Multiple Mini-Robots Navigation Using Reinforcement Learning

A Thesis

submitted to the designated
by the General Assembly of Special Composition
of the Department of Computer Science and Engineering
Examination Committee

by

Piyabhum Chaysri

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

WITH SPECIALIZATION
IN TECHNOLOGIES - APPLICATIONS

University of Ioannina
May 2018

Dedication

For my family.

Acknowledgements

First and foremost, I would like to thank my advisor prof. Konstantinos Blekas for his guidance, patience and understanding throughout my research. Also I would like to thank prof. Vlachos Kostas for his advise and expertise in robotics to help me achieve what have not been done before in a simulator. And I would like to express my utmost gratitude to all the staffs in department of Computer Science & Engineering, University of Ioannina for all your guidance throughout my studying period from bachelor to master degree. And I'd like to thank so many people who help me throughout my stay in Greece. Ever since I step my foot in this country, the experience is overwhelmingly joyful. Thank you, Centre for the Study of the Hellenic Language and Culture (HeLaS) for Greek language courses and those opened the door for me to study in Greece. I would like to thank prof. Dimitrios Noutsos and his family who have always been there for me and my sister in time of need. Thank Mrs. Vasiliki Androutsou for everything. Finally, I would like to thank my family for all they have done for me.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Abstract

Piyabhum Chaysri, M.Sc. in Computer Science, Department of Computer Science and Engineering, University of Ioannina, Greece, May 2018.
Multiple Mini-Robots Navigation Using Reinforcement Learning.
Advisor: Konstantinos Blekas, Associate Professor.

Reinforcement Learning is a machine learning approach for constructing intelligent agents and solving control problems. The agent learns by interacting with its environment over a period of time, without relying on exemplary supervision or complete model of the environment. The task is to discover a target in the presence of obstacles by maximizing the reward signal received and constructing appropriate value functions.

In this work we focus on the navigation of mini robotic vehicles, which are equipped with two vibration motors. These mini robots are used in a simulated environment for medical applications under a microscope. A reinforcement learning multi-agent system is introduced for navigation of multiple mini robots in unknown environments. The goal is to build appropriate decision (navigation) rules for every robot in order to reach their target, and at the same time to maintain (sub) optimal navigation paths by avoiding other mobile robots. We assume that the robots start from different locations, they all share the same map and they are moving simultaneously. An appropriate reward function is used that takes into account the existence of other mini robots in its neighborhood at every step. The reinforcement learning agents were created on-line using the Q-learning algorithm. The system has been implemented in ROS (Robot Operating System) environment and has been evaluated through various noisy cases and variable number of robots.

Εκτεταμενη Περιληψη

Piyabhum Chaysri, Μ.Δ.Ε. στην Πληροφορική, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πανεπιστήμιο Ιωαννίνων, Μαΐου 2018.
Πλοήγηση Πολλαπλών Μίνι-ρόμποτ με χρήση Reinforcement Learning.
Επιβλέπων: Κωνσταντίνος Μπλέκας, Αναπληρωτής Καθηγητής.

Η ενισχυτική μάθηση, Reinfoecement Learning (RL), είναι μια κατηγορία της μηχανικής μάθησης που χρησιμοποιηθεί για την κατασκευή τους ευφυείς πράκτορες και λύνει τα προβλήματα έλεγχου. Ένας πράκτορας μαθαίνει από την αλληλεπίδραση με το περιβάλλον του χωρίς υποδειγματικό επιβλέποντα ή μοντέλο του περιβάλλοντος. Η διεργασία είναι να βρεθεί τον στόχο το οποίο βρίσκεται ανάμεσα εμπόδια από τη μεγιστοποίηση της ανταμοιβής, χτίζοντας τις κατάλληλες συναρτήσεις αξίας.

Η εργασία πραγματεύεται την αυτόνομη πλοήγηση μίνι-ρομποτικών συστημάτων, τα οποία εξοπλισμένα με δύο μικρές κινητήρες δόνησης. Τα οποία είναι πλατφόρμα μικρομετατοπίσεων με φυγοκεντρικούς ενέργειας. Αυτά τα τα μίνι-ρομπότ χρησιμοποιούνται σε προσομοιώσεις περιβάλλοντα για ιατρικές εφαρμογές υπό μικροσκόπιο. Παρουσιάζεται η ενισχυτική μάθηση πολλαπλών πρακτόρων συστημάτων για την πλοήγηση μίνι-ρομποτικών συστημάτων σε άγνωστα περιβάλλοντος. Ο στόχος είναι να χτίζει κατάλληλη πολιτική αποφάσεών (πλοήγηση) σε κάθε ρομπότ (πράκτορα) για να φτάνουν στους στόχους τους. Παράλληλα, να διατηρούν (υπό) βέλτιστα μονοπάτια και αποφεύγον άλλα ρομπότ τα οποία κυκλοφορούν μέσα στο ίδιο περιβάλλον. Υποθέτουμε ότι κάθε ρομπότ ξεκινούν από διαφορετικές θέσεις όμως μοιράζουν ίδιο χάρτη (χώρο διεργασία) και κινούνται ταυτόχρονα. Χρησιμοποιείται κατάλληλη συνάρτηση ανταμοιβής ή οποία αναγνωρίζει γειτονικά ρομπότ σε κάθε βήμα. Ο πράκτορας ενισχυτικής μάθησης δημιουργείται απευθείας (on-line) χρησιμοποιώντας αλγόριθμο *Q-learning* και συμμετέχει σε πολλαπλούς πράκτορες συστημάτων. Παρουσιάσουμε και την χρήση του ε-greedy στο *Q-learning*. Το οποίο είναι μια μέθοδο εξερεύνησης στο περιβάλλον με την πιθανότητα εξερεύνησης. Χρησιμοποιείται το κατάλληλο παράμετρο της πιθανότητας εξερεύνησης και μπορεί να καταλήξει στην βέλτιστη πολιτική. Κάθε ρομπότ (κάθε πράκτορες) έχουν τον δικό τους στόχο και δουλεύουν σε συνεργατική περιβάλλον. Το σύστημα έχει υλοποιηθεί μέσω ROS (Robot Operating System) και έχει αξιολογηθεί με πολλαπλές θορυβώδες περιπτώσεις και πολλαπλά ρομπότ.

Chapter 1

Introduction

One of the major scientific challenges of today is to decode the human intelligence and develop it in artificial systems. We always challenge ourselves to unlock the way human learns and replicate it with a machine. The strive to create a machine that can learn leads to the development of artificial intelligent or AI as we know it today. In Artificial Intelligence (AI) [2], an intelligent agent is typically an autonomous entity, which observes and acts upon a typically unknown environment, directing its activity towards achieving specific goals. The majority of the presented artificial intelligence agents are designed based on the same concept. They accept stimulus from the environment and react according to the history of stimuli they have received. Their difference originates from the fact that a number of different internal structures are used for the processing of the newly arrived information that are generated from the interaction with the environment. Numerous intelligent agents have been proposed so far in a wide range of fields such as: robotics, gaming, navigation, etc. (see figure 1.1 for some indicative examples).

Figure 1.1: Intelligent agents

1.1 Machine Learning on Intelligent Agents

Machine learning plays a crucial role in development of artificial intelligence, as the learning capability constitutes and integral part of an intelligent system. During the last decades a variety of machine learning techniques have been used extensively for the development of advanced decision making mechanism, which can be considered as the agent's core.

Machine learning Machine learning can be categorized into three main subfields according to the problem under consideration: i) *supervised learning*, ii) *unsupervised learning*, iii) *reinforcement learning*. *Supervised learning* refers to the problems where the desired outputs corresponding to some input data are known in advance. Roughly speaking, it can be seen as an explicit teacher is available. In the case where the desired output consists of discrete values, the problem is known as *classification*. The objective in classification problems is to assign an instance to one of a finite set of discrete class labels. On the other hand, if the desired output consists of continuous values, the task is called *regression*. In regression, the goal is the prediction of the output value of an unknown input instance. In contrast to supervised learning, in *unsupervised learning* no knowledge about the target values is supplied. The goal in this case is the discovery of similar groups within the data, this is called *clustering*, or the determination of the data distribution within the input space, known as *density estimation*.

Reinforcement learning (RL) has been demonstrated that constitutes a suitable platform for the development of intelligent agents. In RL, in which this thesis focuses on, an agent interacts with an initially unknown environment and modifies its behaviour (policy) so as to maximize its cumulative rewards. *Reinforcement learning* is also exceptionally good at solving unpredictable and non-linear problem as in real-world environments and applications.

One of the skills that human processed is self-learning or self-taught and we are trying to replicate this behaviour in intelligent agents or robots. Since the core of self-learning robot is for it to learn by interacting with the environment, we can find the framework for intelligent agent development of this type in *reinforcement learning*.

1.1.1 Intelligent Agents

By definition [3] an agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives. An intelligent agent [4] can be a program or a robot which able to interact with its environment and make intelligent decisions by using its features. To put it simply the agents interact with the environment and follow a set of command to react with the environment by predetermined action programmed by human so they can perform the task without intervention. The things that set apart intelligent agents from mere agents are capability of making "*rational*" decisions and actions and ability to optimize their decisions based on interaction with the environment. Upon interacting with an environment the agent receives reward to improve future decisions. We call this self-learning automaton an "*intelligent agent*".

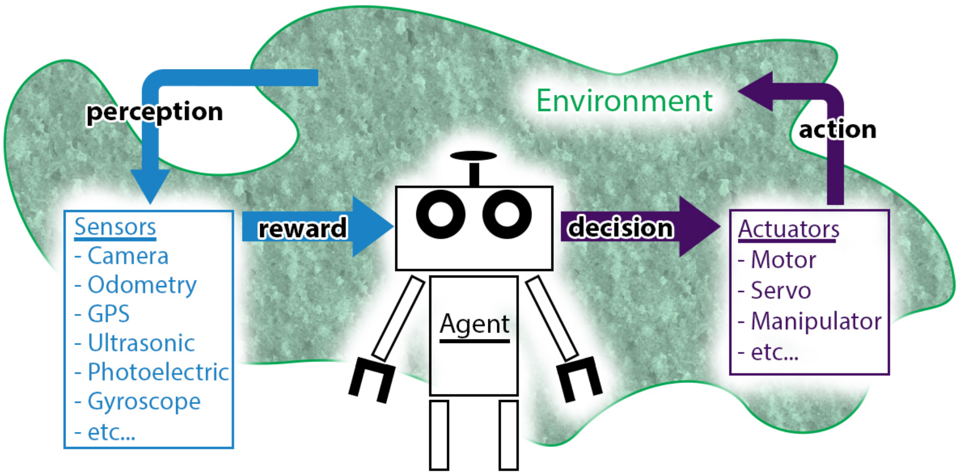


Figure 1.2: A simple depiction of an intelligent agent.

As depicted in figure 1.2 an agent perceives the changes in the environment by **perception** with sensors or features, then a **reward** is given to the agent to make **decision** and subsequently commit an **action** towards the environment then the process is repeated from the **perception**. This agent not only can learn to interact with the environment but can also interact with other agent as well leading to multi-robot system where the agents learn by interacting with both the environment and other robots.

1.1.2 Multiagent Systems

Agents operate and exist in some environment, which typically is both computational and physical. The environment might be open or closed, and it might or might not contain other agents. Although there are situations where an agent can operate usefully by itself, the increasing interconnection and networking of computers is making such situations rare, and in the usual state of affairs the agent interacts with other agents [4]. A multiagent system (MAS) can be cooperative where all the agents are cooperating to complete a task or it can be competitive where each robot competing with the other for resource or completion of a task. From the previously seen intelligent agents, a multiagent system contains some added properties for an agent to work concurrently with other agents within the same environment. One of which being the ability to communicate or exchange the messages between agents or the way for them to interact with one another.

Characteristics of Multiagent Environments

- Multiagent environments provide an infrastructure specifying communication and interaction protocols.
- Multiagent environments are typically open and have no centralized designer.
- Multiagent environments contain agents that are autonomous and distributed, and may be self-interested or cooperative.

Coordination Protocols

In an environment with limited resources, agents must coordinate their activities with each other to further their own interests or satisfy group goals. The actions of multiple agents need to be coordinated because there are dependencies between agents' actions, there is a need to meet global constraints, and no one agent has sufficient competence, resources or information to achieve system goals. Examples of coordination include supplying timely information to other agents, ensuring the actions of agents are synchronized, and avoiding redundant problem solving [4].

Cooperation Protocols

In this thesis the agents are working primarily in cooperative fashion. We will focus on how the cooperation protocols work in general senses and adapt it to suite the learning process that we will use later in agent construction. A basic strategy shared by many of the protocols for cooperation is to decompose and then distribute tasks. Such a divide-and-conquer approach can reduce the complexity of a task: smaller subtasks require less capable agents and fewer resources. However, the system must decide among alternative decompositions, if available, and the decomposition process must consider the resources and capabilities of the agents. Also, there might be interactions among the subtasks and conflicts among the agents.

Task decomposition can be done by the system designer, whereby decomposition is programmed during implementation, or by the agents using hierarchical planning, or it might be inherent in the representation of the problem, as in an AND-OR graph. Task decomposition might be done spatially, based on the layout of information sources or decision points, or functionally, according to the expertise of available agents.

Once tasks are decomposed, they Call be distributed according to the following criteria [5].

- Avoid overloading critical resources.
- Assign tasks to agents with matching capabilities.
- Make an agent with a wide view assign tasks to other agents.
- Assign overlapping responsibilities to agents to achieve coherence.
- Assign highly interdependent tasks to agents in spatial or semantic proximity. This minimizes communication and synchronization costs.
- Reassign tasks if necessary for completing urgent tasks.

Much of traditional AI has been concerned with how an agent can be constructed to function intelligently, with a single locus of internal reasoning and control implemented in a Von Neumann architecture. But intelligent systems do not function in isolation—they are at the very least a part of the environment in which they operate, and the environment typically contains other such intelligent systems. Thus, it makes sense to view such systems in **societal** terms. Different agents can take different roles in a *society* to expand and improve the interaction with the environment. The roles in a society can be assigned by distributing tasks to each agent. The following mechanisms are commonly used to distribute tasks:

- Market mechanisms: tasks are matched to agents by generalized agreement or mutual selection (analogous to pricing commodities).
- Contract net: announce, bid, and award cycles.
- Multiagent planning: planning agents have the responsibility for task assignment.
- Organizational structure: agents have fixed responsibilities for particular tasks.

In conclusion, by dividing role of the agents in a **society**, the roles can be assigned to each agent to function together and form an organized large-scale system which can solve large amount of problems or operate in large environment efficiently. Another benefit of using MAS is scalability, where more agents can be added in to environment to improve the functionality or to optimize the outcome.

1.2 Centrifugal Force Mini Robots

The agent in this thesis is based on centrifugal-force mini robot for medical purpose. The proportion system used in this robot is vibration motor which generates the drive forward and backward for each actuator [1]. The model is based on a low-cost small robot with two actuators from vibration DC motors. This robot is designed to work in high resolution environment such as manipulation under microscope. A photo of the prototype can bee seen in Fig. 1.3.

1.2.1 Motion Principle

The underlying physics of the actuation mechanism are explained using a simplified one degree of freedom (one dof) mobile platform of mass M. The actuation mechanism employs an eccentric mass m, rotated at a constant angular speed ω_m by a

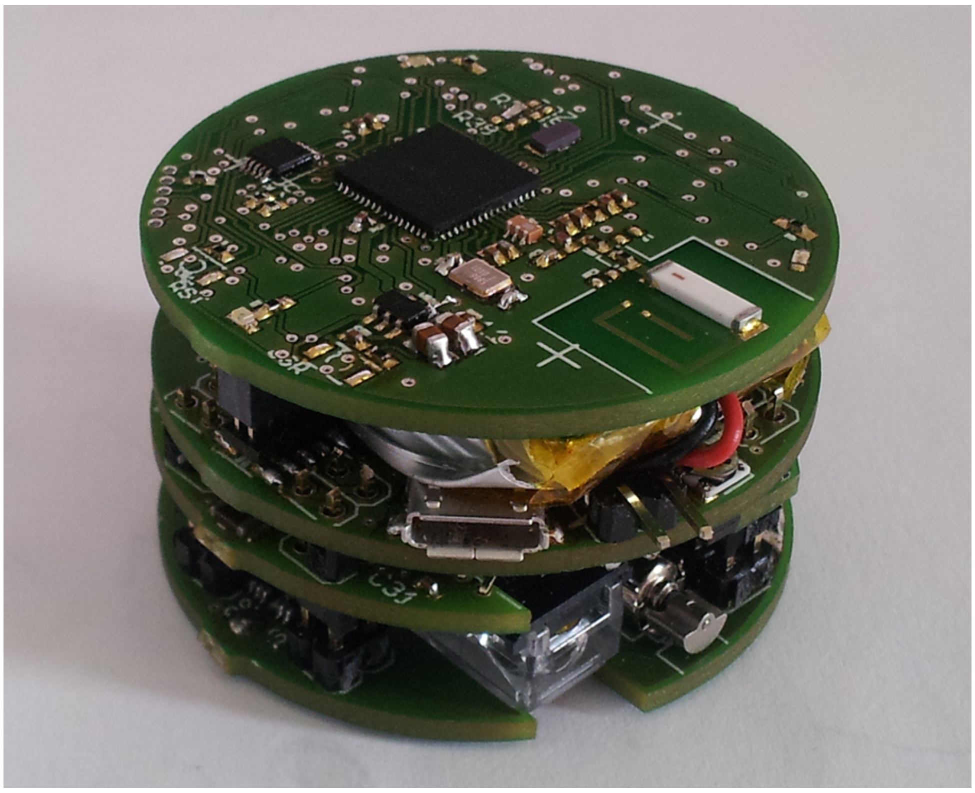


Figure 1.3: A prototype of centrifugal force mini robot.[1]

platform-mounted motor, as shown in Fig. 1.4. The actuation angle θ defines the angular position of the eccentric mass m with respect to the vertical axis, see Fig. 1.4. One cycle of operation is completed when the mass m has described an angle of $360°$. Gravitational and centripetal forces exerted on the rotating mass are resolved along the $y - z$ z axes to yield

$$\begin{aligned} f_{Oy} &= mr{\omega_m}^2 \sin\theta \\ f_{Oz} &= -mg - mr{\omega_m}^2 \cos\theta \end{aligned} \tag{1.1}$$

where g is the acceleration of gravity and r is the arm of eccentricity of m with respect to O. These forces are transmitted to the platform at point O, while the small moment due to m is neglected. When the angular speed ω_m is low, the platform does not move because the horizontal actuation force f_{Oy} is cancelled by frictional forces at the platform contact points A and B. However, if the angular speed ω_m exceeds a critical value $\omega_{m\_critical}$, then f_{Oy} overcomes the support point friction forces, and as a result, the platform begins to slide.

Using a simplified static-kinetic friction model, the motion of the platform along

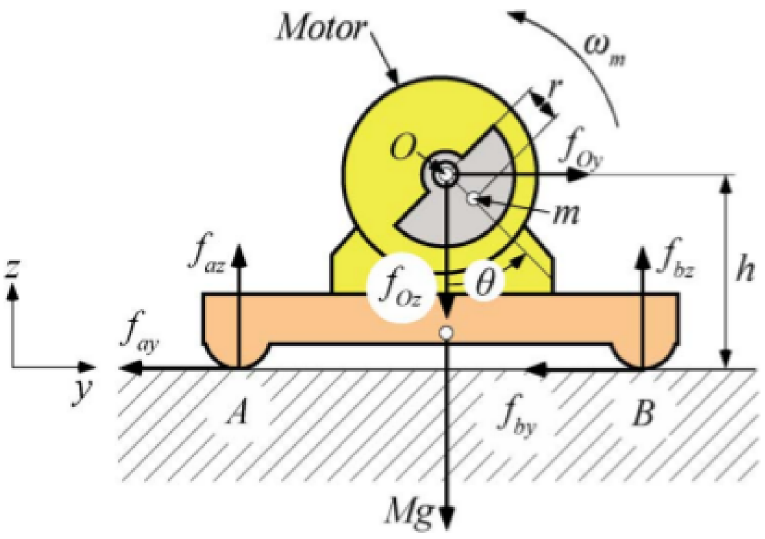


Figure 1.4: Simplified one dof platform with rotation mass m.[1]

the and axes is described by

$$\begin{aligned} M\ddot{y} &= f_{Oy} - f_{fr} \\ 0 &= f_{az} + f_{bz} + (-Mg + f_{Oz}) \end{aligned} \tag{1.2}$$

where all forces are defined in Fig. 1.4, and f_{fr} force. Neglecting viscous friction f_{fr}, is given by

$$f_{fr} = \begin{cases} f_c sgn(\dot{y}), & \dot{y}\neq 0 \\ f_{Oy}, & \|f_{Oy}\| < f_c, \dot{y} = 0, \ddot{y} = 0 \\ f_c sgn(f_{Oy}), & \|f_{Oy}\| > f_c, \dot{y} = 0, \ddot{y}\neq 0 \end{cases} \tag{1.3}$$

where f_c is the Coulomb friction level, i.e. the maximum friction force that can exist for the current normal force, and is given by

$$f_c = \mu(f_{az} + f_{bz}) = \mu(Mg - f_{Oz}) \tag{1.4}$$

The parameter μ is the coefficient of kinetic friction and the function $sgn\ (\dot{y})$ is defined by

$$sgn(\dot{y}) = \begin{cases} +1, & \dot{y} > 0 \\ 0, & \dot{y} = 0 \\ -1, & \dot{y} < 0 \end{cases} \tag{1.5}$$

The forces acting on the platform are given by (1.1), (1.3), and (1.4) and are plotted in Fig. 1.5 for three consecutive cycles.

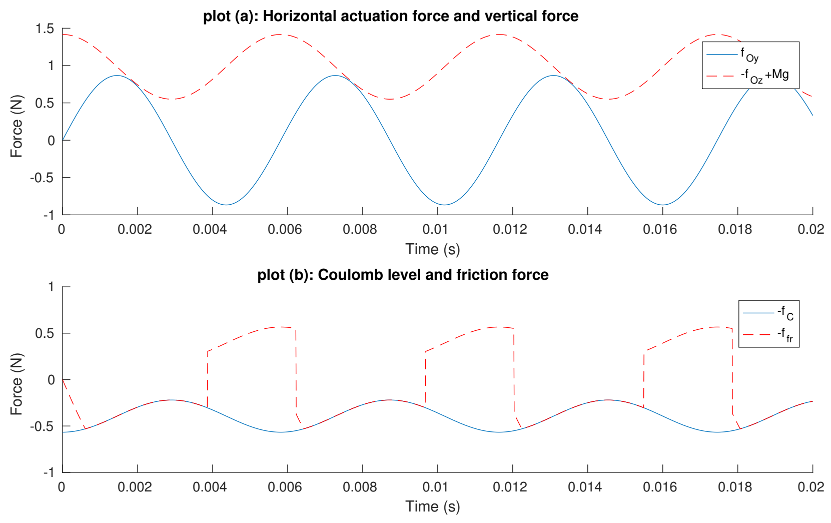


Figure 1.5: Horizontal and vertical force acting on a one dof platform.

It is observed that the horizontal actuation force f_{Oy} and the vertical actuation force f_{Oz} are time periodic and f_{Oz} leads f_{Oy} by $\pi/2$ [Fig. 1.5(a)].

Due to (1.4), the Coulomb friction level f_C is periodic too and in phase with f_{Oz}, but its sign changes from positive to negative depending on the speed direction [Fig. 1.5(b)]. This figure also shows the friction force f_{fr}. The platform's motion response caused by the forces in Fig. 1.5 is computed by numerical integration of (1.2) and is presented in Fig. 1.6.

The physics of the motion principle are explained next in more detail. Due to (1.1), when the actuation angle θ is small, the actuation force f_{Oy} is not sufficient to overcome the Coulomb level and no motion is induced. At a critical angle θ_1, the actuation force f_{Oy} overcomes the static friction limit f_C, and motion is induced (Fig. 1.6). The platform executes forward motion. When m passes the highest point at $\theta = 180°$, the platform already has a positive velocity. As m moves past this point, friction forces together with actuation forces decelerate the platform. As friction still increases, it eventually brings the platform to a stop at a critical angle θ_2 (Fig. 1.6). The actuation forces are now pointing to the left and as a result reverse platform motion starts. While m rotates to the forth quadrant of the actuation cycle, the reverse platform motion decelerates and eventually stops at critical angle θ_3 (Fig. 1.6).

Quite interestingly, as shown in Fig. 1.6(c), for a counterclockwise rotation of the eccentric load, the platform exhibits a net displacement along the positive y axis. This is due to the fact that during platform forward motion, the eccentric mass is at the

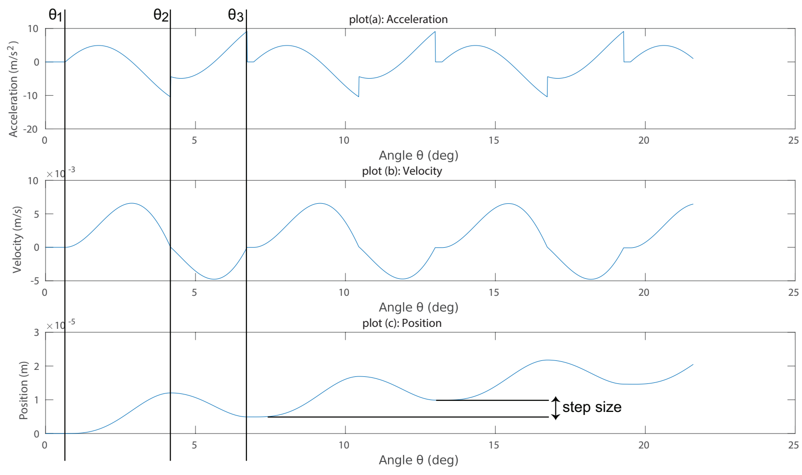


Figure 1.6: Platform motion.

higher points of its trajectory (second quadrant of actuation cycle) and, therefore, the normal forces and the frictional forces are low, whereas during the reverse motion, the mass is at the lower points of its trajectory (fourth quadrant of actua- tion cycle) and the frictional forces are high. Consequently, the platform decelerates more during reverse motion compared to forward motion and therefore—for a counterclockwise rotation of the eccentric load—a net displacement towards the positive y axis takes place.

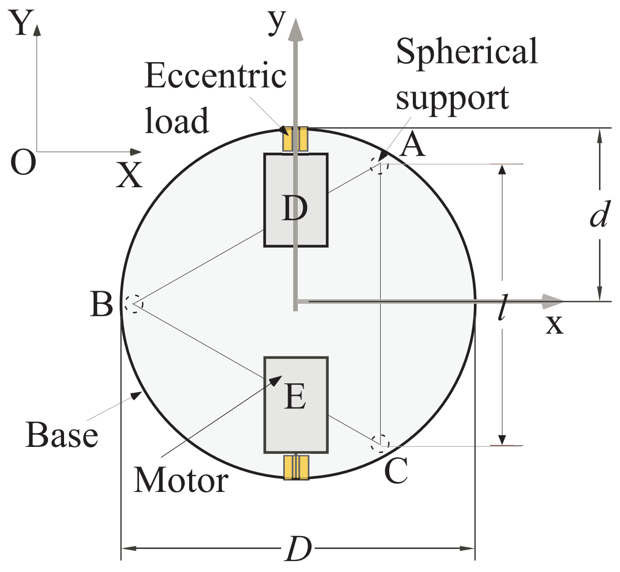


Figure 1.7: The two-actuator platform concept.[1]

1.2.2 Platform Dynamics

Two centrifugal force actuators, are employed in the design of a microrobotic platform capable of two dof planar motions, see Fig. 1.7. Although it is easier to drive the microrobot using more actuators, in the sense that there is no need for a motion planning algorithm, it would be less efficient. When more motors are used some of the components of the horizontal actuation forces cancel out each other (a null-space is generated) and consequently efficiency is reduced. Furthermore, having more motors reduces significantly the capability for miniaturization, increase the cost, and the complexity of the design. Platform base: The contact points between the platform and the ground are provided by three fixed small steel balls A, B, and C located at the vertices of an equilateral triangle (Fig. 1.7).

The length between the ball supports is l, while the radius of the platform base is d (Fig. 1.7). The three-contact point configuration is favoured because it is not over-constrained and ensures static equilibrium along the vertical axis.

Actuators: The actuation of the platform employs miniature vibrating motors. Each vibrating motor is axially coupled to an eccentric load, while the control input is the rotation speed ω_m of the motor. During motor rotation, the eccentric mass of the load generates periodic dynamic forces, which are transferred to the contact points and interact with the friction forces.

The platform dynamics can be described using the Newton–Euler formulation as following.

$$M\dot{v} = R\sum_i {}^b f_i, i = \{a, b, c, d, e\} \tag{1.6}$$

$$I_{zz}\ddot{\psi} = \hat{z}\sum_i ({}^b r_i \times {}^b f_i), i = \{a, b, c, d, e\} \tag{1.7}$$

where b is the body-fixed frame, R is the rotation matrix between frame b and the inertial frame O (see Fig. 1.7), ψ is the platform angle of rotation $v = dotx, doty, dotz]^T$, and is its center of mass (CM) velocity with respect to the inertial frame O. In (1.7),I_{zz} is the polar moment of inertia in the body fixed frame and $\hat{z}$ denotes the unit z axis vector. In both equations, the subscripts $i = \{a, b, c\}$ correspond to frictional forces at the contact points of the platform, and $i = \{d, e\}$ correspond to the forces generated by the two vibrating motors. The actuation forces that act on the platform, when the

DC micromotors rotate (assuming identical micromotors), are given by

$$\begin{aligned} {}^{b}f_{ix} &= mr\dot{\theta}_i^{\,2}\sin\theta_i \\ {}^{b}f_{iz} &= -mg_i - mr\dot{\theta}_i^{\,2}\cos\theta_i \end{aligned} \tag{1.8}$$

where $i = \{d, e\}$ and θ_i is the angle of micromotor i, m is the micromotor eccentric mass, and r is the arm of eccentricity. The dynamics of the DC micromotor are given by

$$\begin{aligned} \ddot{\theta}_i &= -\frac{b}{J}\dot{\theta}_i + \frac{k_t}{J}{}_iL_i - \frac{mgr\sin\theta_i}{J} - \frac{c}{J} \\ i_{Li} &= -\frac{k_t}{L}\dot{\theta}_i - \frac{R}{L}i_{Li} + \frac{1}{L}V_{in\_i} \end{aligned} \tag{1.9}$$

where i_{Li} is the motor i current, R is the electrical resistance, b is the viscous friction, c is the Coulomb friction at the micromotor's axis, k_t is the torque constant L, is the inductance, J is the eccentric's load moment of inertia, and V_{in_i} voltage of motor . the input i.

Using the equation (1.9) we can calculate rotational speed of the eccentric mass responding to input voltage. The nominal voltage is $850mV$ which produce the rotational speed at $1010 rad/second$ as depicted in the figure 1.8.

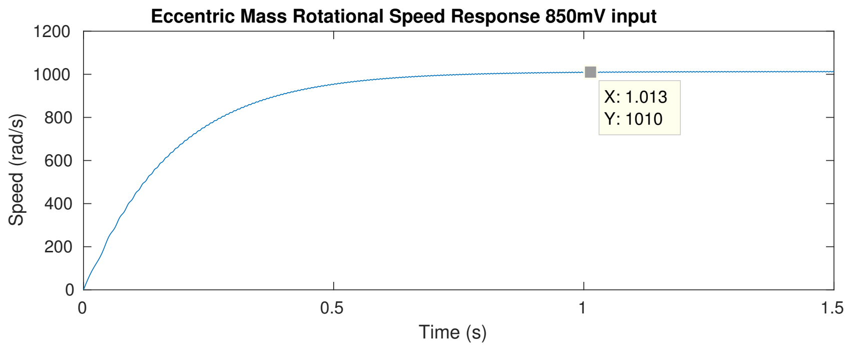


Figure 1.8: Eccentric mass rotational speed response to 850mV.

The acceleration and velocity responding to eccentric mass rotational speed are depicted in the figure 1.9, from the 0 *second* $850mV$ is given to the actuator and from the 1.5 *second* the power is shut off, given $0V$ to the actuator to simulate the time needed to stop the actuator and platform subsequently.

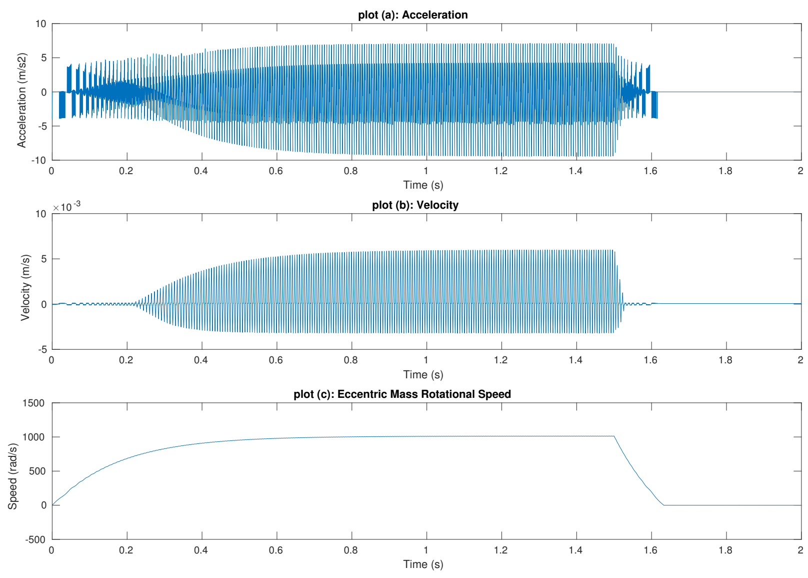


Figure 1.9: Acceleration, velocity and rotational speed respond to 850mV.

1.3 Thesis Contribution

In this thesis we propose the use of reinforcement learning for creating intelligent agents to solve multi-robot navigation problem with active collision avoidance in collaborative environment. Also the novel reward function in order to create appropriate Q value for navigation usage. Parallelism is also in consideration of the work as well, so the agents can work together and sharing their workloads by carrying out task (target) specific for each agent. The agent in this thesis is modelled after the aforementioned mini centrifugal-force robot for medical purposes. The main function of these robot is to inject medicine in to cells, one at a time with precision (see figure 1.10). The work presented in this thesis, the main goal is to create a robust algorithm, for all the agents to autonomously navigate through unknown space to their target without collision, and on optimal path. Which can solve the collaboration navigation problem and optimize the output by sharing workloads among all available agents in the environment. In chapter 3 we propose value functions for multi-robot navigation and collision avoidance.

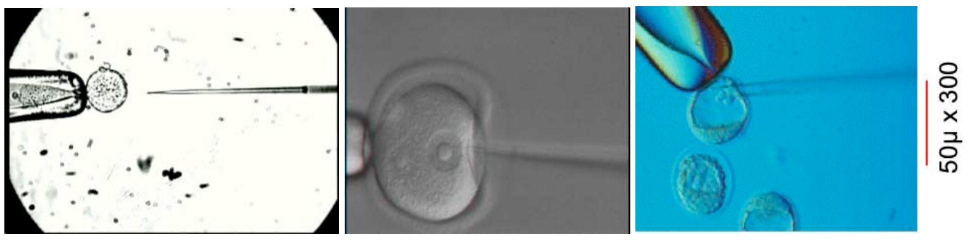


Figure 1.10: Cell injection using manipulator [1]

1.4 Thesis Layout

We start by looking at what agents and multi-agent systems are and how they works, along with what benefits multi-agent systems might bring to robotics and machine learning. Next we will look at the agent used in this thesis which is a micro-robotics platform with 2 degrees of freedom (dof). In the second chapter we will look at the reinforcement learning framework to construct an intelligent agent then the method we use to train the agents for navigation. The third chapter contains experimental results with various noise level for single agent to examine the influence of noisy environment effects on reinforcement learning. Then we will look at the experiment for single robot and four robots environment to examine the learning rate and path length corresponding to different exploration-exploitation ratio. Closing with the performance evaluation comparing the different state-action pairs and exploration probabilities for single and multi-robot navigation.

Chapter 2

Reinforcement Learning

In the previous chapter we discussed about the use machine learning on intelligent agents. Imagine an artificial intelligent (AI) that capable of learning by itself without explicit teacher to tell it what to do in every situation. Through the reinforcement learning (RL) framework we can achieve intelligent agent construction without explicit teacher or one could call it a self-learning agent or robot. However this framework is not limited to control problem in robotics but also for solving non-linear problems too.

2.1 Reinforcement Learning

As stated in N. Tziortziotis' dissertation [2], reinforcement learning (RL) [6] is a framework for constructing an intelligent agent, which capable of solving non-linear problems. In RL an autonomous agent follows a trial-and-error process to learn the optimal action to perform in each state in order to reach its goals. The agent interacts with the environment the agent learns through the reward received from each action.

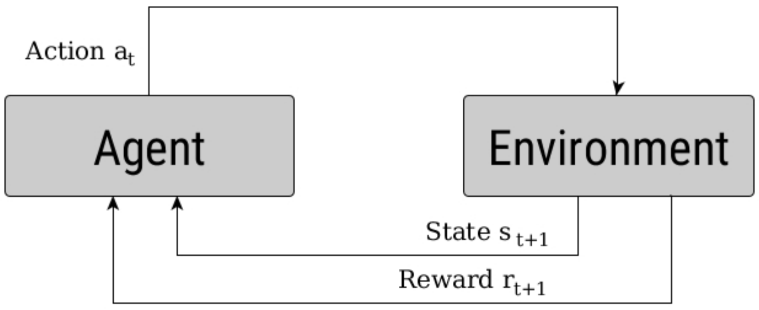


Figure 2.1: Reinforcement learning process

By repeating this process the agent eventually learns which action to take to maximize the reward to reach the goal with optimal action sequence or path. The fundamental parts of a reinforcement learning problem are the policy, the reinforcement function, the value function and the model.

Policy a decision mechanism which contains a map from states to action. To put it simply the policy dictates what action should the agent take in each state.

Reward function defines the goal of RL. It calculates the state-action pair in to one value. It decides what is good action and what is a bad action for the agent tot take.

Value function is the sum of reward for the agent in long term starting from a certain point.

Model is the environment that an agent acts upon. Usually an agent has no knowledge of the environment.

In each iteration or time t (see figure 2.1), the agent perceives its current state ($s \in S$), select an action ($a \in A$), possibly changing it's state, and receives a reward signal to corresponded state and action combination. ($r \in R$). This information is filled overtime when agent explore the environment and fill the information until it learns the optimal action for each state. This sequential decision process can be described as a Makrov Decision Process (MDP) which is typically denoted as a tuple $M = \{S, A, T, R, \gamma\}$, where:

- S is a finite set of state.
- A is a finite set of actions per state.
- $T : S \times A \to P(S)$, is the transition probability kernel $P(S)$ assigns to each state-action pair (s, a) that leads state $s' \in S$.

- $R : S \times A \times S \rightarrow \mathbb{R}$, is the immediate reward function which gives the expected reward when action $a \in A$ is selected at state $s \in S$.
- $\gamma \in (0, 1)$, is a discount factor such that rewards further in to the future are less important than immediate rewards.

For a model to be Markovian, it needs to satisfy the Markov property. Meaning the state at time $t+1$ depends only on the state t, regardless of the states in the previous times:

$$P\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t\} = P\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t, s_{t-1}, a_{t-1}, ..., s_0, t_0\} \tag{2.1}$$

2.1.1 Value Functions

One of the goals in RL is to maximize the accumulated reward [7] rather than immediate reward. The value function is used to evaluate the reward from total expected reward depends on the current state and on the selection of actions in future states. The selection of actions per state is given by a policy π is a mapping of each state $s{\in}S$ and action $a{\in}A$ to the probability $\pi(s, a)$ of taking the action a in state s. One of the goals in RL is to estimate how good to be in a state or how good the selected action is. The notion of "goodness" is defined in terms of future rewards or accumulated reward which represented as value functions. The value function is used to determine how good the state and action is according to each selected policy. The target is to find the policy that produce maximum value function. Given a state $s_t \in S$. at time t and an action $a_t \in A_{s_t}$, the agent receives a reward r_{t+1} and moves to the next state s_{t+1}. The accumulated reward R at time i can be described as:

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + ... + r_{t+i} \tag{2.2}$$

The total expected reward depends on the current state and on the selection of actions if future states. The selection in each state is given by a *policy* π which is mapping of state $s \in S$ and action $a \in A$ to the probability $\pi(s, a)$.

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.3}$$

Where γ is discount rate $0 \leqslant \gamma < 1$

Finite Horizon: the agent tries to optimize the expected accumulated rewards on the next h steps without considering what happens afterwards:

$$E(\sum_{t=0}^{h} r_t)\,. \tag{2.4}$$

In many cases when the agent continue to work for a long time or continuing task. The reward value from the equation 2.4 increases indefinitely from $T = \infty$. In order to reduce the importance of the non-immediate reward, the rewards received by the agent are geometrically reduced according to a discount factor γ, $(0 \leqslant \gamma < 1)$:

$$E(\sum_{t=0}^{h} \gamma^t r_t)\,. \tag{2.5}$$

Transition model is assumed to be Markovian, so the state transition do not depend on previous states, and the transition probabilities are given by:

$$P(s' \mid s, a) = P(s_{t+1} = (s' \mid s_t = s, a_t = a) \tag{2.6}$$

The expected reward value is:

$$R(s' \mid s, a) = E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\} \tag{2.7}$$

The value function of a state s, denoted by $V^\pi(s)$ represents the total accumulated reward that the agent can receive starting at state s and following policy π. Similarly the the value function of a state s taking action a, is denoted as $Q^\pi(s, a)$ and represents the total accumulated reward that the agent can receive starting at state s, taking action a and following a policy π. The goal is to find policy that produce maximum value function rather than maximum immediate rewards. Value function for a state s using an infinite discounted reward model:

$$V^\pi(s) = E_\pi\{R_t|s_t = s\} = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right\} \tag{2.8}$$

Value function for a state s with action a and policy $\pi(Q^\pi(s, a))$ can be expressed as:

$$Q^\pi(s, a) = E_\pi\{R_t|s_t = s, a_t = a\} = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a \right\} \tag{2.9}$$

Expanded expression for $V^{\pi}(s)$:

$$
\begin{aligned}
V^{\pi}(s) =& E_{\pi}\{R_t|s_t = s\} \\
=& E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s_t = s\right\} \\
=& E_{\pi}\left\{r_{t+1} + \sum_{k=0}^{\infty}\gamma^k r_{t+k+2}|s_t = s\right\} \\
=& \sum_a \pi(s,a)\sum_{s'}P(s' \mid s,a)\left[R(s' \mid s,a) + \gamma E_{\pi}(\sum_{k=0}^{\infty}\gamma^k r_{t+k+2}|s_t = s)\right] \\
=& \sum_a \pi(s,a)\sum_{s'}P(s' \mid s,a)\left[R(s' \mid s,a) + \gamma V^{\pi}(s')\right]
\end{aligned}
\tag{2.10}
$$

Where $\pi(s,a)$ is the probability of taking action a in state s under policy π.

The optimal value functions V^* and Q^* can be expressed recursively with the Bellman optimality equation as:

$$V^*(s) = \max_{\pi} V^{\pi}(s) \qquad ,\forall s \in S \tag{2.11}$$

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \qquad ,\forall s \in S \textit{ and } \forall a \in A(s) \tag{2.12}$$

$$Q^*(s,a) = E\{r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a\} \tag{2.13}$$

$$
\begin{aligned}
V^*(s) &= \max_{a \in A(s)} Q^{\pi^*}(s,a) \\
&= \max_a E_{\pi^*}\{R_t|s_t = s, a_t = a\} \\
&= \max_a E_{\pi^*}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s_t = s, a_t = a\right\} \\
&= \max_a E_{\pi^*}\left\{r_{t+1} + \sum_{k=0}^{\infty}\gamma^k r_{t+k+2}|s_t = s, a_t = a\right\} \\
&= \max_a E\{r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a\} \\
&= \max_a \sum_{s'} T^a_{SS'}\left[R^a_{SS'} + \gamma V^*(s')\right]
\end{aligned}
\tag{2.14}
$$

Similarly, for Q values:

$$
\begin{aligned}
Q^*(s,a) =& E\{r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1},a') \mid s_t = s, a_t = a\} \\
=& \sum_{s'} T^a_{SS'}\left[R^a_{SS'} + \gamma \max_{a'} Q^*(s',a')\right]
\end{aligned}
\tag{2.15}
$$

2.1.2 Temporal Difference

The Temporal Difference (TD) family of algorithms [8] provides an elegant framework for solving prediction problems. The main advantage of this class of algorithms is its ability to learn directly from raw experience, without any further information, such as the model of the environment (model free). The temporal difference is a *bootstrapping* technique, where its estimates are updated online based in part on the previously learned value function estimations. More specifically, at each time t , where the agent executes the action a_t at state s_t , the predicted state-value of the newly visited state s_{t+1} along with the immediate received reward are used, in order to estimate the prediction error, known as the temporal difference error. The value iteration is as following:

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right] \tag{2.16}$$

where α is the learning rate.

2.1.3 Exploration and Exploitation

One big subject on self-learning algorithm is how to achieve a balance between exploration and exploitation. While the exploration in this context being to visit all the states enough to learn how to make a decision. And exploration is when an agent uses the collected information to make an intelligent decision.

2.1.4 ε-greedy

One of the common strategies to select actions and explore the environment is ε-greedy. This is the simplest method to conduct an action selection process by selecting the action with the largest estimated accumulated reward value at each state, $a*$, for $Q_t(s, a^*) = \max_a Q_t(s, a)$. At the beginning where all the states have not been visited yet, the method explores the environment by selecting the action randomly with the probability ε and the probability to selects the largest accumulated reward is $1 - \varepsilon$ (exploration). The main benefit of this method is each state will be visited at least once. As the time progress the ε value decreases the method exploit the information collected by the agent and use it to choose the best action available (exploitation).

2.2 Q-learning Algorithm

Q-learning (Watkins, 1989) [9] is one of the most studied topics for reinforcement learning. It offers quick and easy Off-policy temporal different control.

It is a non-deterministic rewards and action method that is widely used in self-learning agent. Q-learning is another extension to traditional dynamic programming (value iteration) that is great at tackling control problem. It does not require the model of the environment. Q-learning belongs to the family of temporal control method, and capable of being implemented in on-line, fully incremental way without the need of waiting until the end of an episode. In addition to that, Q-learning algorithm learns from raw experience without a model of the environment's dynamics (model-free). The policy update rule of Q-learning uses state-action pair as following:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q^{\pi}(s',a') - Q(s,a) \right] \tag{2.17}$$

where $Q(s,a)$ is the state-action pair, α is the learning rate, r is the reward and γ is the discount factor.

The key point is to find optimal Q-value of the each state-action pair (s_t, a_t) using stochastic exploration and the value iteration based on temporal difference. In this thesis we adope the Q-learning algorithm with ε-greedy [10] 2.1 for exploration probability.

As a model-free control method, Q-learning is based on the estimation of the action value function, Q. Learning a policy therefore means updating the Q-function to make it more accurate. One important aspect of model-free algorithms is that there is a need for exploration. To account for potential inaccuracies in the Q -function, the agent must try out different actions to explore the environment for finding possible better policies. The ε-greedy action selection strategy is an effective means of balancing exploration and exploitation in reinforcement learning, which selects actions according to:

$$a = \begin{cases} a \in argmax_a Q(s,a) & \text{with probability} \quad 1-\varepsilon_t \text{ (exploit)} \\ \text{random action in } A & \text{with probability} \quad \varepsilon_t \text{ (explore)} \end{cases} \tag{2.18}$$

where $\varepsilon_t \in (0,1)$ is the exploration probability at time step t.

Usually, the exploration diminishes over time, so that the policy used asymptotically becomes greedy and therefore (as $Q_t \to Q^*$) optimal. This can be achieved by making ε_t approach 0 as t grows. For instance, an ε-greedy exploration schedule of

Algorithm 2.1 Q-learning algorithm with ε-greedy exploration

```
1: Exploration schedule $\{\varepsilon_t\}_{t=0}^{\infty}$
2: Initialize Q (s, a) arbitrarily
3: for each episode do
4:     Initialize s
5:     repeat
6:         Choose action $a$
7:         $a = \begin{cases} a \in argmax_a Q(s,a) & \text{with probability} \quad 1-\varepsilon_t \text{ (exploit)} \\ \text{random action in } A & \text{with probability} \quad \varepsilon_t \text{ (explore)} \end{cases}$
8:         Take action $a$, observe $r$ and $s'$
9:         Update $Q(s,a)$:
10:        $Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q^{\pi}(s',a') - Q(s,a))$
11:    until s is terminal state
12: end for
```

the form $\varepsilon_t = 1/t$ diminishes to 0 as $t \to \infty$, while still satisfying the second convergence condition of Q-learning, i.e., while allowing infinitely many visits to all the state-action pairs. According to that, the agent behaves greedily most of the time, but with small probability, ε, selects an action uniformly random.

2.3 Value Function Approximation

The majority of reinforcement learning algorithms rely on the estimation of a value function, which is a real-valued function over the state or the state-action space. In finite state spaces, value functions can be represented exactly using a tabular form that directly stores in memory a separate value for each individual state. Nevertheless, in the case where the state space is large or infinite (commonly encountered in the real world) an exact value representation becomes prohibitive. This problem not only stems from memory constraints, but also from the time as well as the samples needed for accurately learning all table entries. Therefore, an approximation architecture for the representation of the value function is commonly adopted, which must facilitate generalization. The most typical approximation scheme is the linear function approximation, where the value function is represented as the weighted combination of a set of basis functions:

$$V(s) = \phi(s)^T w = \sum_{i=1}^{k} \phi_i(s) w_i \tag{2.19}$$

where $w \in \mathbb{R}^k$ is a vector of coefficients and $\phi \colon S \to \mathbb{R}^k$ is a mapping from states to a k-dimensional vector, called basis function. Usually, basis functions are fixed and non-linear functions of s. In this way, the value function $V(s)$, is allowed to be non-linear function of the state space. Functions of the form of 2.19 are called linear models, as they are linear in the parameters w. Nevertheless, poor design choices can result in estimates that diverge from the optimal value function and agents that perform poorly. In practice, achieving high performance requires finding an appropriate representation for the value function approximator. Next, we consider a number of approaches suitable for state space representation, which have been used extensively in the area of reinforcement learning.

State Aggregation

State aggregation is the simplest method for defining features for a linear function approximator. This approach discretizes the continuous state space into disjoint segments, whose union covers the state space S. In this way, a binary feature is attached to each region, which can be seen as its indicator. A feature is active (i.e. equal to 1) if the considered state falls into the corresponding region. Otherwise, the feature is 0 and is supposed as inactive.

Chapter 3

Multiagent Reinforcement Learning for Multiple Mini-Robots

In this chapter we will explain the use of reinforcement learning for navigation of unknown space using Q-learning algorithm in this thesis. This will provide framework to build intelligent agents that learn to interact with unknown and unpredictable environment, in order to create robust navigation policies for the desired single robot and multiple robots platform. We use robot operating system (ROS) to implement the centrifugal-force mini robot simulator and Q-learning algorithm for navigation. As for results verification, we use both ROS and the exported data to process in Matlab. ROS also provide the visualization to help us understand movements and paths of the robots better. The visual representation also provides us the clue of how robots interact with an environment during the learning and exploration procedures.

3.1 A Multiagent RL Framework for Multiple Mini-Robots Navigation

Working in multiagent environment can be unpredictable due to the changes made to the environment by other agents. In this work we propose the use of reinforcement learning to construct an intelligent agent for navigation of collaborative medical mini-robotics platforms to be able to navigate to the goal regardless of the initial state and condition. Workspace is divided in to grids, each grid represents the status for each agent the goal for others are considered static obstacle and the other agents are considered dynamic obstacles as seen in figure 3.1. In this thesis we focus on proximity collision avoidance policies that map the distance between other agent in to state-action pairs using centralized control for all agents to perceive the position of other agents near by and subsequently calculate the rewards. Workspace is on a rectangular flat surface. The agents have no prior knowledge of the environment and starting position for each episode are different and to the edge of the workspace. For each episode to end, either an agent goes out of boarder or collide with other obstacles or all the agents reach their goals. The goals for each agent are in different position and occupy one block of grid.

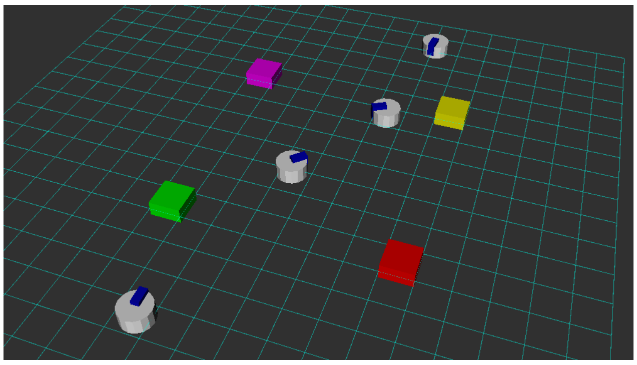


Figure 3.1: Multi-robot navigation concept

3.1.1 Q-Learning Algorithm for Autonomous Robots Navigation

In this section, we present the use of Q-learning algorithm for navigation of multi-robots system. We have adopted the Q-learning algorithm to discover a proper policy, π, based on which our agents select the most appropriate action at each state.

The centrifugal-force mini robot requires very fine integral at 10^{-6} second per step for speed and position calculation due to high frequency of the eccentric mass rotation. Each time step is $50000dt$, which translates to 0.5 second so the robot can have small of movement whether starting from stand still or continue running from previous time step.

In this thesis we examine the use of different exploration probability update rates for one robot environment and add the different reward functions for multiple robots. The Q-values table created from different learning parameters then being compared to observe effects of the aforementioned parameters. The learning rate and discount factor are kept constant throughout all the experiments. The focus is mainly on the exploration.

One of the most crucial part of constructing an agent is balancing between exploration and exploitation. We purpose the update strategy by multiplying the ε by a constant for a certain number of episodes. This way we can choose the appropriate exploration strategy according to the problem size, which we will discuss in the following sections.

3.1.2 Single Robot Navigation

First let us look at the single robot navigation system using Q-Learning in order to understand how the agent works in a simplified environment. The workspace is divided in to grid of 20×20, or 40×40 cells and the states are the $S = (x, y)$ coordinate of the robot. The actions consist of 8 possible directions (in step of $45°$) as depicted in figure 3.2 and the goal is located at the centre of the workspace. There is no obstacle in the workspace so the agent is free to explore through the entire board without any obstruction. The rewards r are defined as following:

$$r = \begin{cases} 100 & , \text{ reaches goal} \\ -100 & , \text{ out of border} \\ -1 & , \text{ otherwise} \end{cases} \tag{3.1}$$

The learning procedure for each episode starts at a random point on the edge of the board. Then the agent is trying to explore the environment to collect data for Q values. Over time the Q value table is being filled and the agent gradually use the gathered data for exploitation rather than exploration. It is crucial to strike a balance between exploration and exploitation so that the agent will construct an optimal policy for navigation. We use the term "optimal" to describe shortest path and the least direction change possible from any starting point. The reason we also use the

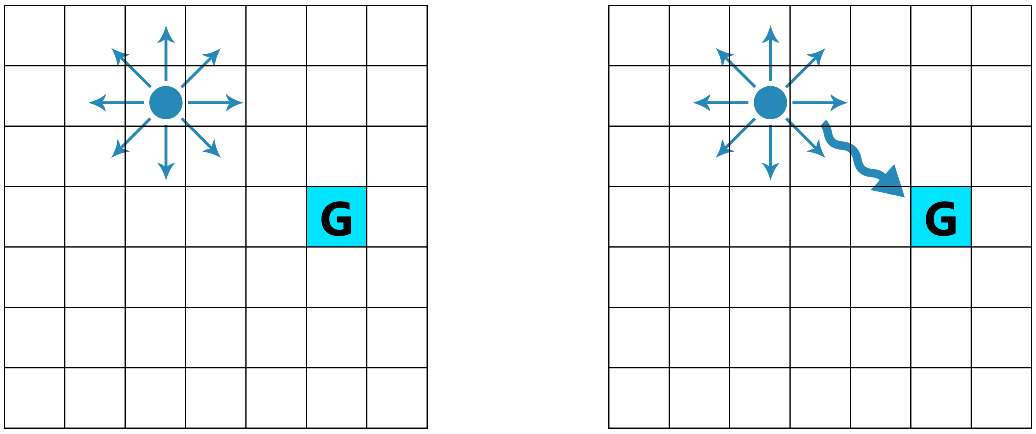


Figure 3.2: Single robot navigation

least number of direction changes is because of the limitation of the robot model we use, this robot requires substantial time to change the direction.

3.1.3 Multiple Robots Navigation

The multiple robots navigation system utilizes the basics from single robot navigation environment with added features such as the discrete distance between robots and considers the goals for other robots as obstacle. An agent only knows distance information about other robots. All robots are sharing the same workspace so they have to avoid collision between one another. Considering each robot has to be in the same board and has to avoid both goals of other robots and other robots as well (figure 3.3), the agents have to be able to navigate through limited space if two or more robots are in close proximity. Each robot has its own goal and not competing to reach the same goal. The goals are located in the centre region of the board with some space between each goal for the robots to be able to navigate between them without collision. This is being done to share the workloads. Imagining the scenario of different medicines have to be injected in different positions. This can be done quickly with more robot rather using only one robot. We will examine the different reward functions to determine the quality of navigation with the percentage of success episodes. Starting positions are different and the failure of each episode is occur when an agent either going out of border, or collide with **obstacles**, defined by **other robots** *(dynamic)* and **other robots' targets** *(static)*.

The workspace is constructed by dividing in to grid of 20×20 cells. The set of states for every robot (i) consists of: $S_i = (x_i, y_i, d_i)$

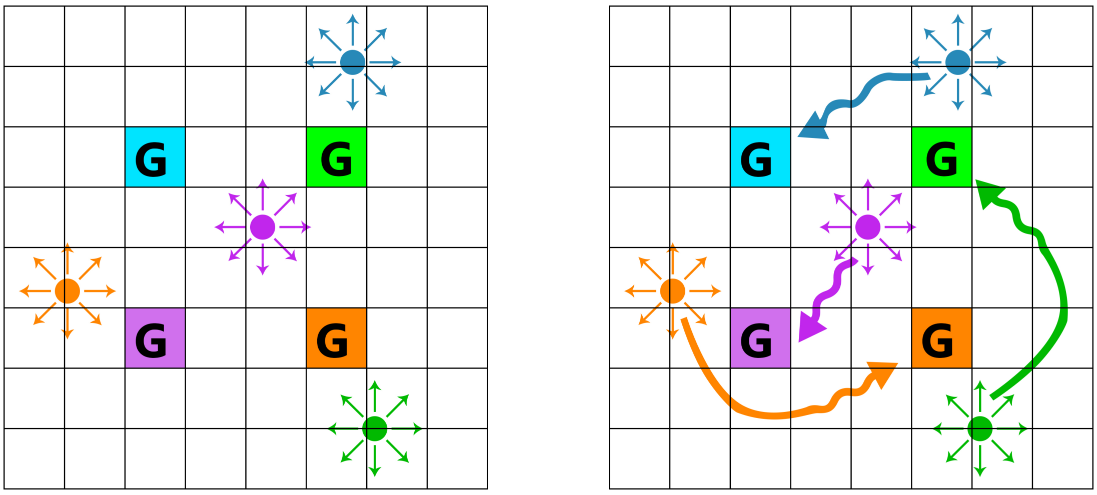


Figure 3.3: Multiple robots navigation

- x_i, y_i, position of robot on board
- d_i, distance of other robots: **mean** or **minimum** *(smallest)* distance

And we examine the set of actions between 4 and 8 directions to see which provides the best results. As we have seen in the state-action pair, the multiple robot has one additional attribute which is the discrete distance between robots. This also adds more complexity and the number of states-action pairs so we need to examine which combination of the state-action produce the best result and for the exploration to be able to visit all the states, often enough to gather sufficient information to use for navigation.

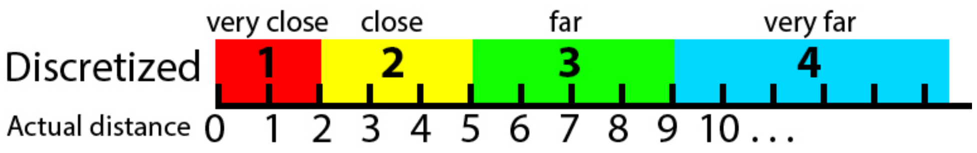


Figure 3.4: Discretization of distance state.

The discretization of the distance is as shown in figure 3.4. We build the distance state by quantizing the distances in to states from very close to very far. The severity of situation is quantized in to discrete distance states starting from 1, meaning there is a robot or are other robots in very close proximity to the selected robot, hence the robot is in imminent danger of collision. The following states mean the closest robots or other robots are further away and pose less threat of collision. The discrete distance state can be adjusted to have appropriate size for the workspace and complexity of the problem. We purpose **two reward functions for multiple robot**, one is by taking the **position of all other robots** in to consideration (see fig. 3.5) by using textitmean discrete distance state to map the position and distance state to the actions. The latter is to use only the **position of the nearest robot** in to consideration (see fig.3.6) and convert the *smallest distance* to discrete distance state.

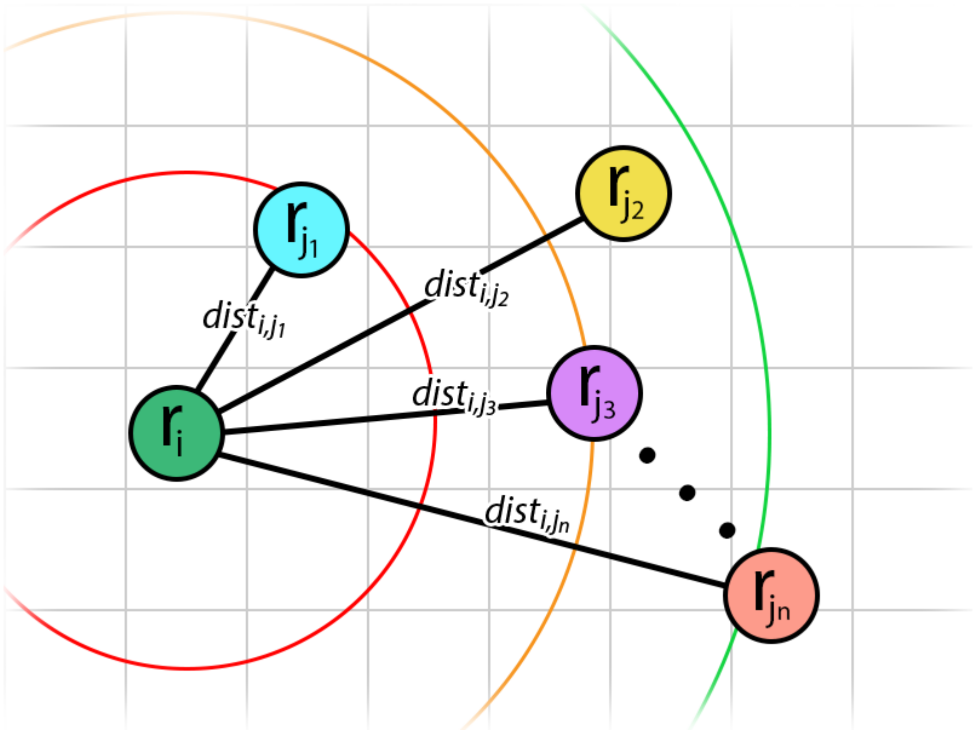


Figure 3.5: Mean distance state

The reward functions are as following:

$$r_i = \begin{cases} L & , \text{ reaches goal} \\ -L & , \text{ out of workspace, collision} \\ cr_i & , \text{ otherwise} \end{cases} \tag{3.2}$$

Calculation of cr_i (**collaborative reward**):

- **mean** distance: $cr_i = -k\frac{\sum_j(1-\frac{dist_{i,j}}{D})}{(N_R-1)} - c$
- **smallest** distance: $cr_i = -k\left[\min_{d_{i,j}}(1-\frac{dist_{i,j}}{D})\right] - c$

where

- $dist_{ij}$ discrete distance between two robots
- N_R number of robots in workspace

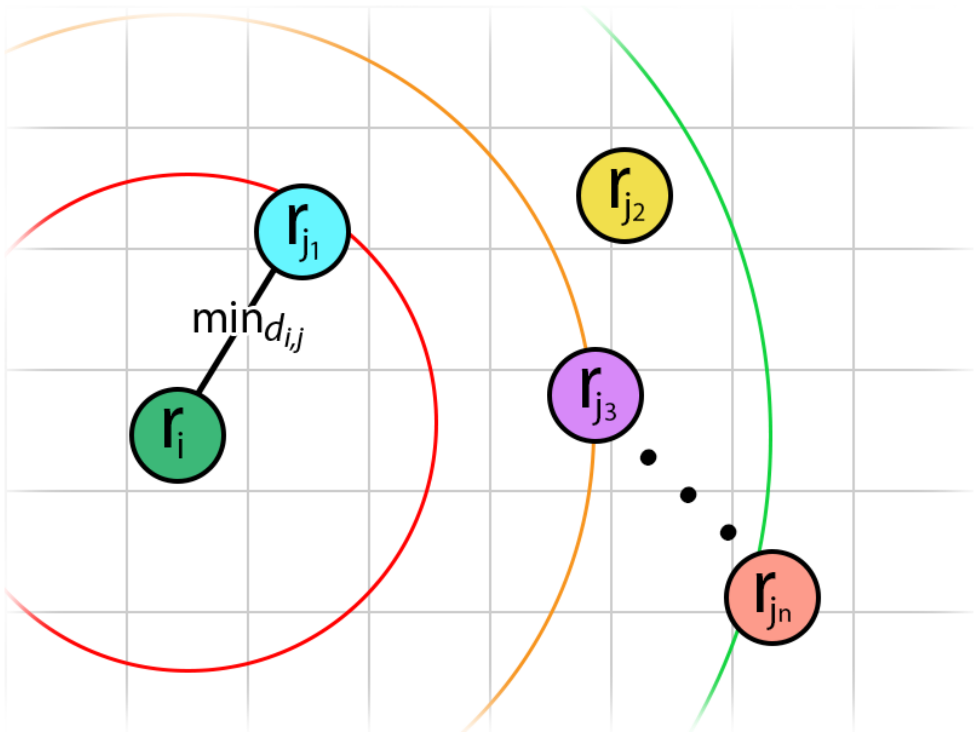


Figure 3.6: Smallest distance state

- D maximum discrete size of distance
- L, k, c constants (common values $L > 2000$, $k = 1$, $c = 1$)

In this thesis we examine different state-action numbers to find the optimal distance states from the quantization of actual distance in to 3 and 4 states with the number of available actions between 4 and 8 actions.

3.2 Implementation Details

3.2.1 Robot Operating System (ROS)

So far we have only mentioned the use of Robot Operating System to implement the reinforcement learning in robot but we have not explained it yet. In this section we will explain in details of what it is, and its importance in development of robotic platforms.

Robot Operating System (ROS) [11] is a framework that is widely used in robotics. The philosophy is to make a piece of software that could work in other robots by making little changes in the code. What we get with this idea is to create functionalities that can be shared and used in other robots without much effort so that we do not reinvent the wheel.

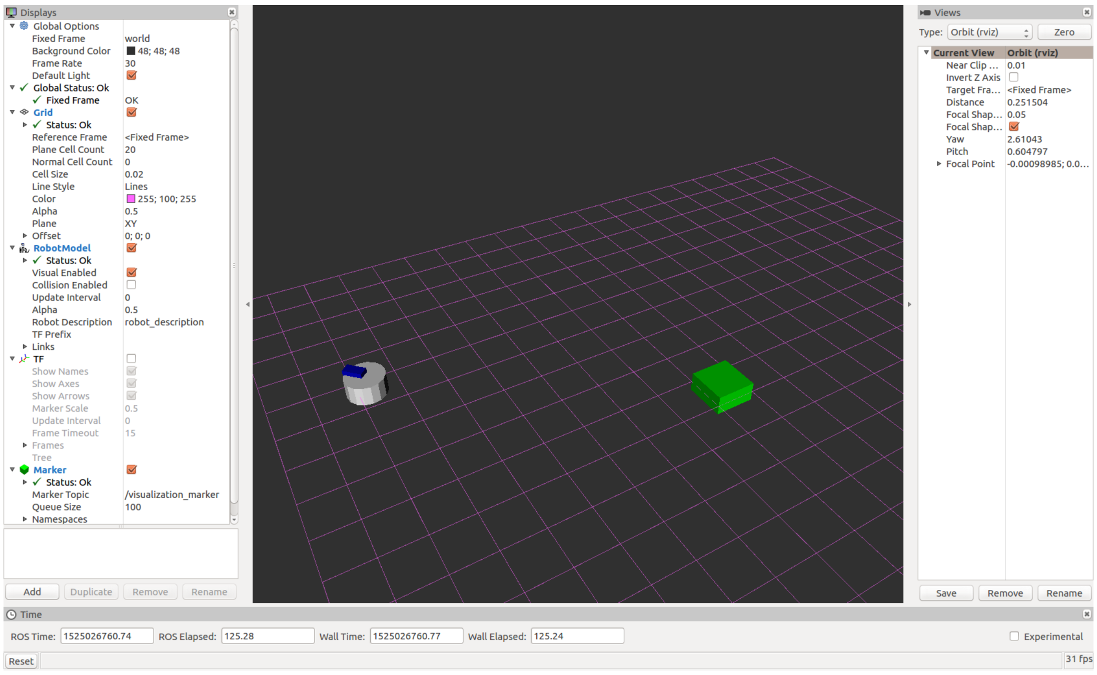
ROS was originally developed in 2007 by the Stanford Artificial Intelligence Laboratory (SAIL) with the support of the Stanford AI Robot project. As of 2008, development continues primarily at Willow Garage, a robotics research institute, with more than 20 institutions collaborating within a federated development model.

A lot of research institutions have started to develop projects in ROS by adding hardware and sharing their code samples. Also, the companies have started to adapt their products to be used in ROS. In the figure 3.7, you can see some fully supported platforms. Normally, these platforms are published with a lot of code, examples, and simulators to permit the developers to start their work easily. The sensors and actuators used in robotics have also been adapted to be used with ROS. Every day an increasing number of devices are supported by this framework. ROS provides

Figure 3.7: Robots supported by ROS

standard operating system facilities such as hardware abstraction, low-level device control, implementation of commonly used functionalities, message passing between processes, and package management. It is based on graph architecture with a centralized topology where processing takes place in nodes that may receive or post, such as multiplex sensor, control, state, planning, actuator, and so on. The library is geared towards a Unix-like system (Ubuntu Linux is listed as supported while other variants such as Fedora and Mac OS X are considered experimental).

The $* - ros - pkg$ package is a community repository for developing high-level libraries easily. Many of the capabilities frequently associated with ROS, such as the navigation library, the Rviz visualizer (figure 3.8) and the Gazebo simulator (figure 3.9), are developed in this repository. These libraries give a powerful set of tools to work with ROS easily, knowing what is happening every time. Of these, visualization, simulators, and debugging tools are the most important ones. There are plenty of packages and tools to develop and build user-written algorithm and applying it to



Global Options	
Fixed Frame	world
Background Color	■ 48; 48; 48
Frame Rate	30
Default Light	✓
✓ Global Status: Ok	
✓ Fixed Frame	OK
Grid	✓
✓ Status: Ok	
Reference Frame	<Fixed Frame>
Plane Cell Count	20
Normal Cell Count	0
Cell Size	0.02
Line Style	Lines
Color	■ 255; 100; 255
Alpha	0.5
Plane	XY
Offset	0; 0; 0
RobotModel	✓
✓ Status: Ok	
Visual Enabled	✓
Collision Enabled	☐
Update Interval	0
Alpha	0.5
Robot Description	robot_description
TF Prefix	
Links	
TF	☐
Show Names	✓
Show Axes	✓
Show Arrows	✓
Marker Scale	0.5
Update Interval	0
Frame Timeout	15
Frames	
Tree	
Marker	✓
✓ Status: Ok	
Marker Topic	/visualization_marker
Queue Size	100
Namespaces	

Type: Orbit (rviz)

Current View	Orbit (rviz)
Near Clip …	0.01
Invert Z Axis	☐
Target Fra…	<Fixed Frame>
Distance	0.251504
Focal Shap…	0.05
Focal Shap…	✓
Yaw	2.61043
Pitch	0.604797
Focal Point	-0.00098985; 0.0…

ROS Time: 1525026760.74 ROS Elapsed: 125.28 Wall Time: 1525026760.77 Wall Elapsed: 125.24 ☐ Experimental

31 fps

Figure 3.8: Rviz visualizer

respective robot or simulator.

ROS is released under the terms of the BSD (Berkeley Software Distribution) license and is an open source software. It is free for commercial and research use. The $* - ros - pkg$ contributed packages are licensed under a variety of open source licenses.

ROS promotes code reutilization so that the robotics developers and scientists do not have to reinvent the wheel all the time. With ROS, you can do this and more. You can take the code from the repositories, improve it, and share it again.

ROS has released some versions, as of now the latest one being Melodic. In this thesis, we are going to use Kinetic Kame because it is the most recent long-term support (LTS) release.

3.2.2 Implementation in ROS

In this work all the nodes are created new without using previously existed packages of the ROS. The centrifugal-force robot simulator complies with ROS standard so it works with standard ROS command to drive the robot (cmd_vel) so it can connect and use with other nodes easily. The simulator is also compatible with Rviz for visualization.

Q-Learning algorithm (algorithm 2.1) and controller nodes are talking to each

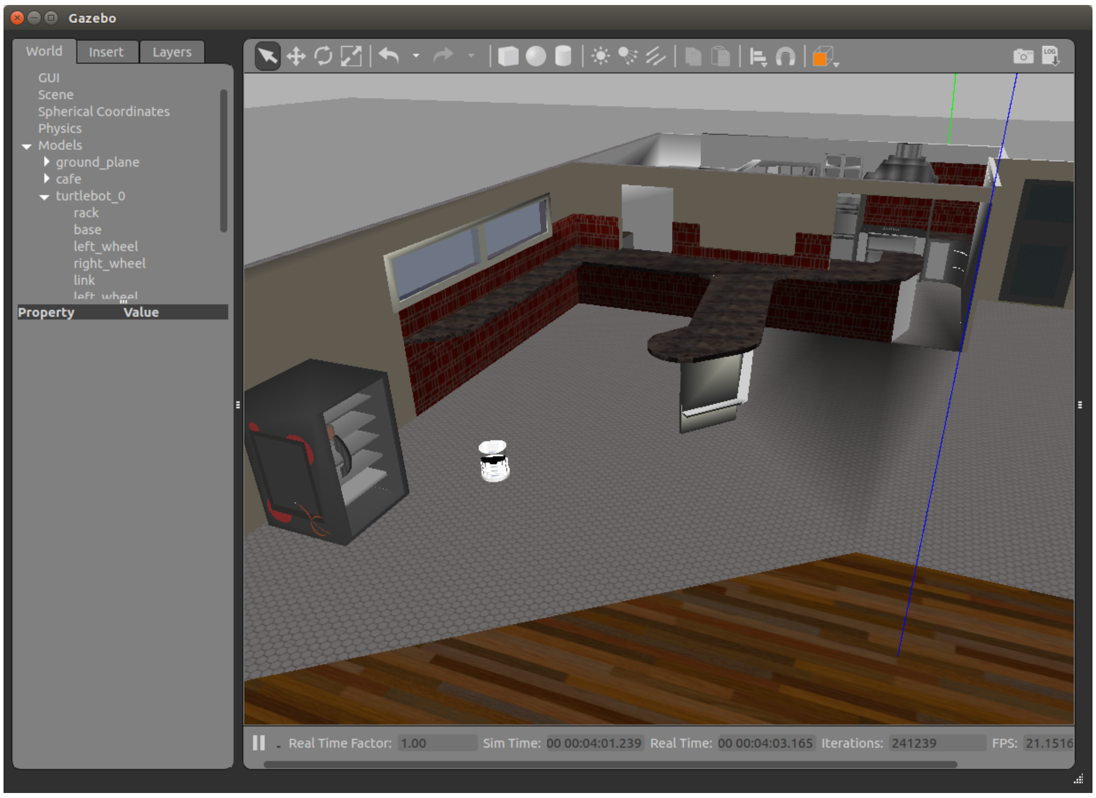


Figure 3.9: Gazebo simulator

other via ROS messages that has many to many capability. The connections between node can be seen in figure 3.10. The controller is talking to the robot via ROS service as one to one pair. Finally we use Rviz for visualization of the robot and workspace. The connection between node is depicted in figure 3.10.

As for the multi-robots Q-learning algorithm the main program is a centralized node that controls all the robots. The node recognize the positions of every robot. It constructs the Q value table and subsequently follow the policies that created for each robots.

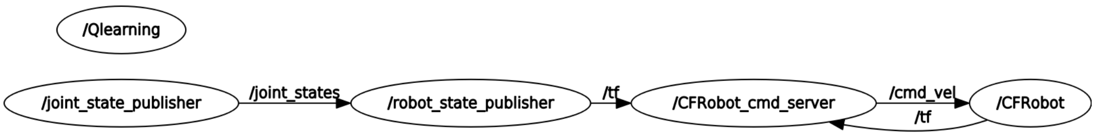
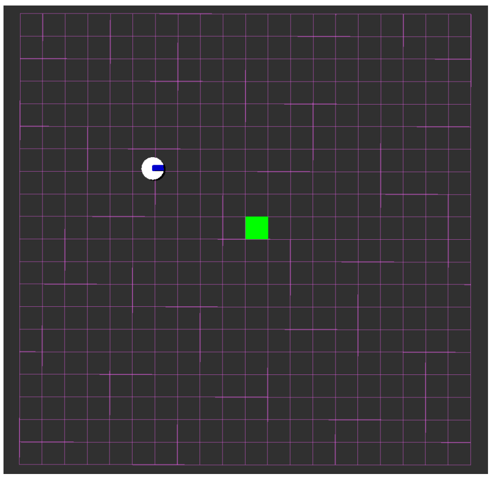


Figure 3.10: Navigation of centrifugal-force robot with Q-Learning nodes

The single robot environment visualization in Rviz is depicted in figure 3.11a completed with 3D model and heading of the robot. This environment also support other features such as publishing topic transform *(TF)* for robot positioning, odometry *(odom)* for position tracking and compatibility with other navigation stack and velocity command *(cmd_vel)* for driving the robot with standard ROS command. The environment for multiple robots can be seen in figure 3.11b. This environment only works as a visualization tool to verify the learning result with no capability to work with other stacks or packages.


(a) 1 Robot visualization in Rviz

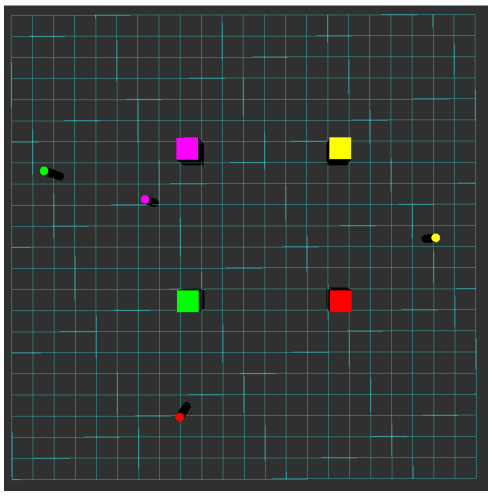

(b) 4 Robots visualization in Rviz

Figure 3.11: Centrifugal-force robot in ROS visualization

Chapter 4

Experimental Results

The experiments conducted in this thesis focused mainly on various ε exploration probability value for single robot environment. As for multi-robot environment different reward functions are being used for finding optimal policy. Furthermore we have experimented with different number of state-action pairs from different discrete distance between robots and simplified actions.

4.1 Experiments of Single Robot

First, we will examine the results from single robot environment to see how the agents react to different noise level and different exploration strategies. The experiments for single robot are as follow:

- Work space is divided in to $20{\times}20$ blocks of $5mm \times 5mm$ tiles, $10 \times 10cm$ in total or $40{\times}40$ blocks of $5mm \times 5mm$ tiles, $20 \times 20cm$ in total.
- The goal is located in the grid $(10, 10)$ for $20{\times}20$ blocks workspace and $(20, 20)$ for $40{\times}40$ blocks workspace.

- Initial exploration probability $\varepsilon = 0.9$
- Experiment with exploration rate update $\varepsilon = \varepsilon \times 0.999$.
- Learning rate $\alpha = 0.001$.
- Discount rate $\gamma = 0.99$.

4.1.1 Noisy Environment

One of the issues a centrifugal-force mini robot can face is the inaccuracy of the vibration motor speed as we can not set the constant angular velocity due to technical limitations, which leads to the different speed and subsequently position of the robot in each time step. We call this phenomenon "noisy environment". To simulate the noise in the environment we add white noise to the speed of a robot after acceleration is integrated. The conducted experiments are for different noise level at 1%, 5%, 10% and 20%.

The environment and Q-learning algorithm parameters are as following:

- Initial exploration probability $\varepsilon = 0.9$
- Experiment with exploration rate update $\varepsilon = \varepsilon \times 0.999$ every 100 episodes.
- Noise signal $1\%, 5\%, 10\%, 20\%$

The results are depicted in figure 4.1 for success rate and figure 4.2 for path length. We concluded that the noise level has no real impact to the performance of the learning and agent construction. The reason being all the states are separated in to grid system, and each movement of the robot at time t to $t+1$ does not immediately move to the next state, giving no negative impact to the agent even when it moves further for maximum of 20% travelled distance.

4.1.2 Experiments 20x20 Grid

This is the standard workspace of 10×10 centimetres. The state-action pairs from this experiment is the smallest, hence the problem has less complexity and should provide the basic understanding of the agent construction. The parameters are as following:

- Work space is divided in to 20×20 blocks.

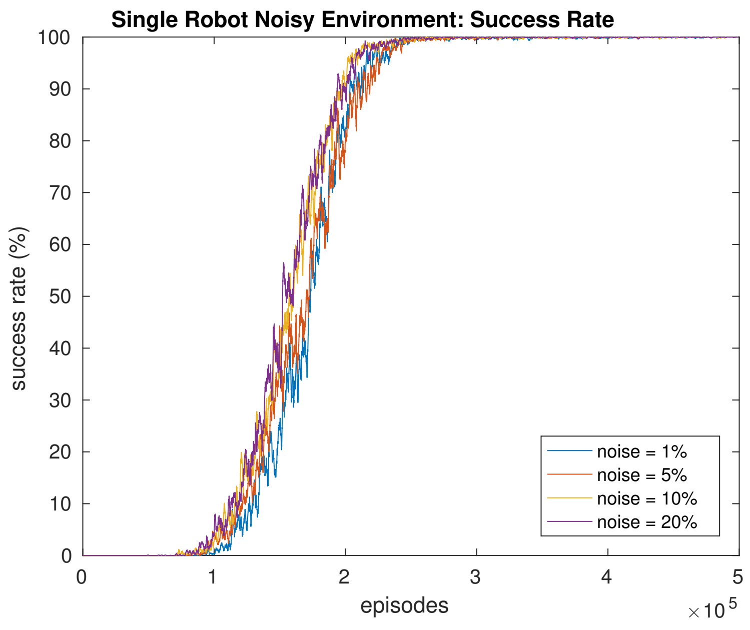


Figure 4.1: Noisy environment success rate

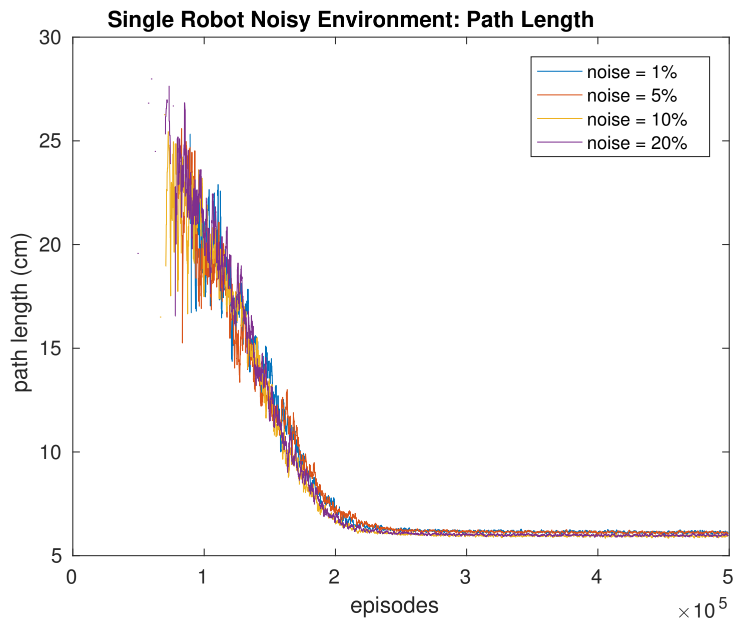


Figure 4.2: Noisy environment path lengths

- Initial exploration probability $\varepsilon = 0.9$
- Experiment with exploration rate update $\varepsilon = \varepsilon \times 0.999$ for each $1, 100, 500$ episodes.
- Learning rate $\alpha = 0.001$.
- Discount rate $\gamma = 0.99$.
- Noise signal $[0\%, 10\%]$.

The results are depicted in figure 4.3 for success rate and figure 4.4 for path length. The learning rate of single robot environment in 20×20 workspace converges very fast, at less than 500,000 episodes. However more exploration does not necessary mean the best result in this case. We can see from the exploration probability update rate at each 500 episodes, the produced policy dropped the success rate from 100% to 99.9923%. Validation is done from 3 runs of 1,000,000 episodes for each policy. The results of path lengths and success rate for each policy are depicted in table 4.1.

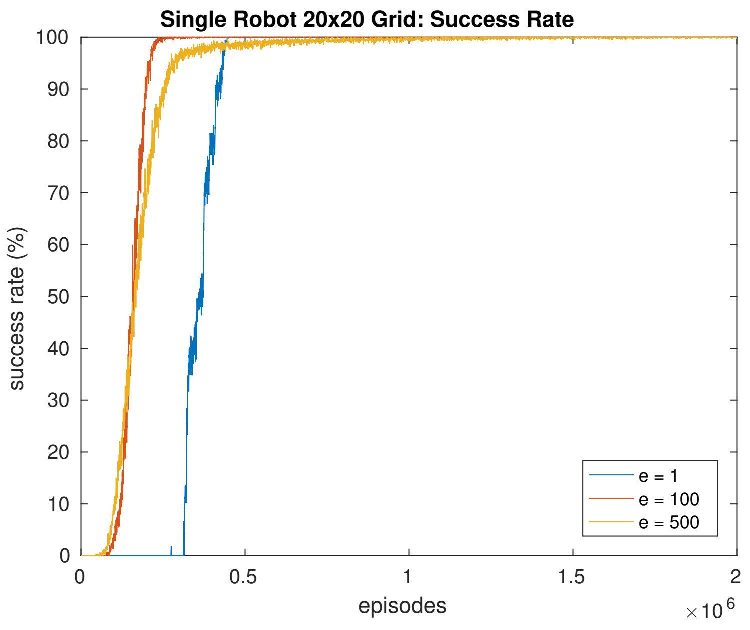


Figure 4.3: Single robot 20x20 grid success rate

We can see that the difference between path lengths from the policies with exploration probability update each 100 and 500 episode is negligible. But the success rate from the update each 500 episodes has dropped from policy with more exploitation.

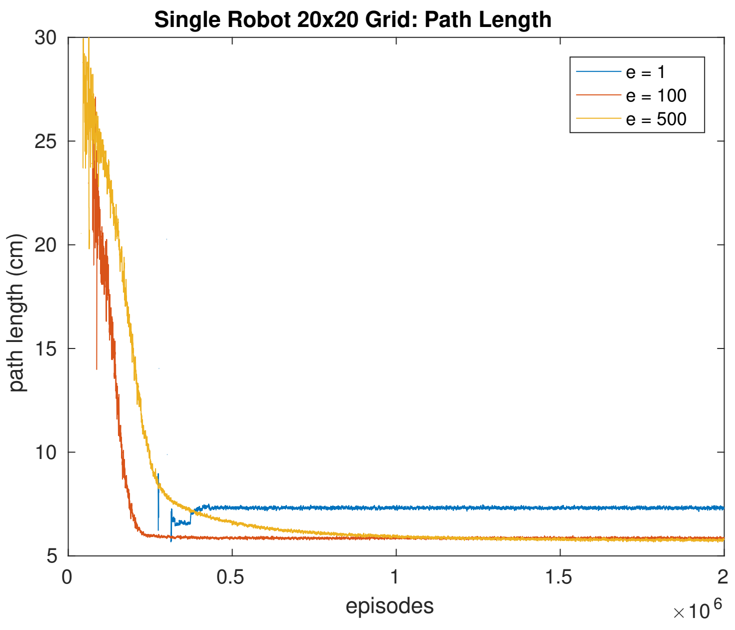


Figure 4.4: Single robot 20x20 grid path length

ε **Update episodes**	**Success rate (%)**	**Path length (cm)**
1	100%	7.31335
100	100%	5.85753
500	99.992%	5.73847

Table 4.1: Single robot 20x20 grid summary

4.1.3 Experiments: 40x40 Grid

These experiments are conducted to investigate how an agent would work in a larger environment and subsequently much higher state-action pairs and how to select good exploration strategy for more state-action pairs.

- Work space is divided in to 40×40 blocks.
- Goal is located in the grid $(20, 20)$
- Initial exploration probability $\varepsilon = 0.9$
- Experiment with exploration rate update $\varepsilon = \varepsilon \times 0.999$ for each $1, 100, 200, 500$ episodes.
- Learning rate $\alpha = 0.001$.
- Discount rate $\gamma = 0.99$.
- Noise signal $[0\%, 10\%]$.

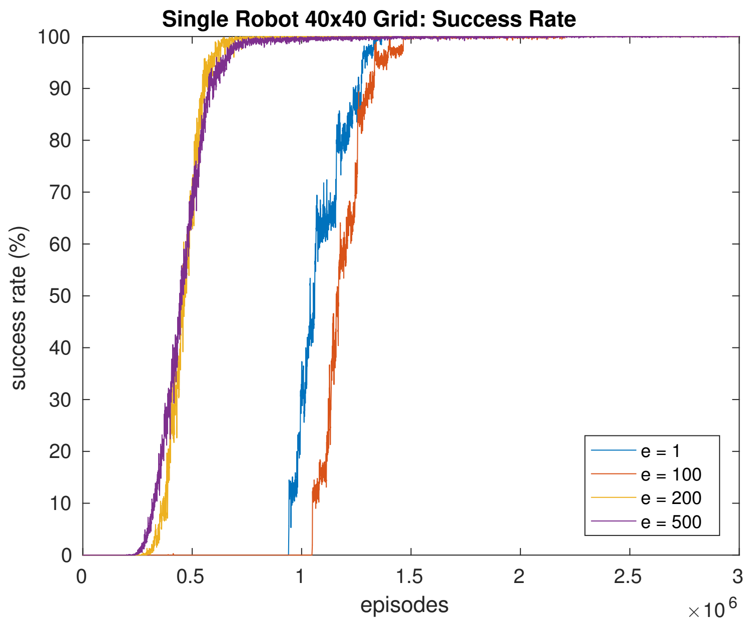


Figure 4.5: Single robot 40x40 grid success rate

The results are depicted in figure 4.5 and figure 4.6. The learning rate of single robot environment in 40×40 blocks workspace has increased from 20×20 blocks workspace about 3 times. The convergence happened at less than 1,500,000 episodes.

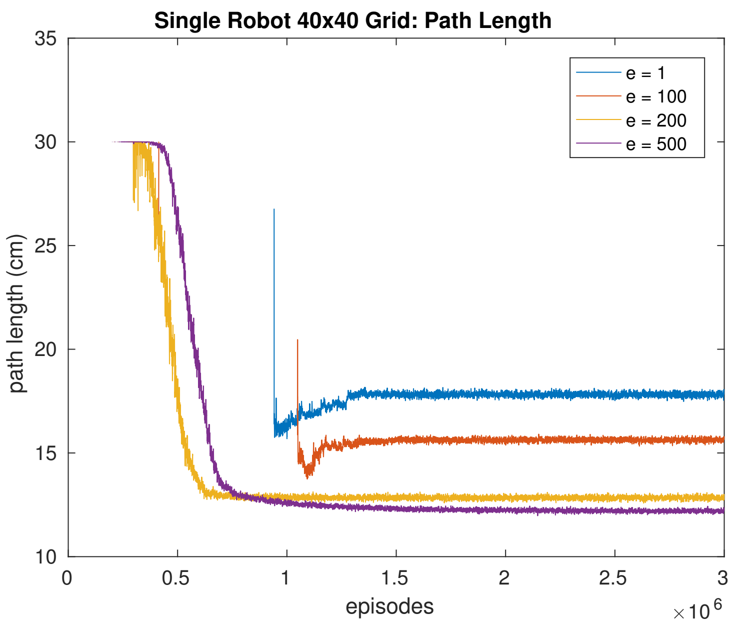


Figure 4.6: Single robot 40x40 grid path length

ε **Update episodes**	**Success rate** (%)	**Path length (cm)**
1	100%	17.821
100	100%	15.625
200	100%	12.842
500	99.99%	12.191

Table 4.2: Single robot 40x40 grid summary

The best exploration strategy to update the probability each 200 episodes and it converges at around 600,000 episodes. Once again we can see that more exploration cause more harm than good to the agent. When the path length is close to optimal, the difference of path lengths between exploration strategy are so slim and almost negligible. As conclude that a few millimetres does not worth the drop of success rate below 100%. Although 99.997% success rate is very satisfying but getting a robot to the target all the time is the main priority and it is not worth the compromise for a few millimetres of shorter path.

4.2 Experiments of Multiple Robots

The main focus of these experiments is to find which combination of the distance state-action pair combining with exploration strategy produces the best result for multiple robots environment. The agents not only need to reach their targets but need to avoid the collision with other robots as well.

The standard working parameters are as following:

- Workspace: grid of 20×20 cells
- The locations of the goals are in different grid for each robot:
  - Robot 1 goal: $(7, 7)$
  - Robot 2 goal: $(14, 7)$
  - Robot 3 goal: $(7, 14)$
  - Robot 4 goal: $(14, 14)$
- State for every robot (i): $S_i = (x_i, y_i, d_i)$
  - x_i, y_i, position of robot on board
  - d_i, distance of other robots: **mean** or **smallest** distance
- Action: 4 or 8 possible directions

First we examine the performance of distance states between mean distance state and minimum distance state starting by using only 4 actions to decrease the number of state-action pairs.

- Distance states: 4 discrete distance states
  - $[0, 2] = 1$
  - $(2, 5] = 2$
  - $(5, 9] = 3$
  - $(9, \infty) = 4$
- Action: 4 possible directions, in step of 90°

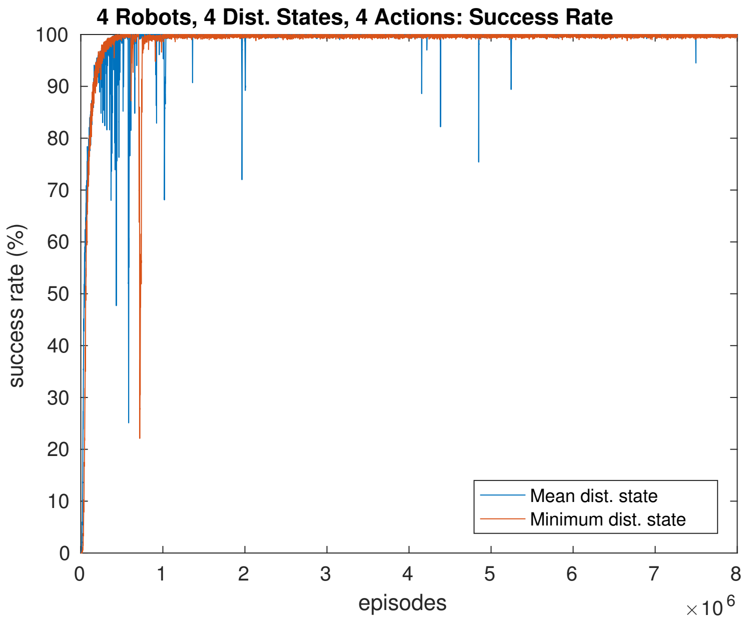


Figure 4.7: 4 Robots success rate, 4 actions
mean distance state VS min distance state

As we can see in the figure 4.7, the success rate for mean distance state fluctuates a lot even after the convergence. However the path lengths are the same for both mean distance state and minimum distance state as depicted in 4.8.

When different exploration strategies are applied. The results are in the same fashion as we can see in figure 4.9 that mean distance state produces many fluctuation in success rate. In figure 4.10, we can observe that mean distance state and minimum distance state both produce the same learning rate and path length, indicating the same overall performance is the same between these two state-action strategies. So we pick the minimum distance state to conduct further experiments.

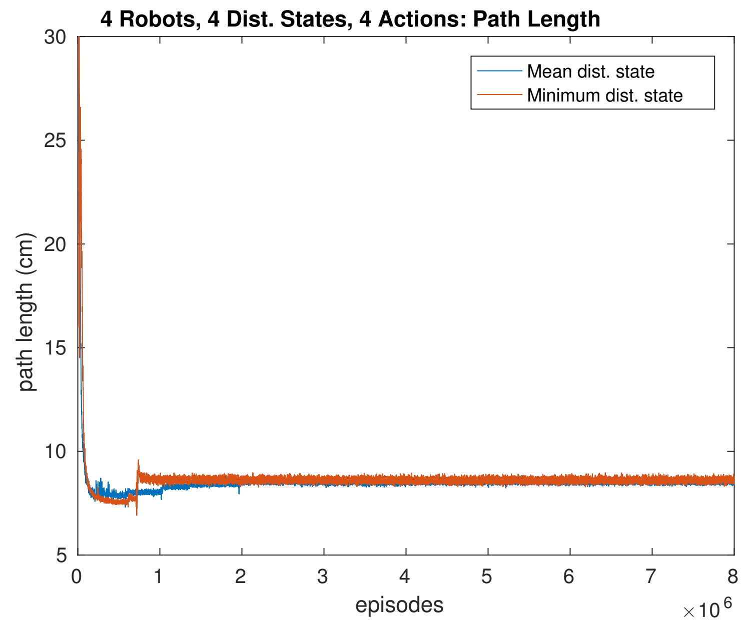


Figure 4.8: 4 Robots path length, 4 actions
mean distance state VS min distance state

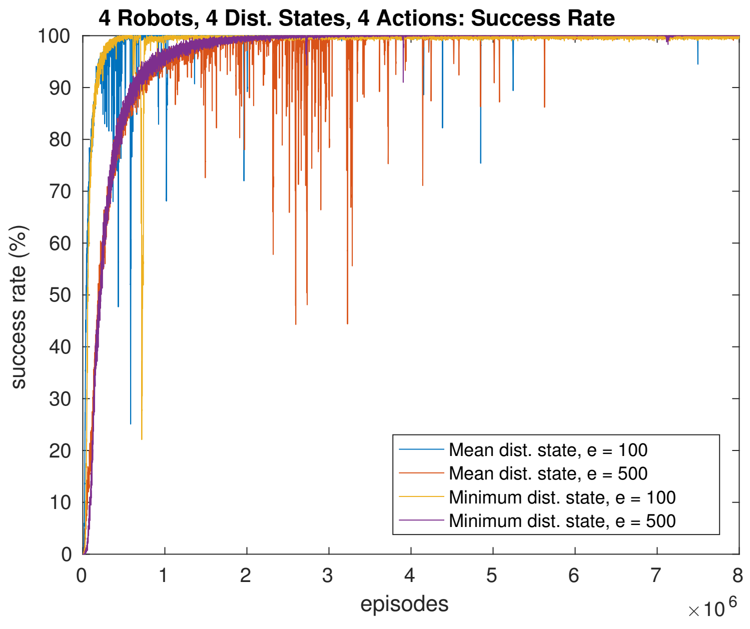


Figure 4.9: 4 Robots success rate, 4 actions
mean distance state VS min distance state with exploration updates

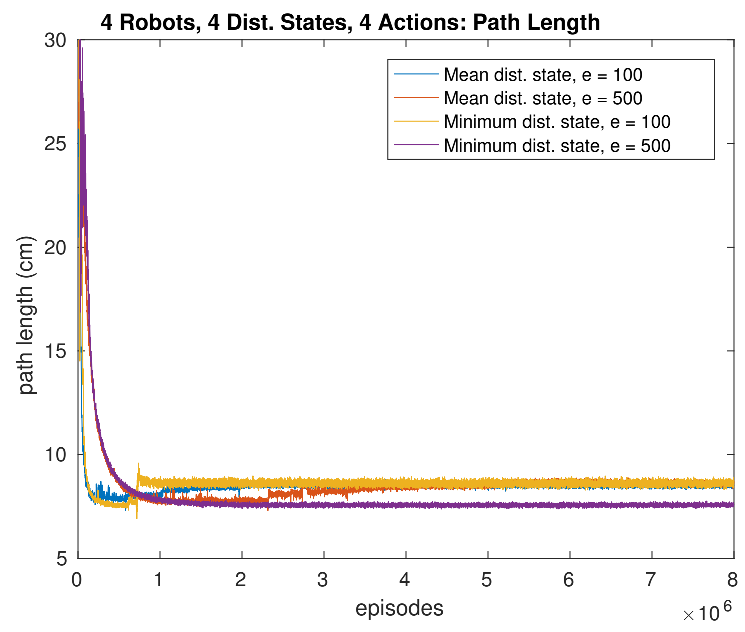


Figure 4.10: 4 Robots path length, 4 actions
mean distance state VS min distance state with exploration updates

Next we examine the performance between 4 action and 8 action states. As we add more actions in to agents, the complexity increases and the states need visiting increase subsequently.

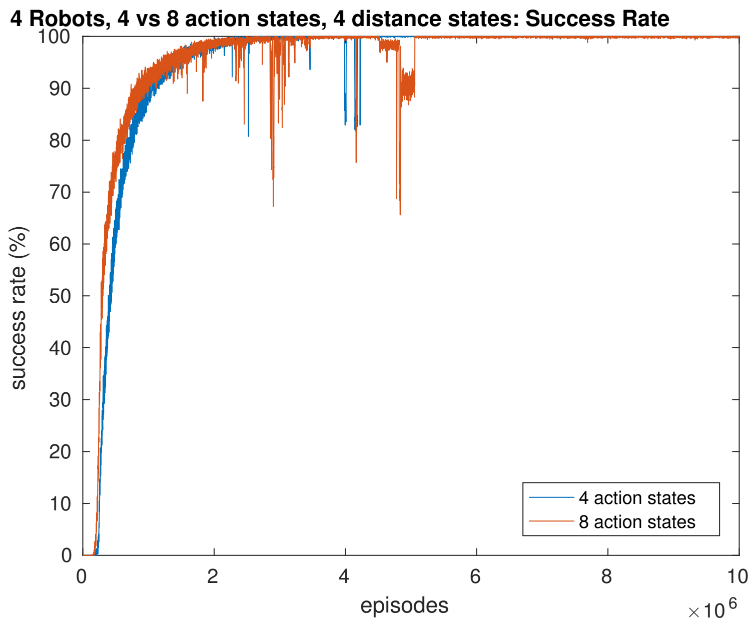


Figure 4.11: 4 Robots, 4 distance states success rate
4 actions VS 8 actions

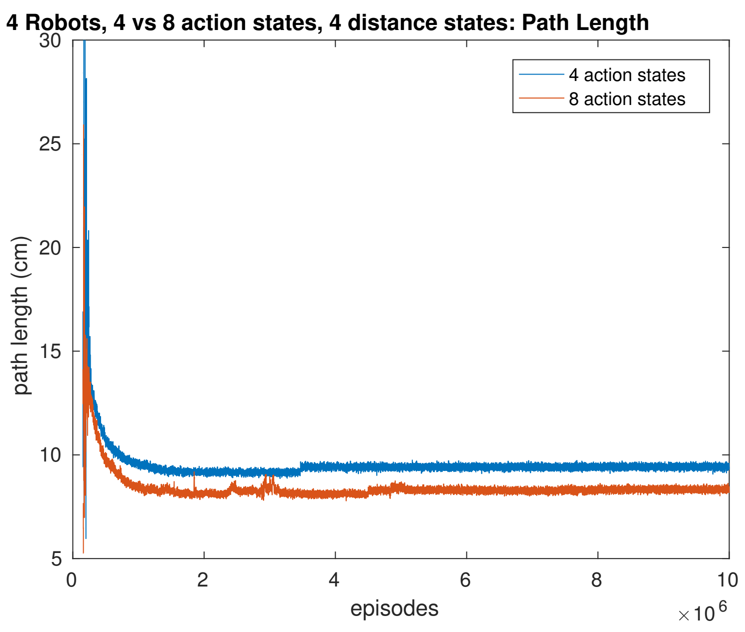


Figure 4.12: 4 Robots, 4 distance states path length
4 actions VS 8 actions

As we can see the results in figure 4.11, the success rate for 8 action states fluctuates a lot more than 4 action states. However the path length (figure 4.12 is shown that 8 action states produce better result for distance travelled. Because there are more directions to move, a robot can navigate through the workspace much easier and can evade other robots better.

Now that we know the 8 actions are good, we will try to lower the complexity of the agent by decreasing number of distance states to 3 as following.

- $[0, 3] = 1$
- $(3, 9] = 2$
- $(9, \infty) = 3$

The results for success rate from 3 distance states (figure 4.13) yield less fluctuation for both 4 actions and 8 actions. As for the path length (figure 4.14) action states, 8 actions yield better results than 4 actions for the aforementioned reason of better movements.

Comparing all the state-action pairs and exploration strategy we can have better understanding of how to construct an optimal policy for a collaborative multiple robots system.

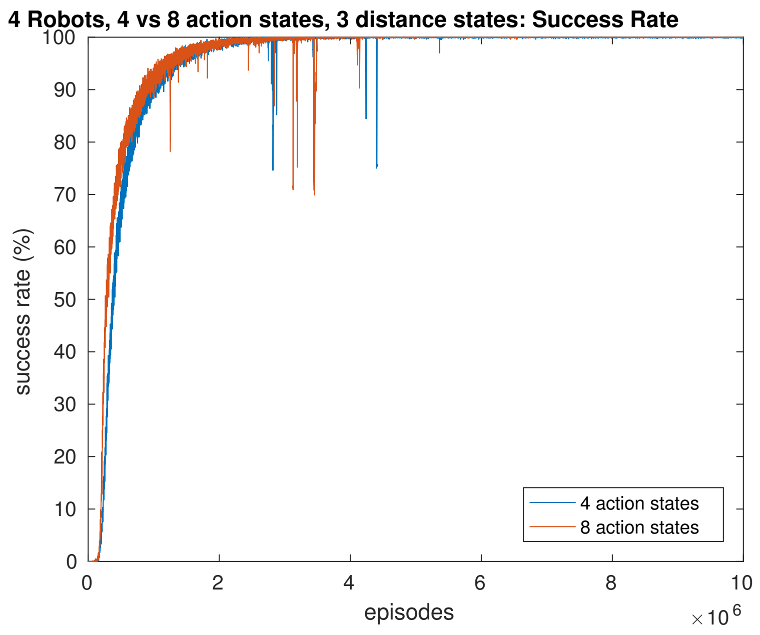


Figure 4.13: 4 Robots, 3 distance states success rate 4 actions VS 8 actions

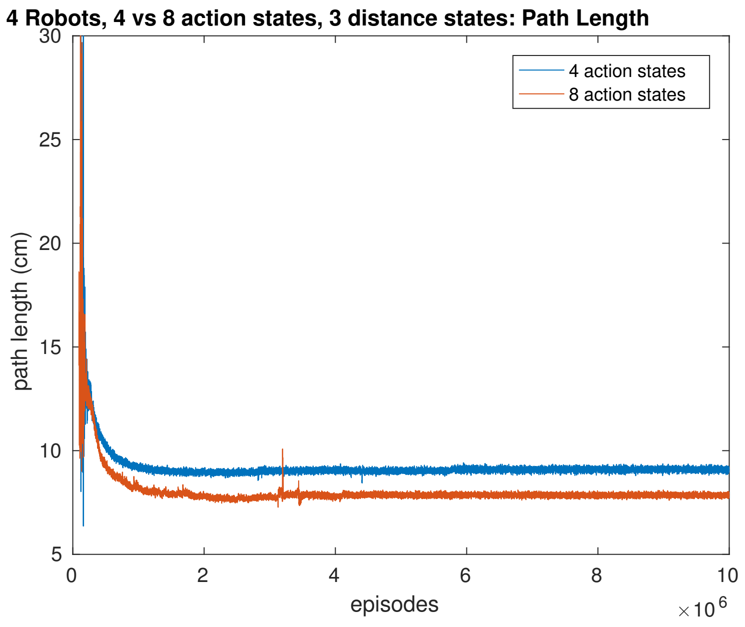


Figure 4.14: 4 Robots, 3 distance states path length 4 actions VS 8 actions

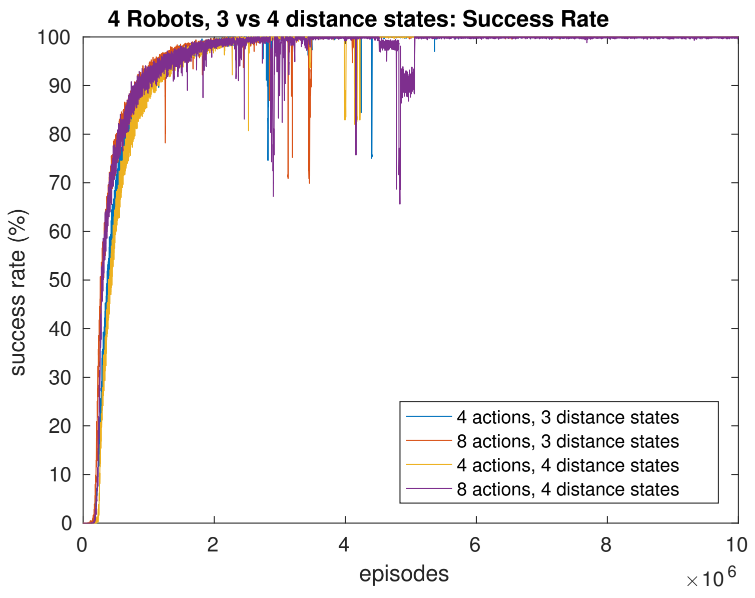


Figure 4.15: 4 Robots, 3 VS 4 distance states success rate 4 actions VS 8 actions

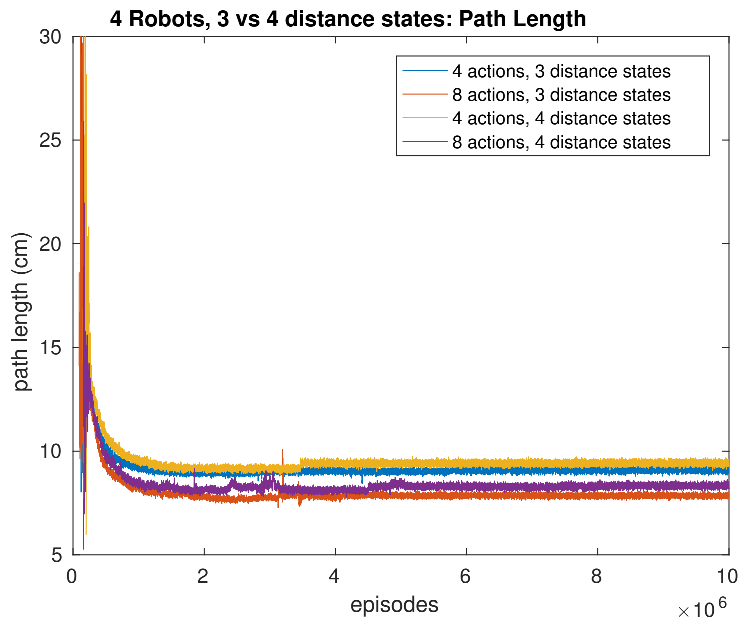


Figure 4.16: 4 Robots, 3 VS 4 distance states path length 4 actions VS 8 actions

Distance state	Actions	ε Update	Success rate	Path length (cm)
Mean distance 4 states	4	100	99.868%	8.53
Mean distance 4 states	4	500	99.846%	8.62
Min distance 4 states	4	500	99.992%	9.4186
Min distance 4 states	8	500	99.92%	8.2794
Min distance 3 states	4	500	99.983%	9.0838
Min distance 3 states	8	500	99.997%	7.8513

Table 4.3: 4 Robots summary

The summary in table 4.3 shows that combination of minimum distance state with 3 stages and 8 actions pair yields the best result. Although every policy shows very similar performance, minimum distance is the safest policy to use because the mean distance state can be unreliable. For example, in case of one robots is very close and one other robot is very far, this will results in medium state of the distance state despite the thread of imminent collision.

4.3 Performance Evaluation

The performance matrices that we focused on in this thesis consist of success rate and path length. The success rate reflects the completion of the task. As for the path length, this indicates how good is the policy respective from Q-learning algorithm parameters. The performance evaluations are done by using Q-values table for each experiment to validate with 100,000 episodes for 5 times and calculate the mean success rate and path length for the final results.

As we have mentioned in previous chapter, the exploration vs exploitation parameters is the key to finding optimal policy. We found that excessive exploration can cause the policy to steer the robot out off optimal path or to stuck in loops.

4.3.1 Single Robot Performance

The optimal paths generated from single robot experiments are the best they could possibly be. No matter where the robot is located, it is able to navigate to the goal as shown in figure 4.17.

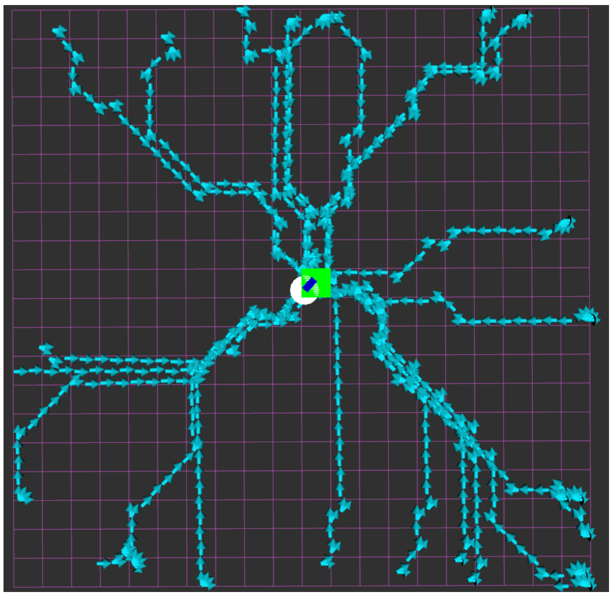


Figure 4.17: Single robots optimal paths

As for the exploration probability (ε) update strategy, for 20×20 grid environment an update per 100 episodes yield the best result for success rate and path length combination with 100% success and average path at $5.86cm$ (table 4.1). In 40×40 grid environment an update per 200 episodes yield the best results with with 100% success and average path at $12.84cm$ (table 4.2).

4.3.2 4 Robots Performance

In 4 robots environment, performance from two distance states are very close and often indistinguishable. However the path from 8 action states are more optimized than path from 4 action states despite the increased complexity. However when working in multiple robots environment we have to consider the path that each robot takes in order to avoid a collision.

Best Case Scenario

In a best case scenario each robot is able to reach its destination and able to keep distance between other robots at the same time for both **3 distance states** as seen in figure 4.18 and **4 distance states** as seen in figure 4.19. The depicted scenario occurs when there are no crossing between any of the paths, which depends on the starting points.

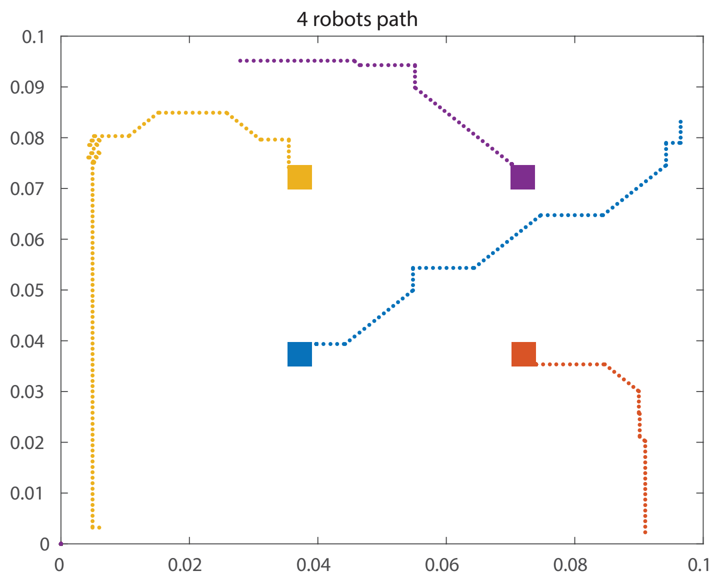


Figure 4.18: 4 Robots optimal path, 3 distance states

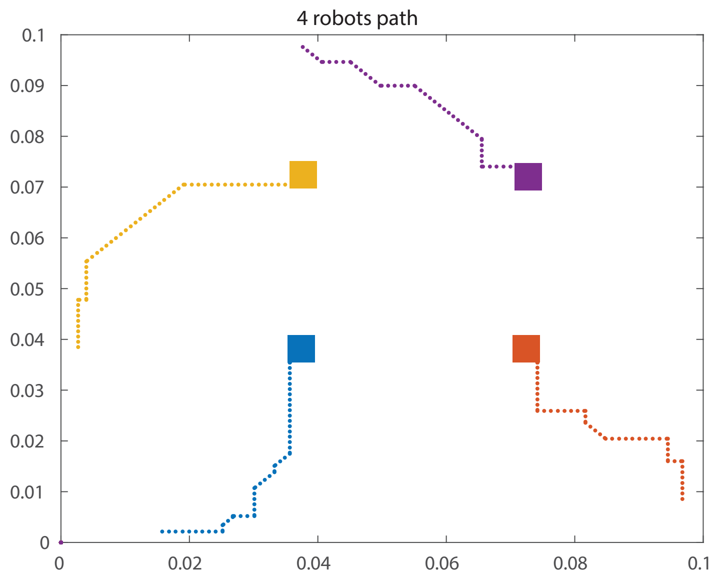


Figure 4.19: 4 Robots optimal path, 4 distance states

Worst Case Scenario

If the starting point of two or more robots are on the opposite side of the board, sometimes the **3 distance states** will not be able to keep distance between robots that crossing pass and will results in near misses, as seen in figure 4.20. But for the **4 distance states** the collision avoidance seems to work correctly as seen in figure4.21. However all robots are still able to navigate to their respective goals without collision with other obstacles.

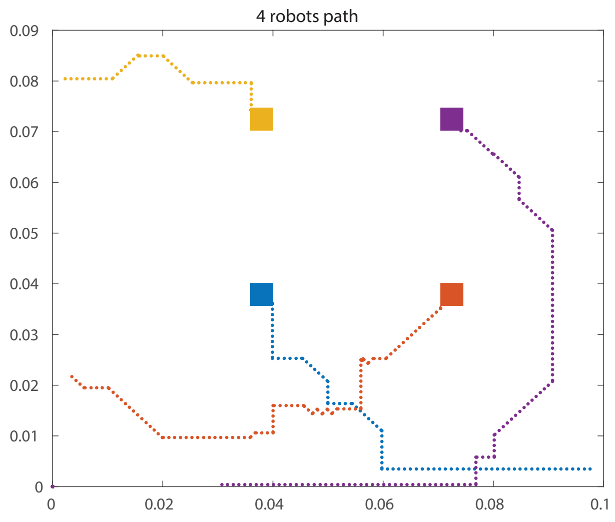


Figure 4.20: 4 Robots, worst case scenario, 3 distance states

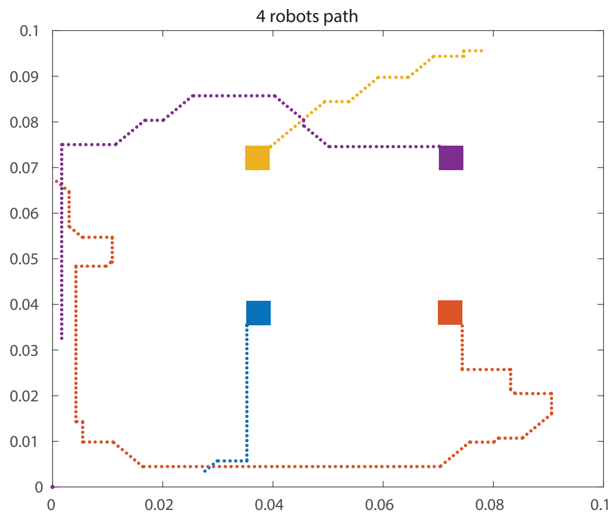


Figure 4.21: 4 Robots, worst case scenario, 4 distance states

4.3.3 Performance Comparison, 1 Robot VS 4 Robots

The comparison between using only one robot in the environment and using 4 robots in the environment can be seen in figure 4.22 and 4.23. Path length for single robot environment is the shortest because it has no need to avoid collision with other robots or obstacles. However the path for 4 robots environment are not much longer than single robot meaning there are 4 tasks done in the in $1.34\times$ times of one task. Furthermore the success rates are all above 99.5% which is remarkable. Finally, out of all the tests done in visualization, none of them fails, all the robots are able to reach their targets.

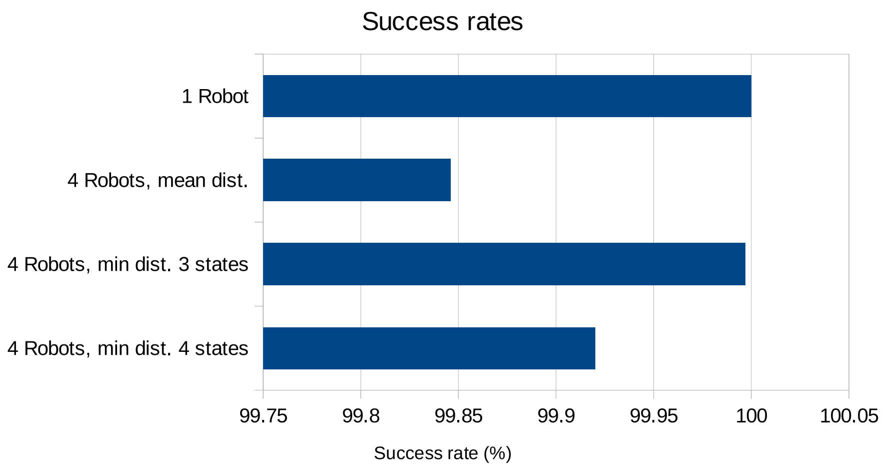


Figure 4.22: Performance analysis: success rate

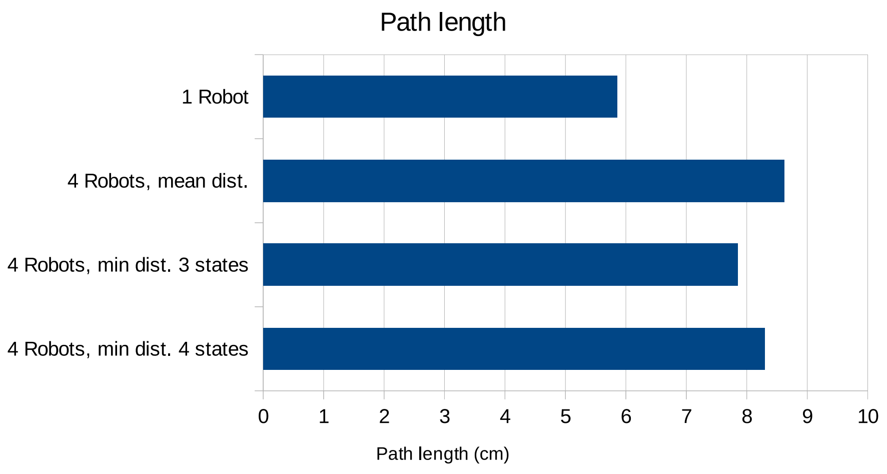


Figure 4.23: Performance analysis: path length

Chapter 5

Conclusions

The main objective of this thesis is the development, implementation and evaluation of reinforcement learning for multiple mini-robots navigation in unknown environments. Specifically, we elaborate on using Q-Learning algorithm for multiple robots in collaborative system. In this section we summarize results of the current research and discuss some possible future works.

5.1 Conclusions

After lengthy experiment, we have found that the results are very promising. All estimated policies found are near optimal, success rates are more than 99%. The results from noisy environment are shown that noise is negligible for this agent as there is no difference in convergence rate for each noise level. This is to be expected because the state are divided in grids and the robot is not fast enough to escape a grid in only one time step. In one robot environment the result from longer exploration period, such as updating the exploration probability (ε) once every 500 episodes can result in negative effect as the success rate drop but path length does not improve.

For multiple robots environment, we choose to work in collaborative perspective giving the workload sharing capability to the agents. In this environment, choosing an appropriate reward function is much more important than single robot environment because not only a robot need to reach its target but also need to avoid both static and dynamic obstacles.

In 4 robots environment, the results from **mean distance state** and **smallest (min.) distance state** are very close in terms of overall success rate and path length. However when we look at the spike that randomly happened, mean distance state is unreliable. This occur when all the states have not been visited or have not visited enough. Due to the design of the distance state that take all robots in to account and if some are very far and some are very close the the selected robot, this will results in medium distance which can also cause the problem with collision and near miss portrayed in figure 4.20. We have come to the conclusion that the **smallest or minimum distance state** yields the **best result overall**. Also it is rare to achieve the full 100% success rate from any reward function and exploration strategy. However there were some experiments that results in 100% for 4 and 2 robots environment but not very often. The way to quantize the distance in to states also plays an important role in the learning process. An appropriate set of distances need to be selected for each environment according to the number of robots in the environment, size of the robot and size of the workspace. The compromise between exploration-exploitation is needed to produce good result (shortest path) and good learning rate. The balance between good exploration and excessive exploration need to be discovered for each type of problem and environment. Finally, the results also show that multiple robots can complete multiple tasks only a fraction slower than single robot completing a single task. We have come to conclusion that autonomous *multiple robots navigation using reinforcement learning* in collaborative environment to be feasible and effective.

5.2 Future Work

In the following, we present some interesting directions for future research that elaborate on a number of open issues related to the methodologies presented in this thesis.

Further research is required for better understanding and use of reinforcement learning in multiagent systems. Especially alternative reward functions need to be developed for different type of scenarios and problem size. So far we purposed two simple yet effective reward functions and state spaces with low complexity. The state spaces need be developed further for better results such as interoperating the directions of other robots in to the state to improve the collision avoidance. The different model-based for value function approximations scheme could be used to improve the accuracy and accelerate the learning process.

The scalability which is one of many strengths of a multiagent system need be further developed in to this scheme by adding more robots in to the environment. After proving that all of the methods are working in the simulator, we would like to implement it in real robots to work in real-world environment.

Bibliography

[1] P. Vartholomeos, K. Vlachos, and E. Papadopoulos, *Analysis and Motion Control of a Centrifugal-Force Microrobotic Platform*, vol. 10. 2013.

[2] N. Tziortziotis, *Machine Learning for Intelligent Agents*. PhD thesis, University of Ioannina, 2015.

[3] M. Wooldridge and N. Jennings, *Intelligent agents: theory and practice*. 1995.

[4] G. Weiss, *Multiagent Systems A Modern Approach to Distributed Modern Approach to Artificial Intelligence*. The MIT Press, 1999.

[5] E. H. Durfee, V. R. Lesser, and D. D. Corkill, "Coherent cooperation among communicating problem solvers," *IEEE Transactions on Computers*, vol. C-36, pp. 1275–1291, Nov 1987.

[6] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. MIT Press, second ed., 2017.

[7] C. Szepesvari, *Algorithms for Reinforcement Learning*. Morgan and Claypool Publishers, 2009.

[8] R. Sutton, *Learning to Predict by the Methods of Temporal Dierence*. 1988.

[9] C. Watkins and P. Dayan, *Q-learning*. 1992.

[10] L. Busoniu, R. Babuska, B. D. Schutter, and D. Ernst, *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press.

[11] A. Martinez and E. Fernandez, *Learning ROS for Robotics Programming*. Packt Publishing, 2013.