights) change during the training phase until an equilibrium point is reached which corresponds to a fitted or trained model.

In this type of networks, neurons are mathematical functions whose input is the weighted sum of signals emanating from all preceding neurons (see Figure 2.3). The function applied to this input when the information traverses the node is known as the neuron's *activation function*. This process propagates signals from the input neurons, to the output layer.

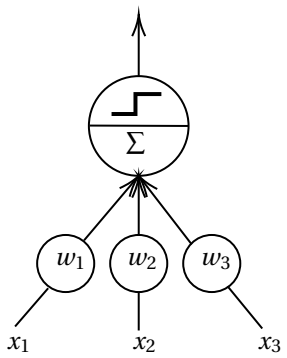


FIGURE 2.3: Operations at one neuron within a network.

Activation Functions

Activation functions are essential bricks in neural networks architectures since they have a direct impact on the transmission (and non transmission) of signals to the next neurons. Moreover, activation functions directly impact the computational efficiency of the network as well as the quality of the output *i.e.*, the accuracy of the predictions. Besides, activation functions also influence the convergence speed and can sometimes even prevent neural networks from converging. The most common activation functions include:

1. Logistic function (sigmoid): $\sigma(z) = \frac{1}{1+e^{-z}}$
2. Hyperbolic tangent function: $tanh(z) = 2\sigma(2z) - 1$
3. Rectified linear units (ReLUs): $ReLU(z) = max(0, z)$

ReLu functions are with no doubt the most widely used [30]. These functions operate exclusively in the hidden layers because they are differentiable and this guarantees that the back-propagation algorithm explained in what follows works properly. As for the output layer, several activation functions can be used depending on the type of Machine Learning problem: Sigmoid units (also called binary step functions) are usually used for binomial classification and Softmax activation functions which are used in the final architecture, are more suited to multinomial classification.

Efforts have been made in the design of the architecture to minimize the impact of the vanishing gradient. As such, the final network is not very deep. Nevertheless, ReLU functions were used preventively instead of sigmoids to avoid the vanishing gradient problem (see section 2.7.3).

The Over-fitting Problem
Over-fitting occurs when the model is getting too complex during the training phase. This translates into the weights becoming very large. This issue can be dealt with in several ways. One of these techniques used in training is the dropout [31]. It consists in randomly dropping units (neurons and their links) as shown in Figure 2.4. By doing so, the overall structure becomes less complex.

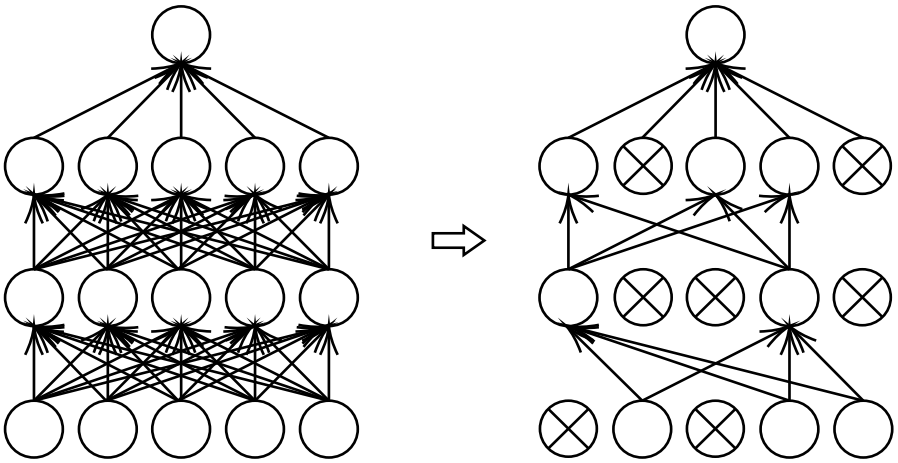


FIGURE 2.4: Example of a simpler network created by dropping out neurons in the network on the left.

Back-propagation & Network Weights Updates
In order to improve the accuracy of the network in predicting the outputs, the weights are updated after each iteration and based on the loss function. For each training instance, the training is performed in the following steps:

1. Forward pass through the network to make a prediction
2. Measure the error by computing the deviation from the real value
3. Backward pass going through each layer in reverse to measure the error contribution from each connection
4. Modify the connection weights to reduce the error

Cost Functions & Optimization
Network updates are performed with respect to a loss or cost function (also called objective function). Loss functions are another crucial component in neural networks. In fact, they define how a neural network learns. Training a NN consists in minimizing the error on a training set. This is carried out using the Gradient Descent method, and requires the use of a loss function that needs to be optimized.

There exist several functions that can be used to estimate the error generated by the set of weights in a NN. In the context of regression models, the RMSE is a very popular one. However, the RMSE is very sensitive to noise as it merely sums the square of the observed errors. That is why in this project, we opt for a slightly different loss, called *Huber* loss that is more robust and less sensitive to outliers in data than the squared error loss. In fact, the *Huber* loss combines the best properties of L_2 squared loss and L_1 absolute loss in the sense that it makes the function "more convex".

A popular stochastic optimization method for solving the error minimization problem is called *Adam* [32]. This stochastic optimizer was specifically developed for training deep neural networks. When it was presented in 2015, it introduced many advancements especially in training speed. Adam is one of the few optimizers that was designed to work well and be efficient across a wide range of problems involving neural networks. The underlying principle of Adam optimization is still the gradient descent method but its novelty lies in the fact that it also implements adaptive estimation of first-order and second-order moments. This means that for each parameter of the network, adaptive learning rates are computed to update the weights. One advantage of this mechanism is that it slightly slows down the learning which enables the network to better learn, thus improving the learning quality. This helps the algorithm converge rapidly, consequently increasing the learning speed.

2.7.2 Convolutional Neural Networks

In the proposed algorithm, a Convolutional Neural Network is used to analyze the environment. In fact, the space is entirely described by a dynamic map represented by image frames. CNNs are a special type of Artificial Neural Networks but they function in a similar way as other types of Neural Networks. Although they can be used for various types of problems, including some involving classification, Convolutional Neural Networks (CNNs) are widely used to analyze images and are known for their remarkable ability to detect, extract features and patterns efficiently in images. They are composed of convolutional layers which apply convolution operations on the input images using filters which are matrices that convolve across the image. Using the convolutional layers allows the agent to learn spatially invariant features across the map as the same filter is applied to different sections of the map. This is particularly useful to identify the visited areas and the areas occupied by obstacles. In this type of network, the deeper the network, the more it will be able to detect sophisticated forms. More specifically, the layers close the the input learn simple shapes whereas the following layers learn increasingly complex features [33]. However, in our case there are no complex shapes to learn but simple geometric forms identified by different colors, representing the state of the considered areas. That is why, in the proposed model, only two convolutional layers are deployed. Given the obtained results and the simplicity of the frames, two seem to be enough. Experimentations showed that more layers did not lead to better results.

2.7.3 The Vanishing Gradient Problem

The vanishing gradient problem [34], [35] is a phenomenon that can be observed during the training phase of a neural network using gradient-based optimization techniques. For a deep neural network, *i.e.*, a network composed of multiple hidden layers, the learning is such that the error is minimized and the weights updated. In this process involving back-propagation, the value of the gradients generally decreases as the values propagate backwards in the network. As a consequence, this creates a disparity in terms of the speed of training because neurons

in close to the input layer learn at a slower rate compared to the deeper layers. In other words, the further one goes through the network, the lower the gradient becomes and the more difficult the updating the weights is.

2.8 Related Work

The advances of techniques in the field of Artificial Intelligence shed lights on novel strategies for learning directly from the raw sensors inputs with different frameworks involving for instance Neural Networks or Reinforcement Learning. In recent years, several methods have been proposed for tackling autonomous navigation tasks with Deep Reinforcement Learning algorithms. These methods usually model the navigation or motion planning problem as a Markov Decision Process, and consider observations obtained from sensors readings as states. The aim is to come up with with an action optimal policy capable of guiding the robot to a target position, sometimes under constraints.

When it comes to complete coverage which is a particular case of motion planning, significant efforts have been made in completely known and stationary environments. The most common approaches are based on cellular decomposition. The core idea consists in decomposing the space to cover into a set of cells that do not overlap. Since this decomposition is typically done offline, it requires knowledge about the environment such as the position of the obstacles and the critical points. The robot can then look for a the sequence of actions which covers all the cells that are adjacent. Full coverage is eventually carried out by back and forth movements of the robot. This approach, although simple in practice, is not always applicable because an exact cellular decomposition does not systematically exist for a given environment. In addition, it requires an *a priori* knowledge about the workspace such as the position of the obstacles. In some cases, instead of dividing the space into cells that cover exactly all the reachable areas, it is more convenient to use an approximate cellular decomposition such as in [36]. A graph coverage is performed by subdividing the workspace into discrete cells and following a path in the graph formed from the cells, the robot is able to cover each point only once. Choset [37] proposed a new solution to the cellular decomposition which combines the advantages of cell breakdown with model-based approaches and minimizes the number of cells used to subdivide the free space. However, this approach also requires prior knowledge of obstacles locations and critical points. Acar and Choset [38] then improved the cellular breakdown algorithm for boustrophedon to ensure that the recovery task does not require any prior knowledge. This study tries to identify all the critical points of a morse function used to characterize the space with means of sensors.

One drawback of the cellular decomposition is due to the fact that when the environment changes, the workspace must be divided again. Additionally, an approximate cellular decomposition is by nature an approximation of the free space and as a consequence some areas are not even considered as being part of the environment. They are therefore not covered which intrinsically decreases the performance of the coverage.

Cao et al. proposed in [39] a method dedicated to a lawn mover to cover an area while avoiding obstacles by combining laser scanning and a boustrophedon

cellular decomposition. However, as explained earlier, information is necessary for this algorithm to work properly such as the garden boundaries and the position of obstacles.

Butler et al. proposed in [40] an algorithm called distributed coverage of rectilinear environments, in which the robot can find the location of obstacles by contact with the sensors.

[41] offers a localization strategy based on a particle filter (PFL) and an update of the environment map using reinforcement learning. The localization strategy of the the robot keeps in memory a map which is updated when there are important changes in the environment. Correct updating is carried out via RL and this method teaches the agent how to decide by allocating rewards that are proportional to the difference between the map and the LIDAR scans. The proposed decision framework for updating maps leads to greater accuracy and offers localization improvements compared to known approaches (e.g., SLAM).

In the literature, several strategies for complete coverage missions have been considered, where the environment is unknown such as in [42]. In this sense, methods proceed without a map which is the case in [43] for example, which presents a learning-based mapless motion planner to reach multiple targets in unknown environments. In general, for mobile robots, motion planners rely on precise laser range sensing for creating and maintaining an obstacle map of the navigation environment.

In general, studies focusing on non stationary environments with dynamic obstacles are limited. Unlike complete coverage in stationary environments, in dynamic environments, the calculated path can become invalid at any time due to changes in the workspace structure. Therefore, the algorithm must be able to quickly re-plan and adjust its path on the fly in real-time. For this purpose, Yasutomi et al. [6] presented a coverage approach where a cleaning robot is taught to clean room, while successfully avoiding obstacles and walls in an unseen environment. However, the complexity of the calculations involved in the learning process, make it hard to handle complex and unstructured environments, which is usually the case in outdoor applications.

One of the most efficient methods currently in use in the field is the bio-inspired algorithm initially proposed by Luo and Yang in [44] where the dynamics of each position on the map are topologically organized in a network. However, for this algorithm, the agent can end up in some situations called deadlocks where it finds itself surrounded by either obstacles, or locations already visited. The graph-like representation of the workspace in this method inspired by a biological model doesn't provide the robot with global information to efficiently avoid these deadlock cases. To solve this problem and deal with situations where the robot finds itself in dead ends, a model setting up priorities and a new global mechanism for backtracking are proposed in [45] to accomplish the coverage task in a dynamic environment. It determines the best point of turning back using a greedy optimization criterion and plans an optimal path. However, this method presupposes a perfect local knowledge of the space (obtained thanks to the sensors) for decision-making. Therefore, even if it leads to good results, this strategy is not sufficient in our case since we do not have this knowledge directly and it is necessary to generate it from data collected on-board sensors. In addition, studies

carried out on environments that are both dynamic and not completely known in advance are still rare and suitable mature methods on this subject are rather scarce in the literature. Hert et al. [46] also presented a real-time CCPP algorithm for a ground robot evolving in an unknown environment, allowing the robot to plan a path covering the workspace and minimizing visiting the same areas several times. In this method, the environment is discretized and divided into cells of fixed size. The robot makes use of a memory combined with environmental information to make a decision. In this publication, the work was done in a 3D environment for an underwater robot with limited visibility. Even though this article is relatively old (article published in 1996) and most of the major advances in robotics and autonomous systems have taken place mainly in the last ten years, this approach is interesting in terms of the formulation of the problem as well as the simplifications made which could serve this study in a 2D space for possible future improvements.

Tse et al. [47] made use of neural networks and presented a model based on back-propagating. During the coverage task, the robot keeps in memory the path learnt and generated previously, as well as other parameters such as path length or the the number of turns. This makes it possible for the robot to to update its memory whenever a new environment is encountered. Thanks to the learning capabilities provided by the neural network, the robot is able to explore an unknown environment and find optimal paths with the least possible overlap and a minimum number of movements for energy conservation. This algorithm is based on a principle very similar to that of Luo and Yang [44].

Finally, other simple and more original approaches have been explored with less conclusive results. Among them, one can cite the work of Russell in [48] where the path of the robot is marked by a trail of digital pheromone so as to form a “heatmap”; then, based on this kmowledge, the robot is more likely to be “attracted” to regions that have not been previously covered. The study in [49] presents a CCPP model that consists of three components: a scanning algorithm, a point-to-point movement algorithm, and a component to deal with areas and corners which are particularly difficult to reach. This enables to considered cleaning robot to avoid missing certain places in the workspace. The work presented in this publication differs from all previously mentioned literature since the covering task is approached in a cooperative manner given that the authors did not use a single robot but several (2 and 5 for simulations).

2.9 Summary

The nature of the coverage problem involves a very large and potentially infinite state space, as, it is usually the case when dealing with practical robotic applications. In this study, the problem is reduced to solving a discrete Markov Process using Reinforcement Learning. More specifically, neural networks are used to learn a policy that allows the robot to perform complete coverage. The representation of the workspace in the form of a dynamic map serves as an input for the model. From a coverage point of view, this map shows the areas and their status (covered, non covered, obstacle). The next section explain in more details the architecture used and in particular, how the algorithm is trained. The RL formulation

makes it possible to use a Neural Network to learn a policy function that enables the agent to learn through experimenting the best behavior possible. However there are three main challenges:

- How to obtain the information to create the map?
- How to formulate the coverage as a game?
- Another difficulty lies in the fact that for a single coverage task, with sufficiently long term horizon, it is difficult to identify which actions lead to good or bad results.

The next chapter focuses on answering these questions and deals with the mentioned difficulties, notably by:

1. Designing a computer vision algorithm for scanning the space, which is integrated to the RL path planner
2. Shaping the reward function
3. Introducing a movement budget to limit the operating time of the agent.

Chapter 3

Methodology

3.1 Deep Reinforcement Learning Model

Figure 3.1 summarizes the architecture used. The DQN is composed of convolutional and fully-connected layers. The model takes as an input the current map containing the observed part of the environment, the coverage information, as well as the position of the robot. The convolutional layers are all padded to ensure that the output has the same shape as the input. Before feeding it into the fully-connected layers, the output of the convolution layers is flattened. The last fully-connected layer is of size $card(\mathcal{A})$ and represents the Q-value for each action.

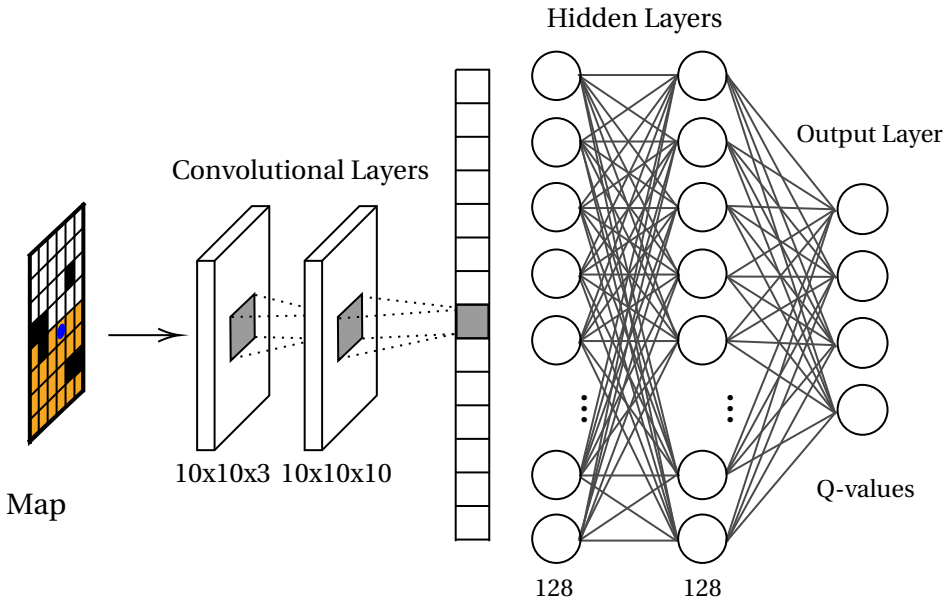


FIGURE 3.1: Neural network structure for the reinforcement learning agent.

The architecture shown in Figure 3.1 has been determined after a number of experiments to find adequate parameters in terms of the number of layers and nodes per hidden layer. Traditionally, in image processing, one hidden layer is considered to not be sufficient, as the more convolutional layers there are, the more details can be learned from the images. A drawback of this high precision, is the increase in complexity (proportional to the number of operations required to compute an output), hence the training time. Therefore it is not possible to use

an arbitrarily large number of layers. For most image processing tasks in Machine Learning applications, two or three layers are usually enough. In the modeled environment, there are no complex patterns or shapes, that is why, two layers or three layers should be sufficient. Additionally, training sessions showed no significant improvements in the performance of the algorithm with three convolutional layers compared to an architecture with only two. When it comes to the hidden neural layers, the design is more delicate, one hidden layer usually makes it possible for the network to approximate any complex function. And two hidden layers offer more flexibility compared to a single layer. And an additional layer adds another level of degrees of freedom. Although the risk of overfitting that comes with using three hidden layers is higher, in practice, it is beneficial for a task which is as complex as the policy learning one, and given the large amount of available training data (see next paragraph). Regarding the neurons in each layer, one rule-of-thumb stated by J. Heaton in [50] is that "the optimal size of the hidden layer is usually between the size of the input and size of the output layers". This helps decrease the number of degrees of freedom, thus making the model simple. This is crucial because the simpler the model, the faster and more robust it is, and the less likely it is to accumulate errors [51].

3.2 Coverage Game & Simulation Environment

The coverage problem is formulated as a dynamic game. More specifically, the library *PyGame* was used to create the simulation environment used for training and testing, while the python library *OpenAI Gym* [52] specifically made for RL applications was used to interact with the environment, start game episodes, observe states, collect rewards and perform actions.

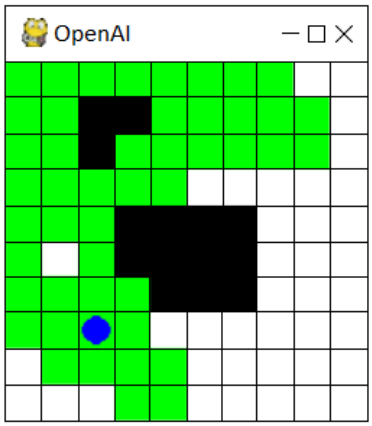


FIGURE 3.2: A 10x10 grid map showing an agent (in blue), two obstacles (in black) and the covered cells (in green).

This representation of the environment does not reveal the trajectory of the robot. We try to show hereafter, the route followed by the agent whenever possible in order to better understand its behavior.

3.3 Training the Network

When training such a network, the questions of where to find an appropriate dataset arises naturally. The lack of a suitable dataset is a real obstacle in all Machine Learning applications. One of the innovations that made this work possible was the introduction of an artificial dataset generated according to a defined distribution and used for training and testing. More particularly, the environment in which the agent evolves is created using a specialized python library.

During the training phase, the agent performs both exploration and exploitation thanks to an ϵ-greedy policy (a $softmax$ classifier is applied to the output layer of the model in Figure 3.1). As for testing, the robot directly exploits the learned knowledge ($argmax$ classifier) when operating in a new environment.

The Algorithm below provides more details on the training procedure for the double deep Q-network. First, the memory of experiences and other network parameters are initialized. Then, a random valid starting position is chosen as well as a random movement budget within a pre-defined range (see Table in Figure 4.1 for the numerical values of the parameters used for training). The episode continues as long as the movement budget has not been consumed and that the coverage is not complete.

To train the model, the input is not taken directly from the states of the observed environment. Instead, tuples (state, reward, action, next state) are collected after each interaction with the environment and stored in a replay memory which is sampled. This approach helps increase the performance of the learning by reducing the correlation between individual samples in a training batch.

This training method is very standard in the literature for this type of architecture; see for example [53]. Further aspects of the training are discussed in the next chapter with the specificities of the coverage problem.

Algorithm 1: Deep Q-Network training

```
Initialize replay memory 𝒟
Initialize movement budget ℬ
Initialize main network parameters θ with random weights ;
Initialize the target network parameters θ̄ ← θ
for episode = 1 to M do
    Initialize agent state with random position;
    Initialize obstacles;
    Sample movement budget b uniformly from ℬ;
    while b > 0 do
        Choose action with probability ϵ;
        Otherwise, Sample a according to a = max(q_π(s));
        Execute action a and observe r, s′;
        Store (s, a, r, s′) in 𝒟;
        for i = 1 to m do
            Sample (s_i, a_i, r_i, s′_i) uniformly from 𝒟;
            Compute loss L_i(θ);
        end
        Update main network parameters θ with gradient loss;
        Update the second network θ̄;
        b = b − 1
    end
end
```

3.4 Training Data

As is the case in all machine learning applications, the question of where to find data to train the proposed model arises. In this context, a significant amount of data is needed to train the model in Figure 3.1. The artificial data in the simulation environment needs to fulfill two conditions : *(i)* it should be abstract and simple enough for it to be exploitable by the Deep Reinforcement Learning framework (*i.e.,* in the form of discrete maps), and *(ii)* it should correspond at the same time to a certain reality. This was accomplished thanks to the map illustration shown in Figure 3.3. Several types of maps were generated according to a pre-defined distribution such that 2 or 3 obstacles of variable size and position are introduced in the space, along with a random starting position. As explained earlier, the major benefit of such a representation is that it highly reduces the dimensionality of the problem. This allows to conduct a study focusing on the improvement of the motion planning algorithm by simplifying as much as possible the other complex aspects of the problem, namely the environment.

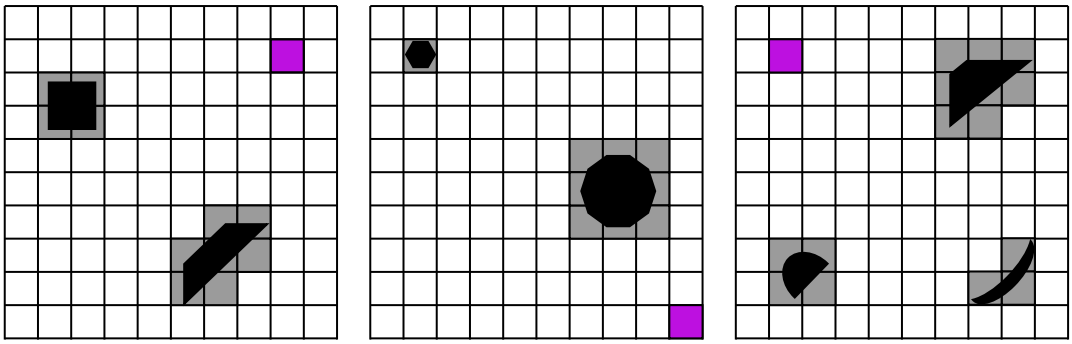


FIGURE 3.3: Typical examples of 2 dimensional maps used as an input for the model.

Generating data and the corresponding environment in which the agent can operate was made possible thanks to Python libraries particularly suited for this type of tasks: *PyGame*[1] and *Gym*[2].

3.5 Algorithm Complexity

One of the advantages of using a neural network or AI-based techniques in general for this type of path planning problem is the fact that the algorithmic complexity is lower compared to other traditional approaches involving exhaustive searches over the entire spaces of solutions [3], [54]. The question of algorithmic complexity is very interesting when it comes to neural networks. This information is very important because it is closely linked with the CPU power and memory capacity needed that should be provided by the embedded hardware. And this has a direct impact on the autonomy or battery life of the device.

There isn't a formal definition for such a concept but one can intuitively guess that it makes more sense to define a complexity only for a trained model, *i.e*, how many operations are necessary for a trained model to generate an output on a testing set. Then, this complexity is directly linked to the number of layers the network is composed of as well as neurons density or how many neurons each of those layers contain. Since we are looking at the inference (or prediction) part, we only have forward propagation. In the model we proposed, one can distinguish two parts: the Convolutional Neural Network and the Deep Neural Network. Each one of them comes with a different complexity. The total computational complexity of a 2D convolutional layer is $\mathcal{O}(n^2 \times k \times d^2)$ [55] where the filter is of size k and the depth dimension is d and n is the size of the input images. We see that this complexity is quadratic in terms of the size of the input n, which is why we strive in the algorithm on the one hand to preprocess the input images of the model by first re-scaling them to their resolution so as to obtain very small images ($n = 10$) and second converting them to gray scale (d = 1). By doing so, we end up with $\mathcal{O}(n^2 \times k)$.

[1] https://www.pygame.org
[2] https://gym.openai.com/

Regarding the dense (or fully connected) layers, the input and output layers are of size 100 (n^2) and 4 ($card(\mathscr{A})$) respectively, and each one of the hidden layers is composed of 128 neurons. Each neuron processes the inputs coming from all neurons of the previous layer as explained earlier. Therefore, For one layer, the number of operations is equal to 100 multiplications and 99 additions, *i.e.*, a complexity of $\mathscr{O}(n^2)$. We conclude that the overall estimated complexity of the system is quadratic in n, the size of the workspace.

It should be noted that this is an approximation of the number of operations in function of n that model requires to produce an output, *i.e.*, take an action. This does not take into account for example the auxiliary operations associated with the testing, nor does it include the computational cost of the training procedure that is carried out offline.

3.6 Obstacle Detection and Local Mapping

One particularity on this work in the the grid mapping chosen to represent the space, and the assumption on the limited visibility of the robot. A challenging aspect that comes with this implementation technique is obstacle detection which is one of the most crucial parts of autonomous navigation. This detection is traditionally carried out by means of sensors measuring distances such as the laser. Nowadays, the computing capacities of hardware having very strongly increased, much more efficient approaches have emerged. Deep Learning for Computer Vision permits to recognize particular shapes in an image and associate them with an object for example. In the case of a lawnmower, the application of deep learning is particularly interesting, because it is a way for the robotic system to distinguish with relatively high accuracy an obstacle as well as areas that have been cut and others that haven't been yet. In fact, unlike a LiDAR, a camera gives the system the ability to discriminate between different scenes and extract information to update the dynamic map. This shows the added value of AI-based techniques in general and deep learning for computer vision and image classification in particular.

A program was created to collect data from the on-board stereo camera, and transfer it to the deep learning algorithm with the goal of eventually extracting information on the shape, size and location of obstacles to update the map. This part was developed with the help of the embedded 3D ZED camera (see Figure A.2) and the Jetson TX2 Nvidia graphics controller for the computations. The advantage of using such a setup is that unlike other sensors, a stereo camera makes it possible to both identify the elements in the space and the distance they are located at with respect the framework of the robot.

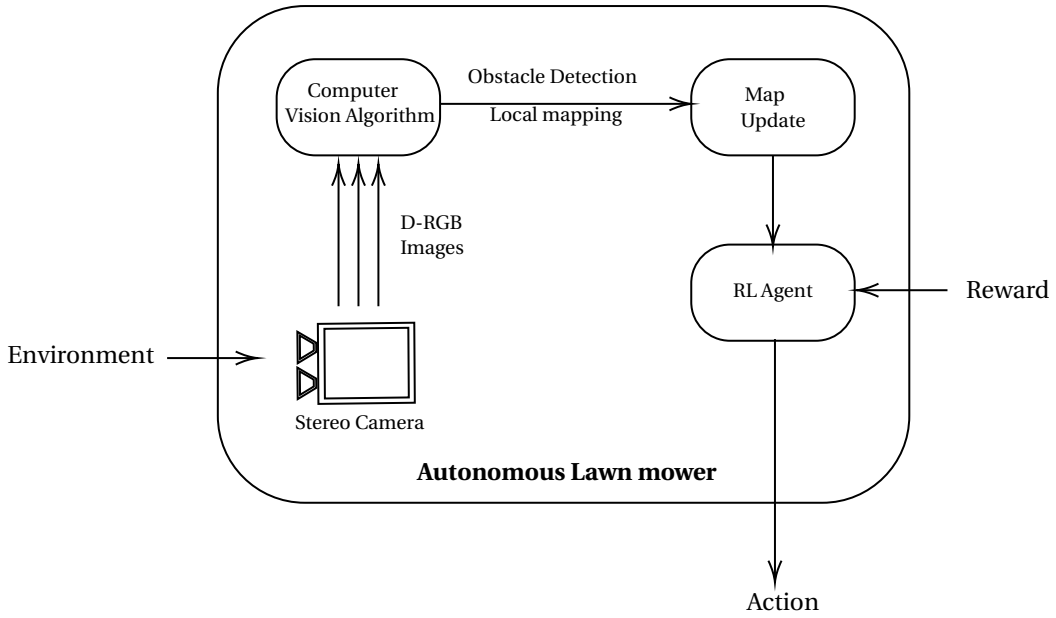


FIGURE 3.4: Block Diagram with sub-systems of the robot.

The way the obstacle detection program operates in coordination with the RL motion planner is shown schematically in Figure 3.4. Although other operations are performed in the program, steps 1 to 5 in Figure 3.5 represent the key-points of the computer vision algorithm.

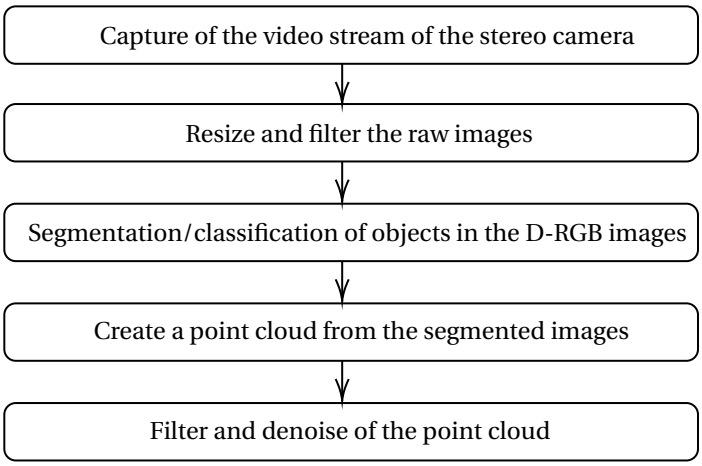


FIGURE 3.5: Principles of the Computer Vision algorithm.

The program developed in C ++, uses the OpenCV and Point Cloud (PCL) libraries. The functions allowing the operation of the ZED camera are provided by the manufacturer's SDK (Software Development Kit) of the ZED camera, available on the Stereolabs website[3]. Finally, in order to work with ROS, the program architecture follows standard convention of any ROS node. Results are shown in the next chapter.

[3]https://www.stereolabs.com/developers/

3.7 Evaluation Metrics

The most important performance indicator in CCPP navigation tasks is the traveled distance or the total length of the path of the robot, which is equivalent to the number of moves in this study. The second criteria we look at to assess the performance of the proposed algorithm, is the *coverage rate* and the *revisited cells rate*, i.e., the percentage of the area that has been covered at least once by the robot, within a given movement budget. We also pay attention to the variation of the values of coverage rates observed for multiple environments, and given by the standard variation. The standard deviation is important as it helps better understand how robust and consistent the DQN model is. This is a very important factor that informs us on how effective the coverage is in terms of the use of available resources. In fact, while a 100% coverage rate is the target goal to reach for the algorithm, it should reach it by avoiding as much as possible to cove cells multiple times, that is to say, with a minimum *revisited cells rate*. All these elements are crucial for evaluating the quality of the results.

A more complete comparative study of the path planner is exposed in what follows, and various situations are considered to study the differences in behavior with other state-of-the-art algorithms.

3.8 Integrating Learning & Path Planning

One of the main hypothesis consisted in supposing that the environment is not known in advance. The space is only assumed to be partially observable. In order to learn the environment, a 2D discrete map is used to represent the workspace. This map gathers all the information the agent collects from the sensors and keeps track of the areas occupied by obstacles as well as the already covered areas. This learned dynamic map is used as input for the model and to generate the next action based on the state of the environment represented by the map. This means that the next move of the robot is determined online on the fly, and no computations are performed offline beforehand. The robot local vision allows it to detect the obstacles, avoid them and update the map accordingly.

3.9 Model Training

To ensure an effective learning that can be generalized to several types of environment which have never seen before, the agent is trained in sufficiently diverse spaces to cover a wide range of situations. For this purpose, the workspaces used for training are of a fixed rectangular shape and contain obstacles of different sizes, that are generated according to a pre-defined distribution.

Instead of feeding only one map of the environment to the input layer of the DQN, four frames are used, mainly because it is difficult to recognize what's happening with only one "shot" and given that we are very much interested in the trajectory, it is wiser to use multiple which show the path followed by the agent. Therefore, a sequence of four game frames are stacked together and by doing this,

the action the agent choice depends not only the current state but is also influenced by the previous game moves through the four previous sequence of game frames.

Relying solely on one network that sets its own target and tries to reach it can possibly make the network unstable and diverge [56]. For this reason, two Q networks are used in the training process: an online DQN which is the main network involved in the learning of the agent at each iteration, and a second network called *target* DQN which is the one used to compute the target Q-values used to train the online DQN. With a pre-defined frequency, the target network is slowly updated with the values of the online network. This technique is used to diminish the risk of the network being destabilized.

3.10 Alternative Coverage Algorithms

As mentioned in the related work section earlier, there have been studies on complete coverage techniques and numerous methods can be found in the literature. However, among them, AI-based approaches are relatively young and not mature enough. Besides the studies focusing on offering a guarantee on the optimality of the solution such as in setups involving multiple robotic systems as in [57] for instance, more recently, [15] introduced a new deep reinforcement learning approach for the coverage path planning problem, which considers limited available power constraints for the robotic system. This method was applied to UAV to patrolling and yields encouraging results. However, this algorithm makes use of the spacial partial local information put on a map that is fed into a Q-network to learn a control policy that generalizes over varying starting positions and varying power constraints. The proposed method in this thesis draws inspiration from the work in [15] bringing other features such as the vision component and others from more general RL frameworks such as the one introduced in [53] and [58]. The results section will examine some of these algorithms.

3.11 Exploration vs Exploitation

The idea of exploration has two meanings: from one hand, exploration is related to the fact that we want to explore the environment as much as possible in order to ensure that all reachable states are reached under certain conditions. However, in the context of learning, it is about exploring the state space but this time by exploring the possible commands or actions in order to improve the best known strategy. These two meanings are very closely related. In order to attain more information about the environment, it is crucial to explore from time to time the space by not only performing actions that we suspect that they could lead to an improved strategy, but also sometimes try out other possible actions that deviate from the current known sub-optimal strategy which can help further optimize the actions taken. As a consequence, exploration in the action space will result in a better exploration in the state space, thus making sure all reachable areas are covered in the long run.

Throughout the learning process, this approach will progressively lead to an increasingly improved overall action policy and eventually make the algorithm converge towards the best action policy by maximizing the total sum of rewards.

The exploration is guaranteed with an ϵ-greedy policy. At each iteration, the agent picks a random action with probability ϵ, and with probability $1-\epsilon$ the action given by the network is chosen. This procedure helps the network explore different actions, thus allowing it to learn new things.

In the context of RL, the agent, when first introduced in the environment has no knowledge or experience it can rely on to make decisions. That is why it has to try randomly (*i.e.,* explore) and learn from the feedback it gets from the random actions taken.

Chapter 4

Results

4.1 Experimental Setup

4.1.1 On The Robot Localization

As explained in a previous section, the problem of the localization of the robot in the space is beyond the scope of this thesis. The issue of determining precisely the position of the agent is a challenge that has been considered in other papers. For example, an estimation engine is developed in [59] and [60] using relative localization to predict the actual position of a mobile robot in a given space thanks to odometry information. Depending on the type of environment, other methods are available to directly obtain the absolute position in the space. Therefore, we assume the position of the agent is known at all times.

4.1.2 The Markov Decision Process

As explained in previous sections, the coverage path planning problem is modeled as a MDP and solved using RL. The MDP is defined by the tuple $(\mathscr{S}, \mathscr{A}, \mathscr{P}, \mathscr{R})$.
The action space $\mathscr{A}$ contains the four actions: $\mathscr{A}$ = {Up, Down, Right, Left} and in a $n \times n$ grid, the dynamic map can be seen as the superposition of 3 independent maps, namely an obstacle map displaying the positions of the obstacles in the workspace, a second map for identifying the positions of the covered and non-covered cells and a map showing the location or the cell occupied by the robot. Additionally, we define the reward function $\mathscr{R} : \mathscr{S} \times \mathscr{R} \longrightarrow \mathbb{R}$ used to help the agent learn the best policy with four different components:

- $r_{coverage}$: positive reward for every new cell target cell that is visited by the robot.
- $r_{obstacle}$: negative reward in case the agent hits an obstacle.
- $r_{movement}$: negative reward to penalize the agent for every unit of movement consumed from the movement budget, i.e, at each step of the game.
- r_{end} : penalty for when the robot depleted all available movement budget, without completing the coverage.

Negative rewards encourage the robot to finish the task as quickly as possible because it makes the agent constantly lose points when the game is still being played. On the other hand, positive rewards incentivize the agent to keep going to accumulate additional rewards.

In this scenario, a game episode is terminated only when the movement budget is entirely consumed or when the agent has successfully performed complete coverage of the whole space.

4.1.3 Training

For the training, the hyper-parameters used are listed in Table 4.1. They were obtained after a series of experimentations with the goal of determining the best combination of parameters. A grid-based search was performed for the couples $(\epsilon, \mathscr{B})$ with a systematic search across intervals. Other hyper-parameters were left constant.

Parameter	Value	Signification
M	5000000	maximum number of training episodes
n	10	width and height of the workspace
$\mathscr{D}$	50000	replay memory size
m	128	minibatch size
$\mathscr{B}$	[80, 150]	movement budget
ϵ	0.1	exploration rate
γ	0.95	discount factor
τ	0.003	target network update factor

TABLE 4.1: Hyper-parameters for the DQN training.

Figure 4.1 describes the evolution of the total reward per episode (or coverage game play) in function of the number of training episodes. The figure shows that throughout the training phase, the agent manages to collect on average an increasingly important total reward which is directly associated directly with the quality of coverage (since the the system is designed to reward the agent based on the exploration of the free space). This shows that the learning improves and the agent gets better at covering the workspace.

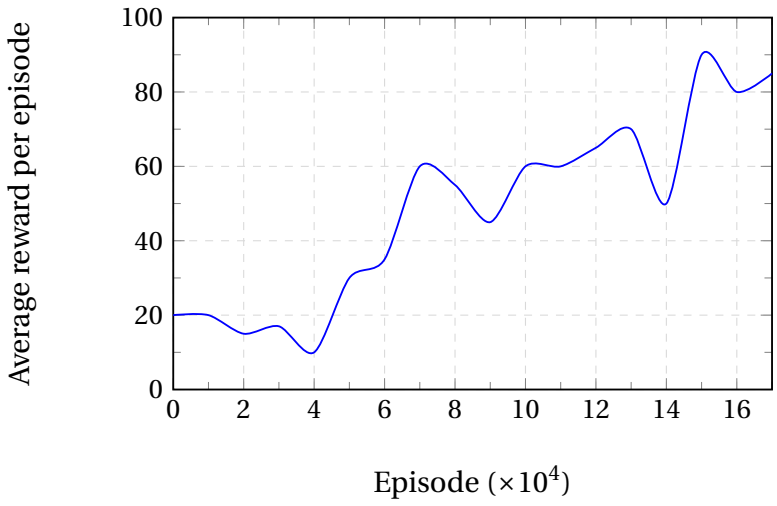


FIGURE 4.1: Average reward per episode during training. The statistics were computed by running an ϵ-greedy policy.

During training, the exploration factor is varying, starting from 1 at the beginning and decreasing throughout the learning process until it reaches 0.1. This is motivated by the fact that the agent initially has not acquired any knowledge, which forces the exploration of the free space in priority. With ongoing training, the agent will gradually exploit and rely on the accumulated knowledge and less on the exploration.

4.2 Simulations and Discussion

In order to have a reference point, we first look at the results of a random approach, then, a naive boustrophedon and spiral trajectory before comparing to specialized algorithms.

For the testing, the trained agent is confronted with a new environment it has never seen before. The workspace is of the same nature as the ones used in the training phase *i.e.*, identical size and distribution of obstacles.

4.2.1 Experiment 1: Varying Movement Budget

In this experiment, we first look at the ability of the trained agent to cover a field with an energy constraint. To this end, we measure the minimum number of movements necessary on average to cover a given map. In fact, one of the motivations that encouraged this work and that was stated at the beginning was to not only develop a simple algorithm from a computational point of view with a relatively low complexity, but also minimize the use of available resources. That is why, in the learning procedure was integrated a movement budget symbolizing the power or energy available to the robot to accomplish the mission.

Figure 4.2 shows the coverage rate or the percentage of the area that was covered for three different maps of size 10×10.

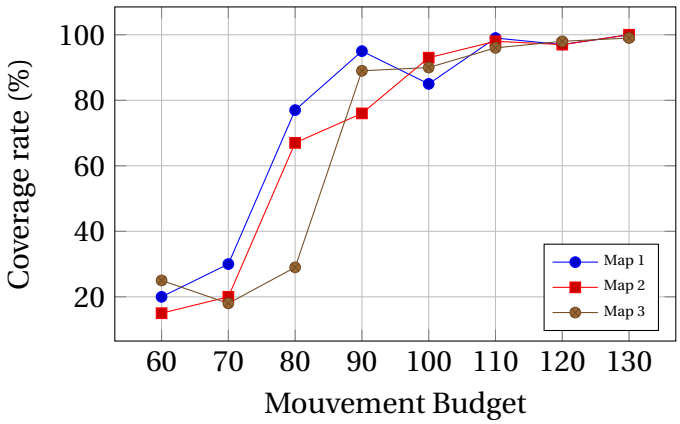


FIGURE 4.2: Percentage of covered free space obtained for different movement budgets.

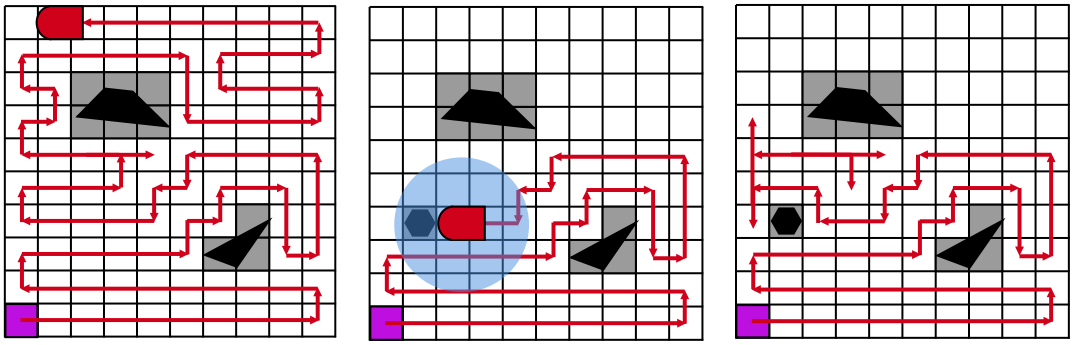
It can clearly be seen that the covered area increases with movement budget, which is expected because with an arbitrary high number of moves, the robot can always cover the entire space and the challenge of the coverage is to do in as few

moves as possible. Although, a mathematical relationship between the budget and the quality of the coverage cannot be established directly, the empirical results show that the limited number of moves for the coverage task that was part of the design of the RL agent and the DQN architecture, is a direct indicator of how efficient the trained agent.

4.2.2 Experiment 2: Dynamic Obstacles

This second experiment aims to highlight how the agent behaves when there are changes in the environment. In fact, the agent takes decisionS based on a global map of the environment that is kept in memory during the coverage and updated when changes in the workspace take place and are detected.

Two situations can be distinguished: first, when an object is introduced in an area of the free space that is out of the field of view of the robot, in which case it does not cause any changes in the behavior of the robot or the actions as the said object will not be visible on the global map until it is scanned by the sensors. The second situation corresponds to the case where an object is introduced close enough to the robot to be detected as it is shown in Figure 4.3 below.



(a) DQN trajectory (b) Introduction of an obstacle (c) Path replanning

FIGURE 4.3: DQN trajectory and obstacle avoidance

The agent is able to detect the newly introduced obstacle in the environment in its field of view and the trajectory is re-planned accordingly as seen in the left hand side map in Figure 4.3. Thanks to the choice of a global map with online local updates, new obstacles when they are spotted are dealt with as if they were present before and this is due to the fact that the agent mostly bases its decisions on local information collected by the sensors on the fly. Nevertheless, this experiment says nothing about deadlock situations where the trained agent finds itself surrounded either by obstacles of already covered areas. Although the agent will manage to get out of the impasse, there is no guarantee as to the choice of actions or trajectory.

4.2.3 Experiment 3: Human Control vs DQN

Comparing the DQN agent to other algorithms (see next section) might give an idea on its efficiency, but the validity of the comparison is questionable, given the differences that exist between the currently existing approaches in terms of complexity, initial assumptions, etc. That is why in this section, we chose to confront

the agent with human operators in order to evaluate the performance of the proposed strategy with respect to manual control by a person. This is relevant in the sense that autonomous robots that have to cover surfaces for instance for cleaning, gardening, mine clearance or painting purposes, it is crucial to check whether the proposed solution is able to outperform a human operator or not and if automating the coverage with a DQN leads to better results than a robot guided by a human being. To answer this question, we let 5 people play the coverage game and compared their performance to the one of the deep Q-Network.

For this experience, a platform was developed[1] with a simple GUI allowing the user to have a view of the environment and to select four actions (up, down, left, right) as shown in the figure below. The environment is generated in an identical way as explained previously, and contains obstacles and cells marked with different colors identifying their state.

Results listed in Table 4.2 show that when the task is carried out manually by a human operator, the traveled distance is on average higher, and even if the person has a global view of the entire workspace and can plan ahead, this does not always help make better decisions.

	Human Control	**DQN**
Number of moves	99	**97**
Number of revisited cells	8	**6**
Time (s)	52	**34**

TABLE 4.2: Human vs DQN

For instance, given the environment in Figure A.3 with three obstacles, the five players were able to cover the space in comparable time but required on average an higher number of moves, thus resources, compared to the DQN agent.

4.2.4 Comparison & Discussion

The results listed in Figure 4.3 include the average percentage of the total covered area as well as the standard deviation, both computed for a complete testing session on the same set of environments for comparison with other algorithms.

Algorithm	**Coverage rate (in %)**	**Standard Deviation**	**Revisited cells rate (in %)**
Random	23	11	13
Spiral Motion	62	3	2
Boustrophedon Motion	68	8	3
DQN	**82**	**7**	**2**

TABLE 4.3: Average coverage rates for various CCPP algorithms.

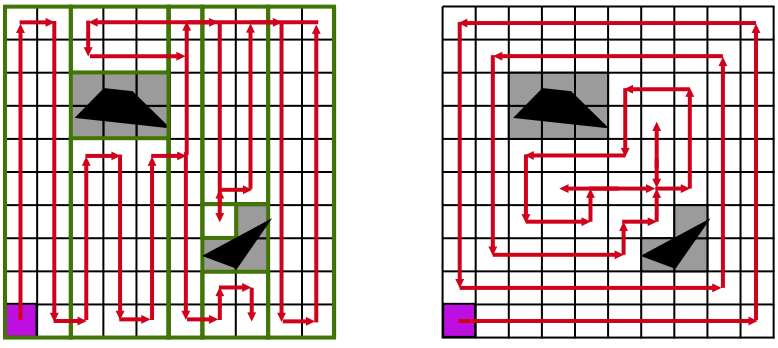
A constraint is imposed on the maximum number of possible actions that can be taken.

[1] https://github.com/omar-bfs/PyBot

The first important parameter directly linked to the optimality of the coverage is the *revisited cells rate*. The lower this rate, the better. The robot indeed doesn't waste time and energy by going over areas of the space that have already been explored. The DQN model seems to perform well and on average, only ≈2% of the total area has been revisited more than once. In most configurations, this percentage cannot be equal to zero, because in general, the environment doesn't admit a trajectory visiting exactly every cell only once, *i.e.*, there doesn't always exist a path that connects each of the cells of the free only once. Therefore, when planning a path, even when a global map is available it is inevitable to revisit certain cells.

For the DQN algorithm, no decomposition is needed and the action which is considered to be the best in view of the available state input is taken. The DQN agent used 91 moves to complete coverage. This shows that the solution provided by the trained agent can be of very good quality, especially compared to the other algorithms. However, it is not always the case as there is no guarantee of optimality of the trajectory.

Regarding the spiral motion trajectory shown in Figure 4.4 (b), 95 moves were necessary to cover the 91 reachable cells, whereas the Boustrophedon motion covered the space with 99 moves. The difference explaining these results lies in the fact that when moving from one sub-region to another in the Boustrophedon approach, additional moves that could have been avoided are often required. This is directly linked to how the space is covered, i.e. in what order the the sub-regions areas covered. The principle of the boustrophedon pattern is based on a cellular decomposition of the space into adjacent regions free of obstacles. Moreover, there are countless ways of dividing up the workspace, each leading to different results of varying quality even for the same environment. The choice of which decomposition to chose is not trivial [61]. Depending on how the space is decomposed into sub-regions (shown in green rectangles in Figure 4.4 (a) below), the passage from one region to another is often expensive for the agent as it has to go over multiple cells that have already been visited. This aspect makes the cellular decomposition less efficient although simple to apply in practice.



(a) Boustrophedon motion (b) Spiral motion

FIGURE 4.4: Path of the robot obtained with two approaches.

4.3 Obstacle Detection and Map Updating

The vision algorithm outputs the segmented image showing the objects detected in the image (see figures below). With each point of the image recorded by the camera, is associated a vector containing the coordinates (x, y, z) expressed in the frame of the camera.

FIGURE 4.5: Left to right. Raw image of the office at the company captured by the 3D ZED camera. RGB segmented image also containing information on how far objects are.

Later, several point clouds which correspond to different objects are formed. However, some points have invalid coordinates as seen in 4.6. These points are filtered by a simple conditional test. The point clouds obtained as a result of these operations are then published on topics corresponding to their class (for example, grass area, tree, etc). Using Rviz, we can then visualize these point clouds.

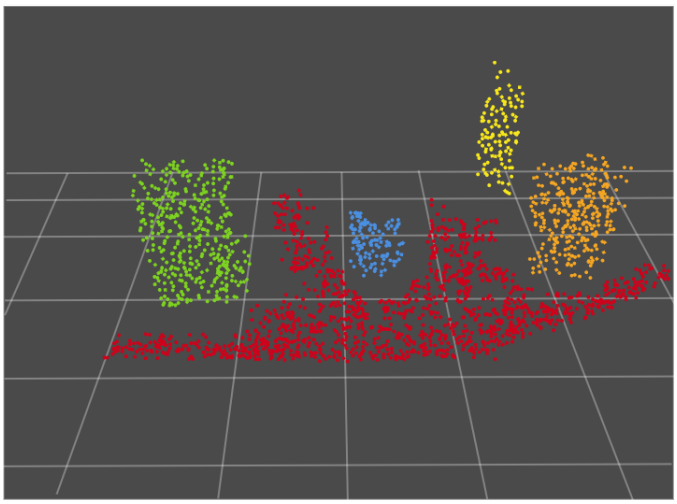


FIGURE 4.6: Segmented point cloud visualized with Rviz.

The point cloud thus constructed is a good starting point, but it is not yet exploitable as such by the path planning algorithm to update the map. Indeed, the

cloud is very noisy, and contains several additional outliers. This phenomenon is shown in Figure 4.7. In this figure, the points circled in pink are outliers, they must therefore be removed.

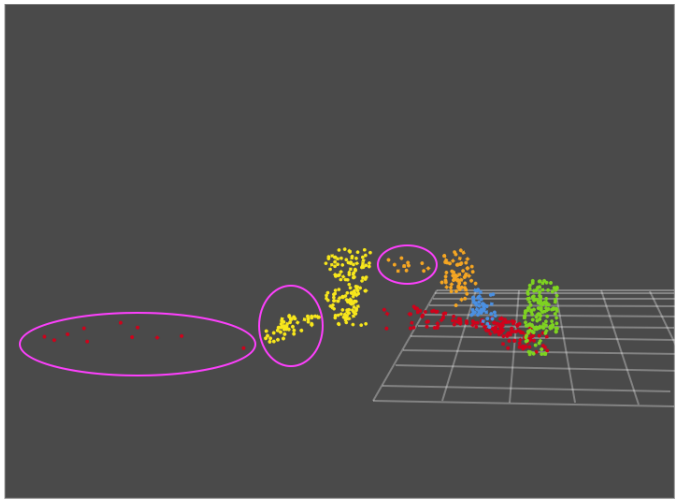


FIGURE 4.7: Point cloud, seen in profile - observation of outliers.

The filter applied to remove outliers from the point cloud is an out-of-the-box filter available on the Point Cloud Library (PCL), called *Statistical Outliers Removal filter*. This filter is commonly used to process data captured by sensors such as LiDARs [62]. It removes all points that are too far from the rest of the points. The underlying working principle of this filter is the k nearest neighbors [63]. This cloud of points corresponds to the scene observed by the camera. A last step is required for it to be usable by the path planning and map updating algorithm. This step aims to project the 3D point cloud on the horizontal plane (x, y) to obtain a 2D point cloud. Using this information, the robot is able to scan its surroundings, build a map, and update the value of the cells.

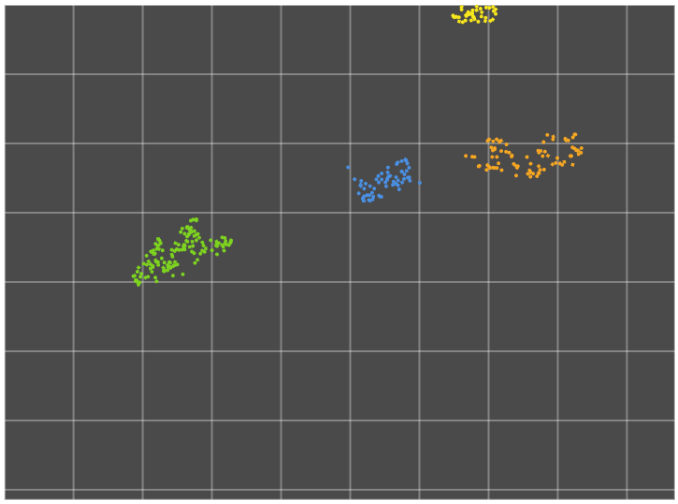


FIGURE 4.8: 2D projection of the point cloud.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

This work examines the problem of complete coverage path planning in unknown environments, with the aim of maximizing the covered area, which is important for diverse applications involving intelligent autonomous systems such as military and humanitarian demining, surveillance and patrolling, or for a more domestic use as in cleaning and gardening.

This thesis presents the different possibilities to solve the problem and several strategies were investigated to accomplish the complete coverage task. Different critical aspects of the question, were taken into account, in particular the constraint of partial visibility of the agent, because of the limited knowledge of the environment, the complexity of the calculations as well as that of the deployment on the real system. The framework was laid using Markov Decision Processes and we've shown that the coverage question boils down to solving a Reinforcement Learning problem. The task was expressed as a game played by a RL agent with the goal of maximizing a carefully chosen score or reward.

The proposed algorithm incorporates the local perceptual knowledge gathered by the sensors, into a global dynamic map, which describe the state of the system. The Reinforcement Learning agent was trained on a suitable data set that was generated. The dynamics of the agent (or robot) are defined through the Deep Reinforcement Learning model proposed allowing the agent to learn a strategy, leading to a convergence of the system towards the action policy for complete coverage. Later, a simulation environment was created to provide the agent training data.

We demonstrated a possible application of recent developments of AI techniques on a challenging robotics tasks like complete coverage. Besides the robustness and easiness of development, one other advantage of the structure of the solution is that it is very flexible and may still be subject to improvement.

Later, this work investigated the vision part of the problem by exposing a system was proposed as a way of observing the space and performing the changes on the dynamic map of the environment. Simulations show that it is possible to create the local map using the camera as a main sensor. Simulation results show that the motion controller yields encouraging results, making the system take actions that maximize coverage and avoid obstacles. More specifically, over 80% of the area is covered on average, with no prior knowledge on the environment.

This can be considered an essential step towards the goal of applying AI in general and RL in particular to real world robotics problems.

5.2 Limitations

As it has been explained throughout the report, this work has been carried out based on several hypotheses. Regarding the environment we chose to study, we assumed it to be of rectangular shape but this does not limit the applicability of the solution to other shapes as long as the space is appropriately discretized. So, this is not a strong assumption as the algorithm can generalize well straightforwardly to other forms. However, real workspaces are rarely perfect 2D planar surfaces. In fact, there are irregularities which create a difference in level or slopes which can be dangerous for the system if no safety measures are in place< In addition, the impact of these irregularities on the coverage performance is uncertain and yet to be determined.

Another assumption consisted in supposing the position of the agent perfectly known at all times. Although it is rarely possible to determine position of the robot with exactitude, this hypotheses remains realistic as the existing localization techniques are reliable and relatively robust to measurement noise. However, it would be interesting to integrate and test a given technique.

Although the simulation results were presented and discussed, it is tricky to have a final say on the ”performance” of the solution, especially because it is an ambiguous concept which may very well be related to the time or memory complexity, or even the energy consumption. Unlike other research studies usually formulating the problem in a way that only one aspect of optimized such as in [64] where the authors developed an algorithm covering the environment with areas that have the different coverage requirements, this study examined the problem of finding an ideal path and presented a viable solution by accounting equally and for as much as possible for the reduction of the energy consumption through the movement budget, the computational complexity by avoiding an exhaustive search on the space of possible solutions, along with a the quality of coverage, measured by the coverage rate and the number of revisited cells.

5.3 Future Directions

Another possible future direction of work could consist in first considering another discretization procedure of the space and possibly a continuous one, for a more realistic representation and modeling of the environment. Additionally, we can take into consideration the uncertainties on the values of the measured parameters in the model, such as the location of the agent [65], in order to investigate the impact of the presence of noise on the performance of the motion controller [66], [67]. For this purpose, an estimation or filtering scheme that utilize real-time measurements of different parameters such as the location. The effect of reducing the visibility of the robot through a more restricted scanned area is not treated, but it would be interesting to see how it influences the decisions taken locally by the agent. The main focus of this project was to design and optimize the software system. While the project tried to not only restrict the study to simulations and use the actual available hardware. As a consequence, the model has not been tested on a real robotic system unfortunately. This can in fact be very interesting to try in the future to check how well the proposed architecture performs in practice.

Appendix A

Figures

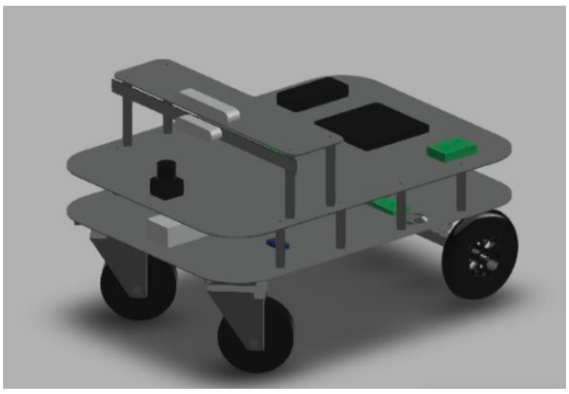


FIGURE A.1: 3D representation of the mobile robot. Dimensions: 70 cm×40 cm.

The robot is equipped with a differential drive in order to enable in-place rotations which are important for the orthogonal displacement movements.

FIGURE A.2: 2K Stereo Camera for Depth Sensing and Spatial AI.

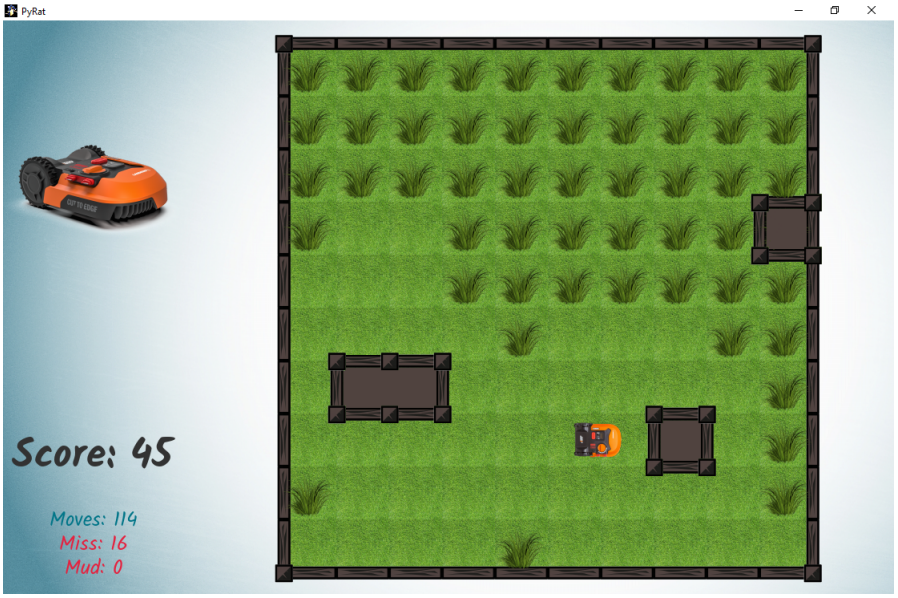


FIGURE A.3: GUI of the coverage game with a 10x10 grid map.

Bibliography

[1] "Bosch Introduces New Autonomous Robotic Lawnmower". In: *IEEE Spectrum: Technology, Engineering, and Science News*. (2020). URL: https://spectrum.ieee.org/automaton/robotics/home-robots/bosch-introduces-new-autonomous-robotic-lawnmower.

[2] Enric Galceran and Marc Carreras. "A survey on coverage path planning for robotics". In: *Robotics and Autonomous systems* 61.12 (2013), pp. 1258–1276.

[3] Amna Khan, Iram Noreen, and Zulfiqar Habib. "On Complete Coverage Path Planning Algorithms for Non-holonomic Mobile Robots: Survey and Challenges." In: *J. Inf. Sci. Eng.* 33.1 (2017), pp. 101–121.

[4] S. Dogru and L. Marques. "Energy Efficient Coverage Path Planning for Autonomous Mobile Robots on 3D Terrain". In: *2015 IEEE International Conference on Autonomous Robot Systems and Competitions*. 2015, pp. 118–123.

[5] Carmelo Di Franco and Giorgio Buttazzo. "Energy-aware coverage path planning of UAVs". In: *2015 IEEE international conference on autonomous robot systems and competitions*. IEEE. 2015, pp. 111–117.

[6] F. Yasutomi, M. Yamada, and K. Tsukamoto. "Cleaning robot control". In: *Proceedings. 1988 IEEE International Conference on Robotics and Automation*. Los Alamitos, CA, USA: IEEE Computer Society, Apr. 1988, pp. 1839, 1840, 1841. DOI: 10.1109/ROBOT.1988.12333.

[7] Ruixia Xu. "Path planning of mobile robot based on multi-sensor information fusion". In: *EURASIP Journal on Wireless Communications and Networking* 2019 (Dec. 2019). DOI: 10.1186/s13638-019-1352-1.

[8] C. Wang, J. Wang, and X. Zhang. "Autonomous navigation of UAV in large-scale unknown complex environment with deep reinforcement learning". In: *2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. 2017, pp. 858–862.

[9] I. Gavrilut et al. "Vision based algorithm for path planning of a mobile robot by using cellular neural networks". In: *2006 IEEE International Conference on Automation, Quality and Testing, Robotics*. Vol. 2. 2006, pp. 306–311.

[10] W. Huang, A. Osothsilp, and F. Pourboghrat. "Vision-based path planning with obstacle avoidance for mobile robots using linear matrix inequalities". In: *2010 11th International Conference on Control Automation Robotics Vision*. 2010, pp. 1446–1451.

[11] M. Emharraf et al. "Mobile robot: Simultaneous localization and mapping of unknown indoor environment". In: *2015 International Conference on Electrical and Information Technologies (ICEIT)*. 2015, pp. 1–6.

[12] A. Elfes. “Using occupancy grids for mobile robot perception and navigation”. In: *Computer* 22.6 (1989), pp. 46–57.

[13] Wei Yue, Xianhe Guan, and Liyuan Wang. “A novel searching method using reinforcement learning scheme for multi-uavs in unknown environments”. In: *Applied Sciences* 9.22 (2019), p. 4964.

[14] F. Niroui et al. “Deep Reinforcement Learning Robot for Search and Rescue Applications: Exploration in Unknown Cluttered Environments”. In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 610–617.

[15] Mirco Theile et al. “UAV Coverage Path Planning under Varying Power Constraints using Deep Reinforcement Learning”. In: *ArXiv* abs/2003.02609 (2020).

[16] Thibault De Swarte, Omar Boufous, and Paul Escalle. “Artificial intelligence, ethics and human values: the cases of military drones and companion robots”. In: *Artificial Life and Robotics* 24.3 (2019), pp. 291–296.

[17] Anne Håkansson. “Portal of Research Methods and Methodologies for Research Projects and Degree Projects”. In: 2013.

[18] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*. 2nd. Athena Scientific, 2000. ISBN: 1886529094.

[19] Douglas John White. *Dynamic programming*. Tech. rep. 1969.

[20] Peter Whittle. *Optimization over time*. John Wiley & Sons, Inc., 1982.

[21] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, Inc., 1994. ISBN: 0471619779.

[22] Richard Bellman. “The theory of dynamic programming”. In: *Bulletin of the American Mathematical Society* 60.6 (Nov. 1954), pp. 503–515. URL: https://projecteuclid.org:443/euclid.bams/1183519147.

[23] Jens Kober and Jan R Peters. “Policy search for motor primitives in robotics”. In: *Advances in neural information processing systems*. 2009, pp. 849–856.

[24] Christopher JCH Watkins and Peter Dayan. “Q-learning”. In: *Machine learning* 8.3-4 (1992), pp. 279–292.

[25] G. A. Rummery and M. Niranjan. *On-Line Q-Learning Using Connectionist Systems*. Tech. rep. 1994.

[26] Vijay R Konda and John N Tsitsiklis. “Actor-critic algorithms”. In: *Advances in neural information processing systems*. 2000, pp. 1008–1014.

[27] Ivo Grondman et al. “A survey of actor-critic reinforcement learning: Standard and natural policy gradients”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42.6 (2012), pp. 1291–1307.

[28] Volodymyr Mnih et al. “Asynchronous methods for deep reinforcement learning”. In: *International conference on machine learning*. 2016, pp. 1928–1937.

[29] Lucian Busoniu et al. *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press, Jan. 2010. ISBN: 1439821089. DOI: 10.1201/9781439821091.

[30] Prajit Ramachandran, Barret Zoph, and Quoc V Le. "Searching for activation functions". In: *arXiv preprint arXiv:1710.05941* (2017).

[31] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[32] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[33] Jason Yosinski et al. *How transferable a re features in deep neural networks?* 2014. arXiv: 1411.1792 [cs.LG].

[34] Sepp Hochreiter. "Untersuchungen zu dynamischen neuronalen Netzen". In: *Diploma, Technische Universität München* 91.1 (1991).

[35] Sepp Hochreiter et al. *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies*. 2001.

[36] Yoav Gabriely and Elon Rimon. "Spanning-tree based coverage of continuous areas by a mobile robot". In: *Annals of Mathematics and Artificial Intelligence* 31 (2001), p. 2001.

[37] Howie Choset. "Coverage of Known Spaces: The Boustrophedon Cellupdar Decomposition". In: *Autonomous Robots* 9 (Jan. 2000), pp. 247–253.

[38] E. U. Acar and H. Choset. "Critical point sensing in unknown environments". In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*. Vol. 4. 2000, 3803–3810 vol.4.

[39] Zuo Llang Cao, Yuyu Huang, and Ernest L. Hall. "Region filling operations with random obstacle avoidance for mobile robots". In: *Journal of Robotic Systems* 5.2 (1988), pp. 87–102. DOI: 10.1002/rob.4620050202. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.4620050202. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.4620050202.

[40] Z. J. Butler, A. A. Rizzi, and R. L. Hollis. "Contact sensor-based coverage of rectilinear environments". In: *Proceedings of the 1999 IEEE International Symposium on Intelligent Control Intelligent Systems and Semiotics (Cat. No. 99CH37014)*. 1999, pp. 266–271.

[41] L. Garrote et al. "Mobile Robot Localization with Reinforcement Learning Map Update Decision aided by an Absolute Indoor Positioning System". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2019, pp. 1620–1626.

[42] E. Garcia and P. Gonzalez de Santos. "Mobile-robot navigation with complete coverage of unstructured environments". In: *Robotics and Autonomous Systems* 46.4 (2004), pp. 195–204. ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2004.02.005. URL: http://www.sciencedirect.com/science/article/pii/S0921889004000235.

[43] Lei Tai, Giuseppe Paolo, and Ming Liu. “Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2017), pp. 31–36.

[44] Chaomin Luo and Simon X. Yang. “A Bioinspired Neural Network for Real-Time Concurrent Map Building and Complete Coverage Robot Navigation in Unknown Environments”. In: *IEEE Transactions on Neural Networks* 19 (2008), pp. 1279–1298.

[45] Weibo Huang Hong Liu Jiayao Ma. “Sensor-based complete coverage path planning in dynamic environment for cleaning robot”. English. In: *CAAI Transactions on Intelligence Technology* 3 (1 Mar. 2018), 65–72(7). URL: https://digital-library.theiet.org/content/journals/10.1049/trit.2018.0009.

[46] Susan Hert, Sanjay Tiwari, and Vladimir Lumelsky. “A Terrain-Covering Algorithm for an AUV”. In: *Underwater Robots*. Ed. by Junku Yuh, Tamaki Ura, and George A. Bekey. Boston, MA: Springer US, 1996, pp. 17–45. ISBN: 978-1-4613-1419-6. DOI: 10.1007/978-1-4613-1419-6-2. URL: https://doi.org/10.1007/978-1-4613-1419-6_2.

[47] P. W. Tse et al. “Design of a navigation system for a household mobile robot using neural networks”. In: *1998 IEEE International Joint Conference on Neural Networks Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98CH36227)*. Vol. 3. 1998, 2151–2156 vol.3.

[48] R. Andrew Russell. “Mobile robot guidance using a short-lived heat trail”. In: *Robotica* 11.5 (1993), pp. 427–431. DOI: 10.1017/S0263574700016970.

[49] Daisuke Kurabayashi et al. “Cooperative Sweeping in Environments with Movable Obstacles”. In: *Distributed Autonomous Robotic Systems 2*. Ed. by Hajime Asama et al. Tokyo: Springer Japan, 1996, pp. 257–267. ISBN: 978-4-431-66942-5.

[50] Jeff Heaton. *Introduction to neural networks with Java*. Heaton Research, Inc., 2008.

[51] Howard B. Demuth et al. *Neural Network Design*. 2nd. Stillwater, OK, USA: Martin Hagan, 2014. ISBN: 0971732116.

[52] Greg Brockman et al. *OpenAI Gym*. 2016. arXiv: 1606.01540 [cs.LG].

[53] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. ISSN: 00280836. URL: http://dx.doi.org/10.1038/nature14236.

[54] Jian Jin. “Optimal field coverage path planning on 2D and 3D surfaces”. In: (2009).

[55] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[56] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. URL: http://incompleteideas.net/book/the-book-2nd.html.

[57] Chunqing Gao et al. “Optimal Multirobot Coverage Path Planning: Ideal-Shaped Spanning Tree”. In: *Mathematical Problems in Engineering* 2018 (Sept. 2018), pp. 1–10. DOI: 10.1155/2018/3436429.

[58] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: 1312.5602 [cs.LG].

[59] C. Cadena et al. “Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age”. In: *IEEE Transactions on Robotics* 32.6 (2016), pp. 1309–1332.

[60] Nuwan Ganganath and Henry Leung. “Mobile robot localization using odometry and kinect sensor”. In: *2012 IEEE International Conference on Emerging Signal Processing Applications*. IEEE. 2012, pp. 91–94.

[61] F. Lingelbach. “Path planning using probabilistic cell decomposition”. In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA ’04. 2004*. Vol. 1. 2004, 467–472 Vol.1.

[62] André Carrilho, Mauricio Galo, and Renato Dos Santos. “Statistical outlier Detection Method for Airborne LiDAR Data”. In: vol. XLII-1. Sept. 2018. DOI: 10.5194/isprs-archives-XLII-1-87-2018.

[63] T. Cover and P. Hart. “Nearest neighbor pattern classification”. In: *IEEE Transactions on Information Theory* 13.1 (1967), pp. 21–27.

[64] Claudio Piciarelli and Gian Foresti. “Drone patrolling with reinforcement learning”. In: Sept. 2019, pp. 1–6. ISBN: 978-1-4503-7189-6. DOI: 10.1145/3349801.3349805.

[65] Shoudong Huang and Gamini Dissanayake. “Robot Localization: An Introduction”. In: Aug. 2016. DOI: 10.1002/047134608X.W8318.

[66] Dongsheng Guo et al. “A New Noise-Tolerant Obstacle Avoidance Scheme for Motion Planning of Redundant Robot Manipulators”. In: *Frontiers in Neurorobotics* 12 (2018), p. 51. ISSN: 1662-5218. DOI: 10.3389/fnbot.2018.00051. URL: https://www.frontiersin.org/article/10.3389/fnbot.2018.00051.

[67] Jia Pan and D. Manocha. “Fast and Robust Motion Planning with Noisy Data using Machine Learning”. In: *Proceedings of the 30th International Conference on Machine Learning*. Citeseer. 2013.