UNIVERSIDADE FEDERAL DE MINAS GERAIS
Instituto de Ciências Exatas
Programa de Pós-Graduação em Ciência da Computação

Thiago Meireles Grabe

An investigation of different state representations for learning to coordinate in swarm navigation

Belo Horizonte
2023

Thiago Meireles Grabe

An investigation of different state representations for learning to coordinate in swarm navigation

Version

Thesis presented to the Graduate Program in Computer Science of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Master in Computer Science.

Advisor: Luiz Chaimowicz
Co-Advisor: Leandro Marcolino Soriano

Belo Horizonte
2023

Grabe, Thiago Meireles.

G727i An investigation of different state representations for learning to coordinate in swarm navigation [recurso eletrônico] / Thiago Meireles Grabe– 2023.
1 recurso online (85 f. il, color.) : pdf.

Orientador: Luiz Chaimowicz.
Coorientador: Leandro Marcolino Soriano.
Dissertação (Mstrado) - Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciências da Computação.
Referências: f. 77-85.

1. Computação – Teses. 2. Robótica – Teses. 3. Aprendizado profundo. – Teses. 4. Sistemas multiagentes – Teses.
I. Chaimowicz, Luiz. II. Soriano, Leandro Marcolino
III. Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Computação. IV. Título.

CDU 519.6*82.10(043)

Ficha catalográfica elaborada pela bibliotecária Irénquer Vismeg Lucas Cruz
CRB 6/819 - Universidade Federal de Minas Gerais - ICEx

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

An investigation of different state representations for learning to coordinate in swarm navigation

THIAGO MEIRELES GRABE

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. LUIZ CHAIMOWICZ - Orientador
Departamento de Ciência da Computação - UFMG

PROF. LEANDRO SORIANO MARCOLINO - Coorientador
Universidade de Lancaster - Universidade de Lancaster

PROF. DOUGLAS GUIMARÃES MACHARET
Departamento de Ciência da Computação - UFMG

PROF. ARMANDO ALVES NETO
Departamento de Engenharia Eletronica - UFMG

Belo Horizonte, 1 de março de 2023.

Aos meus pais, não só pela vida, mas pela caminhada...

Acknowledgments

Most of my acknowledgments are to people who speak Portuguese, and this section is written in Portuguese so they understand without a translator.

Durante a minha trajetória houveram referências e pessoas que me deram o suporte técnico, psicológico e afetivo para seguir em frente. Uma dissertação é a formalização de um processo que começa um sonho e desejo de fazer pesquisa além de crescer profissionalmente, mas essa jornada, chamada *mestrado*, recebe inúmeros significados, especialmente pelas pessoas que passam, nos ajudam e principalmente contribuem com a joranda.

Logo, agradeço imensamente:

- À minha família, pais e irmão além da Alika pelo suporte ao longo de toda a vida. Sem o apoio, cada um a sua maneira, não seria possível as importantes conquistas que tive e ainda vou ter ao longo da minha trajetória.
- À minha esposa pela compreensão, carinho e sorriso mesmo em meus momentos mais desafiadores ao longo do caminho.
- Aos amigos de *Verlab*, em especial João, Murilo, Rezeck, Maurício... entre tantos outros que me ajudaram quando prazos estavam apertados, mas também pelos inúmeros cafés na cantina do *ICEx* e pelas gargalhadas: sem elas, não teria graça as incontáveis horas no laboratório.
- Ao meu orientador Luiz Chaimowicz e meu coorientador Leandro Soriano, não somente pelo apoio técnico e serem extremamente talentosos, mas pelo amor com que exercem suas profissões, o carinho e paciência para me auxiliar e orientar durante a minha pesquisa.
- À CAPES, CNPq e Fapemig pelo apoio a essa pesquisa.
- Por fim, agradeço às diversas versões do Thiago e as inúmeras escolhas que me trouxeram até este momento. A escolha pelo estudo nem sempre é a mais óbvia e tãopouco a mais fácil, mas em todos as grandes encruzilhadas da minha vida, a educação se mostrou sempre a melhor aposta.

"*Mestre não é quem sempre ensina, mas quem de repente aprende.*"
(João Guimarães Rosa)

Resumo

Sistemas multiagentes têm sido um foco importante de pesquisas para diversas aplicações desde a robótica com robôs interagindo e competindo entre si ou em jogos em que diversos agentes possuem um objetivo comum ou conflitante. É notável não só a evolução dos algoritmos determinísticos que abordam tarefas como navegação de robôs evitando colisões em ambientes controlados, como também abordagens que envolvem técnicas de aprendizado de máquina incluindo aprendizado por reforço. Recentemente, estes métodos baseados em aprendizado por reforço se mostram eficazes ao aproveitar redes neurais e a capacidade de processamento para tarefas complexas.

Dentre as tarefas de enxames ou grupo de robôs uma tarefa em específico se mostra desafiadora: Navegação segregada. Esta tarefa consiste em um grupo de robôs, do mesmo grupo ou não, navegarem pelo ambiente em que estão inseridos e possuem o objetivo de navegar em conjunto de um ponto inicial até um objetivo final de forma coordenada evitando colisões com os outros robôs no ambiente ou obstáculos.

O estado da arte neste campo conta com algoritmos determinísticos que atendem a tarefa não apresentam uma escalabilidae adequada para um número grande de robôs ou vários grupos de robôs heterogêneos. Além disso, essas técnicas possuem algumas restrições como o conhecimento prévio de características dos robôs e atributos como o grupo que um determinado robô pertence.

Para entender como o aprendizado por reforço pode auxiliar a comunidade a superar alguns destes desafios, propomos duas metodologias para aplicar aprendizado por reforço com redes neurais (*Deep Reinforcement Learning*) na tarefa de navegação segregada de grupos heterogêneos de robôs que pertencem a grupos distintos.

A primeira abordagem utiliza uma representação dos estados a partir dos sensores dos robôs para mapear o ambiente e os robôs dentro de um determinado campo de visão. Nessa estratégia, a ação que o robô executa está diretamente ligada à quantidade de robôs na região de visão dos sensores. O treinamento é realizado em um determinado *setup* no qual o robô é exposto a diferentes cenários com relação ao número de robôs e número de grupos. Em tempo de teste, essa política aprendida é aplicada no mesmo *setup* de treinamento, mas também o que chamamos de *extrapolação*, no qual a política é aplicada em cenários mais complexos com mais robôs por grupo.

Já a nossa segunda abordagem utiliza uma representação de elipse para representar o grupo de robôs além de uma estratégia de campos potenciais para evitar as colisões. Neste caso, a representação dos estados é feita pelos parâmetros da elipse de cada grupo e

testamos duas formas de utilizar campos potênciais para evitar colisões: a primeira sendo o vetor resultado como uma entrada na rede neural a ser aprendida e a outra como soma vetorial após a resposta da rede.

A primeira metodologia apresenta resultados relevantes para cenários em que o *setup* de teste é o mesmo de treinamento. Ao expandirmos o teste e aplicarmos a política em cenários mais desafiadores, os resultados não são tão bons. Já com a segunda abordagem, em especial a segunda metodologia, obtivemos uma taxa de sucesso expressiva em relação à primeira metodologia da mesma abordagem e mesmo ao compararmos com a primeira abordagem.

Palavras-chave: robótica; sistemas multi agentes; aprendizado por reforço; matemática

Abstract

Multi-agent systems have been a pivotal area of research for numerous applications in robotics, where robots interact and compete with each other, or in games where multiple agents have common or conflicting objectives. This field has seen significant advancements not only in deterministic algorithms addressing tasks such as collision-free robot navigation in controlled environments but also in machine learning techniques, including reinforcement learning. Recently, reinforcement learning methods leveraging neural networks and computational power have proven effective for complex tasks.

A particularly challenging task within robot swarms is collision-free segregated navigation. This involves a group of homogeneous or heterogeneous robots navigating from a starting point to a final destination in a coordinated manner, avoiding collisions with other robots and environmental obstacles.

Current state-of-the-art approaches rely on deterministic algorithms, which, while effective, do not scale well to large numbers of robots or multiple heterogeneous robot groups. Additionally, these methods require prior knowledge of the robots' characteristics and group attributes.

To address these limitations, we propose two methodologies to apply deep reinforcement learning in the segregated navigation of heterogeneous robot groups.

The first approach uses state representations from robot sensors to map the environment and the robots within a certain field of view. Here, the robot's actions are directly influenced by the number of robots detected within the sensor's range. Training is conducted in a controlled setup, exposing the robots to varying scenarios regarding the number of robots and groups. During testing, the learned policy is applied both within the training setup and in more complex extrapolated scenarios with a higher number of robots per group.

The second approach employs an ellipse representation for the robot groups and a potential field strategy to avoid collisions. State representations are based on each group's ellipse parameters. We explore two potential field applications: one where the resultant vector serves as an input to the neural network and another where it is summed vectorially after the network response.

The first methodology yields significant results when the testing setup mirrors the training environment. However, its performance declines in more challenging scenarios. Conversely, the second methodology, particularly the latter variant, demonstrates a higher success rate than the first and initial approaches within the same approach.

Keywords: robotics; swarms; reinforcement learning; mathematics

List of Figures

List of Tables

List of Algorithms

Contents

Chapter 1

Introduction

Robotics research has been at the forefront of mobility, defense and industry applications using the cutting edge technologies such as computer vision [87], neural networks [93] or reinforcement learning [42]. These developments enable a huge range of possibilities from aerospace applications like the Mars rovers [88] or in the automotive industry using robots for assembly lines or logistics [5]. Alongside the technology developments we also face the necessity to reduce human risks when dealing with robots in hazardous operations or the cost reduction of these robot's system, then to achieve same results with lower risks and cost it is been developed idea of robot swarms applications.

Swarm is concerned with coordinating a multiple robot system where a collective behavior emerges from the interaction among the robots and the environment. [1] Usually, robotics swarms are inspired by biological swarms such as the flock of birds, groups of ants or a hive of bees [6] which a large numbers of simple robots are used to perform a coordinated task. Those robots as individuals do not have the full capability to complete the tasks, but as a group they present an emergent behavior that allows the entire swarm to achieve their goals [60]. These robots ideally should be low cost with limited hardware components to justify the deployment of several robots to perform the given task and the execution of swarm robotics tasks put forward challenges like Software Architecture, Hardware, Communications, Sensor Fusion, Testing, Cooperative Intelligence among others [6].

Examples of swarms tasks are aggregation of robots from the same group spread in a common environment, navigation of a given group from point A to point B, foraging in an environment searching for resources, transportation of heavy objects, coordinated movement to design patterns, assembly applications in several industries like automotive and many other applications researchers are investigating from different industries from farming to aerospace and defense. These tasks allow the development of new collective and cooperative intelligence through new algorithms and paradigms to enhance the key results for each application.

Several methods and algorithms have been proposed to address safe navigation

[1] It is also referred as multi-robot system which a swarm the robots are homogeneous and within a multi-robot system they may be heterogeneous [75]

tasks. For obstacle avoidance, the authors in [26] proposed the Velocity Obstacles (VO), which was improved by [85] as Reciprocal Velocity Obstacles (RVO). Then, the authors proposed the Optimal Reciprocal Collision Avoidance (ORCA) [85], which represents one of the best methods for safe navigation considering a group of robots. Moreover, in recent years, as can be seen in several research areas, Machine Learning (ML) has become increasingly present in applications using multi-agent systems [37], and [45]. Several researchers have been working on solutions for the most diverse types of tasks using ML algorithms. However, when proposing an approach that uses this type of technique, it is necessary to analyze a series of factors that can significantly influence the quality of the developed solution, such as the type of learning to be used, the most appropriate modeling to represent what will be learned by the robots and the tools used to evaluate the quality of what is being learned.

Of particular interest to this dissertation, the Reinforcement Learning (RL) framework applied to multi-robot systems is extremely important as a research area. Using RL as a learning method for complex algorithms that might request higher processing power or more expensive hardware may be a solution to deploy simpler robots to solve complex tasks. Maybe the most common Reinforcement Learning algorithm, the Q-Learning [89], has been used to solve several problems like path planning [57] and [43] but also extended using deep neural networks to solve collision avoidance tasks [52], [19] and [51].

This work explores a deep reinforcement learning approach to map states to actions of a robot inserted in a group of robots that share the same objective and aim to complete a specific task. The robot task may be summarized as going from a starting position in a shared environment with other groups and reaching a final position alongside its group peers (other robots). At each time step, the robot uses the proposed methodology to map the state to a feasible action, avoiding collisions with other robots that share the same environment and keeping cohesion with its group as represented in Figure 1.1 where each frame represents a time step when the robot is sharing the environment with other robots and other groups.

We investigate two approaches using different state representations: the first with a spatial representation of the robot's sensor and the second using a group of robots to build ellipses that encapsulate each group.

1.1 Motivation

The swarm's goal may be a variety of tasks, like self-organization, when the robots in the swarm, based on their sensors, determine the direction to move and the interac-

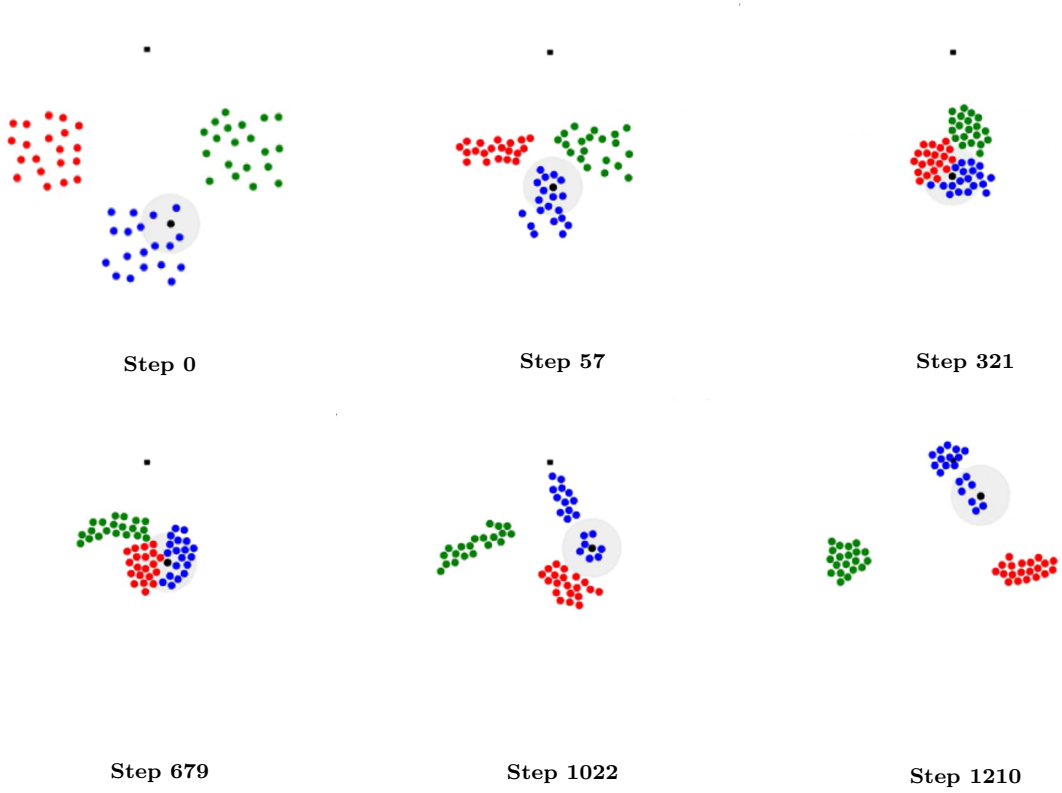


Figure 1.1: Example of segregated navigation task.

tion with other robots. Other tasks like rescue operations or even assembly in different industries may be the objective of a swarm.

In many applications, specific tasks may need the division of the swarm into distinct groups of agents, each specialized in particular subtasks. For instance, in resource collection scenarios, different types of robots may be designated to collect specific resources [53]. These specialized groups are then deployed to areas where their respective resources are abundant. Maintaining effective coordination and segregation among these groups presents significant challenges. The robots must navigate safely while ensuring they do not interfere with other groups, and their efficiency must not be compromised by these constraints.

A pertinent example is found in rescue operations, where distinct groups of robots may be tasked with different aspects of the rescue, such as debris removal, medical supply delivery, and victim search [41]. Each group must perform its function without disrupting the others, necessitating advanced coordination strategies. Similarly, in industrial assembly, robots might be specialized for various assembly stages, requiring precise coordination to ensure seamless integration of parts [33].

In swarm robotics, reinforcement learning (RL) methods such as Q-Learning are often employed to map sensor signals to a state space representing the environment. While Q-learning has advantages in discrete environments due to its simplicity, it encounters sig-

nificant limitations in more complex, continuous environments where different groups have distinct targets. The scalability of Q-Learning is hindered by its reliance on maintaining a table of state-action values, which becomes impractical as the number of states and actions increases. Additionally, Q-Learning struggles with generalizing learned behaviors to unseen states and efficient exploration of intricate environments [78].

Deep reinforcement learning (DRL) addresses these limitations by leveraging neural networks to approximate value functions or policies, enabling the handling of large state and action spaces inherent in continuous environments. DRL's ability to generalize from past experiences to new states is particularly valuable in dynamic environments where agents must adapt quickly. Moreover, DRL can process complex representations directly from raw sensor inputs and handle continuous action spaces, essential for precise control and coordination in swarm robotics [58]. By overcoming the limitations of tabular methods, DRL provides a robust framework for developing advanced swarm intelligence and coordination strategies, making it an invaluable tool for modern swarm robotics applications [49].

The state representation is also important to a DRL paradigm associated with the task it is being developed. First, it determines how well the learning algorithm can generalize from past experiences to new, unseen situations. Second, it impacts the efficiency of the learning process, as a good representation can simplify policy learning by emphasizing important patterns and reducing noise. Third, it enables the integration of diverse sensory inputs, such as a vision for images, touch for sponsored robots, and also for abstract sensing like laser distance measurement, into a coherent state description that the learning algorithm can use effectively.

In this dissertation, we investigate and propose two methodologies based on Deep Reinforcement Learning that use two different states' representations and use the success rate as a key metric to measure the results. In the first part of the work, the representation and simulation consider that the robot belongs to the FL-ORCA proposed by Inácio *et al.* [40], which means that the learning robot is known for the other robots in the simulation. The second state representation uses an ellipses concept to represent the state space; the learning robot is a complete stranger to the other robots, and it no longer belongs to the FL-ORCA swarm, increasing the task difficulty.

1.2 Objectives

The objective of this dissertation is to investigate a Reinforcement Learning approach to the segregated navigation task of a swarm. More specifically, given a swarm

of robots divided by groups and sharing the same environment, we propose two different strategies with two different states' representations for the RL framework that enable a robot to learn the behavior presented by the group to which this robot belongs. The other robots are ruled by the FL-ORCA algorithm, while in the first strategy, the learning robot, despite using our methodology, is recognized by the other robots as part of the environment. On the other hand, in the second strategy, we use a completely different state representation using the ellipses concept, and the other robots are not aware of the learning robot's presence in the environment creating a more complex scenario for the learning process.

Local View Representation: The first methodology investigates a state representation using a local sense of the environment. The state is basically a circular construction based on the distances from the other robots and their angles within the sense region. This investigation led us to interesting results where we set the training process in an environment with a certain number of groups and robots. We simulate the testing time with an extrapolation where the testing scenario considers more robots and/or groups than the learning robot was exposed to during the training session. We also investigate the impact when the robot is not part of the FL-ORCA group algorithm and when the robot does not belong. This work was published in the *4th International Symposium on Swarm Behavior and Bio-Inspired Robotics (SWARMS 2021)* [79].

Ellipses Representation: Our second approach uses a different state representation and considers ellipsis and a covariance matrix to define the coverage for all group robots. This methodology is inspired by the work presented by Soriano *et. al* [65] where the authors use the ellipse idea but for a different task without Reinforcement Learning. We show some interesting results and improvements when the learning robot is not considered part to the FL-ORCA group. Although the first strategy using a sensor-based state representation achieves a satisfactory success rate, the learning robot has the restriction of being part of the FL-ORCA group, and we remove this restriction using a different space state construction to achieve reasonable results.

1.3 Outline

Here, we present a summary of each part of this dissertation, highlighting what the reader may expect from each chapter.

- **Chapter 2:** This chapter outlines the background of this dissertation about swarms and reinforcement learning. It begins with important concepts about both disci-

plines and emphasizes the algorithms we use in this work. Lastly, we present a literature review related to the main topics of this dissertation.

- **Chapter 3:** The third chapter describes our approach considering the **Local View Representation**. It outlines and explains the methodology in detail and presents the main results considering when the robot belongs and does to the FL-ORCA group. We also investigate two main failure reasons and how they relate to the number of robots and groups.

- **Chapter 4:** The **Ellipses Representation**is described in chapter 4. The chapter outlines the second methodology, where we use an ellipsis representation of the state alongside the local sense of the learning robot. Also, in this work, we present results considering when the robot belongs and does not belong to the FL-ORCA group. Finally, we investigate two main failure reasons and how they relate to the number of robots and groups. Going beyond that, we compare the best sensor-state representation of this chapter with Chapter 3.

- **Chapter 5:** Finally, in this chapter, we conclude our work by highlighting the key points of the work and reinforcing the investigation, results, and possible contributions of the presented methodologies. We also introduce several possibilities relevant to the community in future works.

Chapter 2

Background & Related Work

In this section, we review several related studies covering the two main topics of this work: Reinforcement Learning and Robotic Swarms.

2.1 Reinforcement Learning

In this section, we present a review of reinforcement learning and the fundamentals behind the learning process. As described by Sutton and Barto [78], it is a type of machine learning where an agent learns to make decisions by performing actions in an environment to maximize some notion of cumulative reward. Unlike supervised learning, where the learning process is guided by a set of labeled examples, RL relies on the agent exploring the environment and receiving feedback in the form of rewards or penalties. This trial-and-error approach is fundamental to how RL algorithms operate.

We begin by introducing the RL paradigm and the Markov Decision Process (MDP), which forms the backbone of most RL algorithms. Following this, we explore essential algorithms such as Deep Q-Learning (DQN) [58] and Double Deep Q-Learning (DDQN) [36]. This discussion is supplemented with references about applying these methodologies in the robotics area and recent advancements in the field.

2.1.1 Reinforcement Learning Paradigm

The canonical Reinforcement Learning (RL) problem can be formalized as a learning process where the agent interacts with the environment without examples or supervision, or even complete models of the world in which it is inserted and aims to achieve long-term goals [78], [28]. The RL paradigm differs from supervised learning, whose main

focus is to map a function from inputs to outputs given a labeled set of input-output pairs. In addition, RL also differs from unsupervised learning since the main goal of RL is to maximize the reward. In contrast, in unsupervised learning, we try to uncover patterns of unlabeled data.

The Markov Decision Process (MDP) was first introduced by Bellman [11]. It is a time stochastic control process represented by a tuple $(S,\ A,\ P,\ R,\ \lambda)$ where S is a set of states, A denotes a set of discrete actions, $P : S \times A \times S \Rightarrow \{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$ is the transition probability function, R is the reward function, and λ is the discount factor applied. It provides a mathematical framework for modeling decision-making in environments with probabilistic transitions.

The agent's objective is to learn a policy $\pi(a|s)$ that maximizes the expected cumulative reward, known as the return G_t:

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

This return accounts for all future rewards starting from time step t, discounted by γ.

In summary:

- **State space** (S): The set of all possible states in the environment.
- **Action space** (A): The set of all possible actions the agent can take.
- **Transition probability function** $(P(s'|s,a))$: The probability of transitioning to state s' from state s after taking action a.
- **Reward function** $(R(s,a))$: The immediate reward received after taking action a in state s.
- **Discount factor** (γ): A factor $0 \leq \gamma \leq 1$ that represents the importance of future rewards.
- **Policy** $\pi(a|s)$: A policy is a strategy that the agent employs to determine the next action based on the current state. It can be deterministic (mapping states to specific actions) or stochastic (mapping states to a probability distribution over actions).

The MDP represented in Figure 2.1 exemplifies the interaction where actions influence immediate rewards and subsequent situations or states and those future rewards.

At each time-step t, the agent selects an action $a_t \in A$. Then, the agent receives a reward $\tau_t \in R$ and changes from state $\mathrm{s}_t \in S$ *to* $\mathrm{s}_{t+1} \in S$. The agent aims to learn a policy that maps every state to a probability distribution over all possible actions, maximizing the expected reward obtained in an episode.

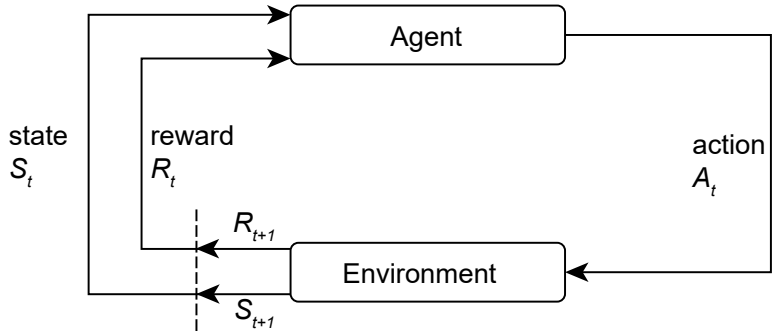


Figure 2.1: Agent-environment interaction in a Markov Decision Process. Adapted from [78]

.

Several authors extended those ideas and comprehensively understood RL, especially Sutton *et. al* [10] while it was first proposed by Bellman [12]. Although agents and environment are highly cited in RL systems, other aspects are also important: policy, reward, value function, or the model of the environment.

2.1.2 *Policy*

The policy maps the agent's states to actions. It may be a function or look-up table and can also be stochastic, which means a probability associated with the choice of each possible action [78].

- **Deterministic Policy:**

  A deterministic policy, denoted by π, is a function that maps states to actions. This means that given a state s, the policy π specifies the action a that the agent should take.

  $$\pi : \mathcal{S} \rightarrow \mathcal{A}$$

  where $\mathcal{S}$ is the set of all possible states and $\mathcal{A}$ is the set of all possible actions. For a given state $s \in \mathcal{S}$, the action a chosen by the policy is $\pi(s)$.

- **Stochastic Policy:**

  A stochastic policy, denoted by $\pi(a|s)$, specifies a probability distribution over actions given a state. This means that for each state s, the policy provides the probability of taking each possible action a.

  $$\pi(a|s) = P(A_t = a \mid S_t = s)$$

Here, $\pi(a|s)$ represents the probability that action a is chosen when the agent is in state s.

- **Optimal Policy:**

  An MDP often aims to find the optimal policy π^* that maximizes the expected return. The optimal state-value function $V^*(s)$ and the optimal action-value function $Q^*(s, a)$ are defined as:

$$V^*(s) = \max_\pi V^\pi(s)$$

$$Q^*(s, a) = \max_\pi Q^\pi(s, a)$$

  An optimal policy π^* achieves these optimal value functions.

2.1.3 *Reward*

On each step, the agent should interact with the environment, and given an action and the policy, it receives an associated reward. The reward signal defines what good and bad outcomes for the agent are. The reward changes the policy behavior: if a bad reward follows an action the agent selects, the policy may be changed to select another action shortly [78].

In RL, an agent learns to make decisions by interacting with an environment. The agent's goal is to maximize cumulative reward over time. The reward function is a crucial component of this framework as it defines the agent's objective. Proper reward design is essential for guiding the agent's behavior towards the desired outcomes [21]

2.1.3.1 Reward Function

A reward function R provides feedback to the agent about the immediate benefit of taking action a_t in a given state s_t [17, 61]. Mathematically, the reward function is defined as:

$$R(s_t, a_t, s_{t+1})$$

where s_t is the current state, a_t is the action taken, and s_{t+1} is the resulting state after taking action a_t.

The reward r_t received by the agent at time step t is:

$$r_t = R(s_t, a_t, s_{t+1})$$

2.1.3.2 Cumulative Reward

The agent's objective is to maximize the cumulative reward, often called the return. Depending on the problem, the return can be defined in various ways [21, 61]. For example, in an episodic task, then return G_t from time step t can be defined as the sum of future rewards:

$$G_t = \sum_{k=0}^{T-t} \gamma^k r_{t+k}$$

Where:

1. $\gamma \in [0, 1]$ is the discount factor determining future rewards' present value.
2. T is the time step when the episode ends.

2.1.3.3 Designing the Reward Function

Reward design is critical because it influences the learning process and the resulting policy. A well-designed reward function should:

1. **Encourage Desired Behavior:** Rewards should be given for actions that lead to desired outcomes [21].
2. **Avoid Unintended Incentives:** Poorly designed rewards can lead to undesirable or unintended behaviors [34, 22].
3. **Balance Immediate and Future Rewards:** The discount factor γ helps balance short-term and long-term gains [91].

Regarding reward design strategies, we highlight the use of *Sparse* and *Dense* rewards. While *Sparse* are given infrequently, often only when a goal state is reached [55], the *Dense* rewards are provided more frequently, possibly at each time step, providing constant feedback [39].

For example, in a navigation task using a *Sparse reward*, we can associate a reward r_t at a given time step t if the state s_{t+1} is a terminal or goal state. On the other hand, if we consider a *Dense Reward*, the reward r_t can be a function f of a given environment feature, for example, the distance from a target position considering the navigation task.

1. Sparse reward:

$$R(s_t, a_t, s_{t+1}) = \begin{cases} r_t & \text{if } s_{t+1} \text{ is the terminal state} \\ 0 & \text{otherwise} \end{cases}$$

2. Dense reward:

$$R(s_t, a_t, s_{t+1}) = \text{distance}(s_{t+1}, \text{goal state})$$

2.1.4 *Value Function*

The value of a state is the total reward an agent can expect to accumulate over the future, starting from that state [78].

The value of a policy π is often evaluated using value functions. There are two main types of value functions:

– **State-Value Function $V^\pi(s)$:** This function gives the expected return (sum of rewards) starting from state s and following policy π.

$$V^\pi(s) = \mathbb{E}^\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \middle| S_0 = s \right]$$

where γ is the discount factor ($0 \leq \gamma \leq 1$), R_{t+1} is the reward received at time $t+1$, and $\mathbb{E}^\pi$ denotes the expectation given that the agent follows policy π.

– **Action-Value Function $Q^\pi(s,a)$:** This function gives the expected return starting from state s, taking action a, and thereafter following policy π.

$$Q^\pi(s,a) = \mathbb{E}^\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \middle| S_0 = s, A_0 = a \right]$$

2.1.5 *Model*

Finally, the model of the environment may also be an important key to the RL problem. The model represents the environment dynamics and allows inferences from the environment in which the agent or robot is inserted. To solve an RL problem, there are two main strategies that consider *model-free* where the agent performs a trial and error approach and the *model-based* methods, where the agent plans each action based on an environment's model.

The optimal action-value function is based on the *Bellman equation* 2.1 [11]. In real applications, finding an optimal action-value function is impossible, so it is common to use a function to approximate it, and it can be linear or even a neural network. It converges to the optimal action-value function $Q_i \to Q^*$ as $i \to \infty$ [78].

$$Q(s,a) = Q(s,a) + \alpha \left(r + \gamma \max_{a' \in A} Q(s',a') - Q(s,a) \right), \tag{2.1}$$

One of the most basic RL algorithms is the Q-Learning presented by Watkins *et. al* [89] based on the *Bellman Equation* [11]. It is an off-policy method, and it is considered a Temporal Difference Learning (TD) that allows the robot to learn directly from raw experience without a model of the environment's dynamics [78]. The learned value function Q approximates the optimal value function.

One of the main drawbacks of Q-learning is that it overestimates some state actions under certain conditions, specifically with insufficiently flexible function approximation. Thrun & Schwartz [81] showed that if an action contains errors uniformly distributed, the target is overestimated proportionately by the number of actions taken during training. Moreover, as Q-Learning is a table-based algorithm, every state-action pair should be explored to converge to the optimal policy.

To overcome the Q-learning tabular limitation, DQN [58] has been successful in various complex tasks, such as playing Atari games from raw pixel inputs. One of the key advantages of DQNs is their ability to generalize across similar states due to the use of deep convolutional neural networks, which extract relevant features from the input space, facilitating efficient learning. This capability has led to significant advancements

in artificial intelligence and has showcased the potential of combining deep learning with reinforcement learning.

However, DQN also has several drawbacks. One significant issue is the instability and divergence during training, which can arise from the correlation between consecutive samples and the large updates to the Q-values. This instability can lead to sub-optimal policies or even failure to learn. Additionally, DQN can suffer from overestimation bias, where the algorithm tends to overestimate action values, leading to less efficient learning and poorer performance [78]. We explore the DQN in sub-section 2.1.6.

Double Deep Q-Networks (DDQN) was developed to address some of the key limitations of DQN, particularly the overestimation bias. The core idea behind DDQN is to decouple the action selection from the action evaluation in the Q-learning update, thereby reducing the overestimation of action values. Using two separate networks to estimate the Q-values – one for selecting the action and another for evaluating the action – DDQN can provide a more accurate estimate of the action values [36].

Moreover, DDQN enhances the stability of the learning process by mitigating the issues caused by correlated samples and large updates. The separation of action selection and evaluation helps smooth the learning updates, leading to a more robust and consistent improvement in policy performance. This advancement has made DDQNs a preferred choice for many reinforcement learning applications, where stability and accuracy are crucial [36]. We explore the DDQN in sub-section 2.1.7

2.1.6 DQN algorithm

The Reinforcement Learning paradigm aims to learn directly from high dimensional inputs known as states to actions that may be continuous or discrete depending on the agent, the environment, and the task to be learned. With the recent advances in deep learning techniques, it is possible to enhance the state representation with high-level features from the environment and process these raw data into meaningful information for the agent, for example, in computer vision applications [20] and [72] or even audio generation [63].

In view of the deep learning applications within the RL paradigm, the Deep Q-Network (DQN) was proposed by V Mnih *et. al* [58] and uses convolutional neural networks to control policies from video data. It is an extension of the Q-learning algorithm that uses a neural network to approximate the action-value function, $Q(s, a; \theta)$, where θ represents the parameters of the neural network. DQN has been instrumental in enabling agents to learn from high-dimensional sensory inputs, such as images, and perform well

in various complex tasks.

One of the key features of DQN is the Experience Replay, which allows the agent, instead of learning from consecutive state transitions, to store experiences (s, a, r, s') in a replay buffer. During training, mini-batches of experiences are randomly sampled from this buffer. This technique reduces the correlation between consecutive samples, leading to more stable and efficient learning.

Another important feature is a separate target network to generate the target values for the Q-learning updates. The parameters of this target network (θ^-) are updated periodically to match the parameters of the primary network. This helps to stabilize training by preventing oscillations or divergence in the Q-value estimates.

One important advantage of the DQN compared to the Q-learning method is that the DQN allows the representation of high-dimensional states representations and uses a deep neural network. Additionally, another advantage of DQN over other methods is that at each time-step, the weights are updated more efficiently given the *experience replay* method. As the *experiences* are randomly sampled from the dataset D, it reduces the variance of the updates as consecutive samples would be less efficient due to correlations between consecutive experiences.

The Q-value update rule in Deep Q-Networks (DQN) is given by the following equation:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right] \tag{2.2}$$

Where:

- $Q(s_t, a_t)$: The current Q-value for the state s_t and action a_t.
- α: The learning rate, which determines how much new information overrides the old information.
- r_{t+1}: The reward received after taking action a_t in state s_t.
- γ: The discount factor, which determines the importance of future rewards.
- $\max_{a'} Q(s_{t+1}, a')$: The maximum Q-value for the next state s_{t+1} over all possible actions a'.

The term $(r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t))$, represents the temporal difference (TD) error, which measures the difference between the predicted Q-value and the updated Q-value based on the observed reward and the estimated optimal future value. The Q-value update adjusts the current Q-value towards this updated value.

The loss function in Deep Q-Networks (DQN) is used to minimize the difference between the predicted Q-values and the target Q-values. It is given by the following equation:

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\right)^2\right] \tag{2.3}$$

Where:

- $\mathcal{D}$ represents the distribution of experiences sampled from the replay buffer.
- r is the reward received after taking action a in state s.
- s' is the subsequent state.

DQN has been successfully applied to various robotics tasks, where the agent learns to perform complex actions based on high-dimensional inputs like images from cameras. In robotics applications, Yushihisa *et al.* [82] uses DQN to explore the Arm Manipulation of a robot using visual inputs from a camera to understand the position and orientation of objects and then learn the optimal actions to manipulate the arm. In a similar work Zhang *et al.* controls a robotic manipulator with visual perception only using the DQN.

Algorithm 1 puts into perspective the steps used by the authors, especially the stored transitions D, and how it is used to optimize the loss function applying a gradient descent step.

Algorithm 1: DQN

Initialize replay memory $\mathcal{D}$ with capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights θ^-
for *episode* $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{if terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{if non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ.
    Every C steps, reset $\hat{Q} = Q$
  end for
end for

Algorithm 2.1: DQN Algorithm.

It begins by initializing a replay memory with a specified capacity to store past experiences. It also initializes the action-value function Q with random weights θ and a target action-value function $\hat{Q}$ with weights θ^-.

The algorithm starts with an initial sequence and preprocesses it for each episode. At each time step, it selects an action either randomly, with a certain probability ϵ, or by choosing the action that maximizes the action-value function Q for the current state. After executing the selected action, it observes the reward and the next state.

The algorithm then stores the transition in the replay memory and samples a random minibatch of transitions from it. For each transition in the minibatch, it calculates the target value based on whether the next state is terminal or not. It then performs a gradient descent step to minimize the difference between the target value and the predicted value by the action-value function Q. Periodically, the target action-value function $\hat{Q}$ is updated to match the action-value function Q. This process repeats until the end of the episode and continues for the specified number of episodes.

2.1.7 DDQN - Double Q-learning algorithm

The Double Deep Q-Network (DDQN) was developed to address certain limitations of DQN, such as the overestimation of action values. This overestimation occurs when the same network is used both to select and evaluate actions, leading to biased estimates of the Q-values. DDQN mitigates this by decoupling the selection of the action from the evaluation of the action's value, using two separate networks: the primary Q-network and the target Q-network. As introduced by Hasselt et al. [36], this approach helps to reduce the bias and provides more stable and reliable training.

In DDQN, the action selection is still performed using the primary network, but the evaluation of the selected action is done using the target network. This modification results in a more accurate estimate of the action values, leading to improved performance and stability in training. The use of two networks ensures that the updates to the Q-values are less susceptible to the noise and correlations present in the training data, making the learning process more robust. DDQN has shown significant improvements in various reinforcement learning tasks, particularly those involving high-dimensional state spaces and complex decision-making processes [36].

The Q-value update rule in DDQN is given by the following equation:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1}, \arg\max_{a'} Q(s_{t+1}, a'; \theta); \theta^-) - Q(s_t, a_t) \right] \quad (2.4)$$

Where:

- $Q(s_t, a_t)$: The current Q-value for the state s_t and action a_t.

- α: The learning rate, which determines how much new information overrides the old information.
- r_{t+1}: The reward received after taking action a_t in state s_t.
- γ: The discount factor, which determines the importance of future rewards.
- $Q(s_{t+1}, \arg\max_{a'} Q(s_{t+1}, a'; \theta); \theta^-)$: The Q-value evaluated using the target network parameters θ^- for the action selected by the primary network.

The DDQN loss function is designed to minimize the difference between the predicted Q-values and the target Q-values. It is given by the following equation:

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\left(r + \gamma Q(s', \arg\max_{a'} Q(s', a'; \theta); \theta^-) - Q(s, a; \theta)\right)^2\right] \tag{2.5}$$

Where:

- $\mathcal{D}$ represents the distribution of experiences sampled from the replay buffer.
- r is the reward received after taking action a in state s.
- s' is the subsequent state.

DDQN has demonstrated its effectiveness in various complex reinforcement learning tasks. In autonomous driving, DDQN has been applied to optimize the decision-making processes of autonomous vehicles, resulting in safer and more efficient navigation [73] and [50]. The ability of DDQN to provide stable learning and accurate value estimation makes it suitable for applications requiring high reliability and precision.

Algorithm 2 outlines the steps used by DDQN, emphasizing the separate target network to reduce overestimation bias and improve learning stability.

The DDQN starts by initializing a replay memory with a specified capacity to store past experiences. It also initializes the action-value function Q with random weights θ and a target action-value function $\hat{Q}$ with weights θ^-.

For each episode, it starts with an initial sequence and pre-processes it. At each time step, it selects an action either randomly, with a certain probability ϵ, or by choosing the action that maximizes the action-value function Q for the current state. After executing the selected action, it observes the reward and the next state.

The algorithm then stores the transition in the replay memory and samples a random mini-batch of transitions from it. Each transition in the mini-batch calculates the target value based on whether the next state is terminal or not, using the target network for evaluation. It then performs a gradient descent step to minimize the difference between the target value and the predicted value by the action-value function Q. Periodically, the target action-value function $\hat{Q}$ is updated to match the action-value function Q. This process continues for the specified number of episodes.

Algorithm 2: DDQN

Initialize replay memory $\mathcal{D}$ with capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights θ^{-}
for *episode* $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set
    $$y_j = \begin{cases} r_j & \text{if terminal } \phi_{j+1} \\ r_j + \gamma Q(\phi_{j+1}, \arg\max_{a'} Q(\phi_{j+1}, a'; \theta); \theta^{-}) & \text{if non-terminal } \phi_{j+1} \end{cases}$$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ.
    Every C steps, reset $\hat{Q} = Q$
  end for
end for

Algorithm 2.2: DDQN Algorithm.

2.1.8 Comparison of Double Deep Q-Network (DDQN) and Deep Q-Network (DQN)

The Double Deep Q-Network (DDQN) was introduced as an enhancement to the Deep Q-Network (DQN) to address the significant issue of overestimation of action values in the original DQN algorithm. This overestimation arises from the DQN's use of the same network to select and evaluate actions, resulting in biased Q-value estimates. To mitigate this, DDQN employs two separate networks: the primary Q-network for action selection and the target Q-network for action value evaluation. This decoupling helps reduce bias and provides more stable and reliable training [36].

The Q-value update rule in DDQN is represented by equation 2.2 where the target network θ^{-} is used for evaluating the selected action, thereby yielding a more accurate estimate of the action values. The DDQN's approach, which involves minimizing the loss function represented in equation 2.5 ensures that updates to Q-values are less affected by noise and correlations in the training data.

Consequently, DDQN demonstrates significant improvements over DQN in various reinforcement learning tasks, particularly in high-dimensional state spaces and complex decision-making scenarios, as evidenced in studies by Lillicrap et al. [49] and Shalev-

Shwartz et al. [74]. This stability and precision make DDQN highly effective for applications requiring reliable and accurate decision-making processes.

2.2 Robotic Swarms

Robotic swarms are composed by a large number of robots that can be used to perform complex and collective tasks. Usually, in robotic swarms it is desired a high-level system that implies scalability, flexibility and robustness [68] to keep the simplicity of the system when compared to a single robot system that has a centralized control. Moreover, the system should also be able to operate with heterogeneous groups with different sizes.

2.2.1 Swarms Tasks

In the swarm robotics discipline, various tasks are studied by observing the natural behaviors of animals; for example. These can be solved using a group of robots and algorithms that allow each robot to use local interactions with the environment to make their own decisions [23]. To accomplish those tasks, the robots can use local vs. global sense, which is how the robot observes the environment. In the local sense, the robot has a limited view of the environment, and everything inside this environment is sensed by the robot's sensors. On the other hand, in the global sense approach, usually, the robot can have full information about the environment, for example, other robots and obstacles. Additionally, the robot may use various algorithms and techniques to enhance its localization and navigation through the environment [80].

The aggregation task involves self-organization methods to control the robot's movement based on the environment sensing. These tasks are related to the robot's direction and interactions with other agents and the environment aiming to gather a number of robots in a common target place. Self-organization is inspired by natural animal formations, such as bird flocks and insects. The attraction/repulsion forces [59] can be formulated using local sense and using the distance information among the robots to evaluate a feasible velocity [86], [24]. Probabilistic methods use a finite state machine that is characterized by two main states: *walk* and *wait* [7]. Evaluating aggregation tasks may involve calculating the robots in a certain area [8] or the distance from a fixed reference point in space [35].

The flocking strategy consists of a known behavior from birds where a large group of agents moves toward a target location. This behavior emerges from the group with local interactions among the agents, taking into account the distance and orientation among the near agents and avoiding collision [67]. Also, Siegwart *et. al* [76] apply the potential artificial fields approach where each agent is independent and navigates based on its local sensing. Turgut *et. al* [83] apply their methodology in real robots generating a flocking in a swarm using a virtual heading system (VHS) measuring the distance and relative orientation of nearby robots and obstacles. On the other hand, Santos *et. al* [69] uses an extension of the Velocity Obstacle concept and proposes a strategy with a hierarchical abstraction and flocking behaviors. To measure the flocking performance, an intuitive metric can be the distance between the evaluated robot and its swarm center of mass [83]. A more complex metric is proposed by Baldassarre *et. al* [9], and uses measurements to evaluate an agent and group behaviors during a flocking simulation.

On the other hand, the navigation task implies the capability of a robot or a group of robots to reach a target location from a different start position, limited sensing and localization capabilities, and using help from other robots in the environment. Different techniques may be applied in navigation tasks such as gradient descent [54], a stigmergy mechanism using indirect coordination through the environment [90], or even a neuro-fuzzy controller [62].

Another important task considering a robotics swarm is the segregated navigation (SV). Safe navigation is only part of a robot's task in a complex scenario. It does not necessarily lead the swarm to its objective by increasing performance. The robots might still face difficulties in systems with heavy congestion. The SV task considers environments where a number of robots from different groups are deployed. Using strategies to optimize navigation without congestion among groups reaching their goals is important.

Some algorithms also address the collision avoidance issue like those proposed by Krontiris *et. al* [46], and Antonio Franchi *et. al* [27] while the main task, like coordination or navigation, is still effective.

Several other tasks for swarms are widely studied, and all have their applications and importance for various scientific fields. For example, area coverage [13], search and rescue operations [18], foraging [38], task allocation [47], assembly [92], etc.

2.2.2 FL-ORCA Algorithm

Collision avoidance is crucial for any robotics system. Safe navigation with robot sensing and acting in its environment is a challenging task, especially when multiple robots

are deployed in the same environment configuring swarms' of robots [56] or even for crowd simulation [66].

The Velocity Obstacles (VO) proposed by Fiorini *et. al* [26] and the Reciprocal Velocity Obstacles (RVO) proposed by Van den Berg *et. al* [85] are important works that use Minkowski sums, and their algorithms are designed to avoid collisions with obstacles that are moving at a known velocity in the environment. While the first method considers a known obstacle velocity which is an important constraint, the second also considers that the obstacle or the other robots can have their velocities affected by the interaction with the given agent and modify the collision avoidance strategy to guarantee safe navigation.

To overcome VO and RVO restrictions like known obstacles velocity, Berg *et. al* [15] proposed the ORCA method for collision avoidance where multiple robots share the same environment and must avoid collisions. The authors also use a decentralized and simplified model, and the robots are circular or have a convex polygon form and move in a 2-dimensional environment. The ORCA algorithm ensures local collision-free navigation [15]. The algorithm evaluates for all agents in the environment the possible velocities that ensure a free collision path among all agents. Then, the agent chooses the optimal allowable velocity calculated by solving a system of linear equations.

Although the ORCA algorithm has ensured safe navigation, avoiding collision among the robots, complex tasks involve not only safe navigation but also group segregation, and ORCA is unable to handle common target problems when it reaches a certain configuration that the robots are close to their goal but does not necessarily reach their final goal position or area [77].

To leverage ORCA, Inácio *et. al* [40] proposed the FL-ORCA, an alternative to achieve safe navigation by avoiding congestion using the SV approach. The authors apply flocking concepts and the ORCA algorithm [15] to avoid collisions among the agents from the different groups sharing the environment. Even with a local view and unbalanced groups (groups with a different number of robots), the results presented by the authors outperform several similar SV works that use local sense to segregate a group of robots, but do not address navigation in the environment [25]. Other works address the SV task with local sense [70], but its performance is proportional to the radius at which the robot senses the environment [40].

Algorithm 3 presents the canonical version of the FL-ORCA algorithm. At each time step, a new velocity for the learning robot is set using specific evaluations considering *cohesion*, *separation*, and *alignment* of the group. Then, using a *state machine* proposed by the authors, it describes the possible situations each robot faces during the simulation at each time step. Additionally, the FL-ORCA is a decentralized algorithm and each robot executes the algorithm independently, but assumes that all other robots in the simulation are also ruled by the FL-ORCA.

The *state machine* in Figure 2.2 has several nodes that play an important role in

each decision:

- **Single Group:** This state is enabled when the robot does not sense any other agent from a different group. In this case, the robot moves towards the goal while maintaining group cohesion.
- **Vision Free:** In this case, the robot might sense one or more robots of different groups, but it has a free vision of its target.
- **Follower:** The follower case is when the robot does not have a clear view of its target, but a neighbor robot from its group is at one of the past states (Single Group or Vision Free). The robot will follow its neighbor toward the goal position in this case.
- **Turn Right:** The last possible state is when the robot is not in any of the previous scenarios. According to the Inácio *et. al*, this situation involves a congested situation, and as *traffic rule*, the robot tries to get around the congestion situation or area.

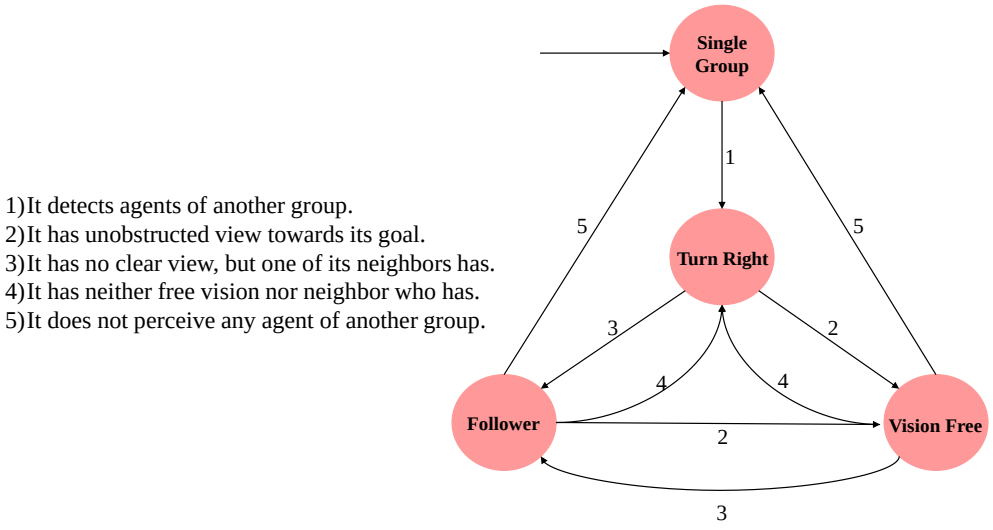


Figure 2.2: FL-ORCA *state machine*. Each state controls the robot velocity at each time step weighting each component of the FL-ORCA algorithm. Adapted from [40].

Algorithm 3: FL-ORCA Algorithm

Inputs: N_i List of neighbors of the robot i at each time-step
Output: $u_{FL-ORCA}$ as a new robot i velocity to be performed at each time-step
for *Each simulation time step:* **do**

$v_{\text{cohesion}} \leftarrow$ *Evaluate cohesion*;
$v_{\text{separation}} \leftarrow$ *Evaluate separation*;
$v_{\text{alignment}} \leftarrow$ *Evaluate alignment*;

Evaluate the current state of robot i according to *state machine* in Figure 2.2

$v_{\text{flock}} \leftarrow$ *Evaluate flocking*;
$v_{\text{pref}} \leftarrow$ *Evaluate best velocity*;
$u_{FL-ORCA} \leftarrow ORCA(v_{\text{pref}})$

Apply the new velocity $u_{FL-ORCA}$ to the agent i;

end for

Algorithm 2.3: FL-ORCA Algorithm.

2.3 Related Work

In this section, we review several related studies covering the two main topics of this work: Reinforcement Learning and Swarms. Particularly, we are interested in some works crossing both areas where the authors apply Reinforcement Learning methods to a multi-agent system or a swarm to accomplish a given task.

An important area of Artificial Intelligence is the development of autonomous robots that can interact with the environment and different robots to perform common or different tasks, and achieve goals. In a *ad-hoc teamwork* context, these robots that work as a group is supposed to perform without any prior knowledge of how to interact with the environment, and work together. Albrecht and Stone [3] presented a survey about modeling different approaches where some methods like *group modeling*, and *policy reconstruction* are in focus. Nevertheless, some works are usually presented in a simple grid-world scenarios [1], [64], considering a global view of the environment [30], and, despite few works such as the work presented by Katie *et. al* [29], their applicability to real robotic scenarios have not yet been demonstrated [2].

Several works were developed to allow the effective and safe navigation of a group of autonomous robots using Machine Learning. Godoy *et al.* (2013) [31] propose a methodology for the navigation task in environments with congestions that combine reinforcement learning with the ORCA collision control algorithm, where robots learn to adjust the preferred speed. In a different work, Godoy *et. al* (2015) [32] use Machine Learning and Game Theory to address the congestion problem in environments with many robots. They propose a reinforcement learning algorithm that uses a reward function that considers the agent's progress toward its goal and the impact of its behavior on the performance of robots close to him.

One important work presented by Long *et. al* (2017) [52], the authors use a convolutional neural network to allow robots to learn how to choose speeds that guarantee safe navigation from the Optimal Reciprocal Collision Avoidance (ORCA) algorithm [84]. In their work, the authors recorded the set of frames from the collision avoidance simulator and created a database considering the possible outcomes of the ORCA in different situations. It is later used as input to a deep neural network called *CANet*, responsible for calculating the speeds the robots should perform during their navigation. The experiments evaluate the author's methodology using different test scenarios, such as *Crossing* (agents from one group crossing a different group with different agents), *Circle* (agents are placed along a circle, and the target position for each agent is the antipodal position), and a task that is similar to the task we cover in our work called the *Swap* task, where two groups of agents move in opposite directions and swap their positions. Interestingly, the results presented by the authors show a failure rate of 2% for a given static obstacle

environment task, but they do not extend this metric to all setups and tasks, for example, the *Swap* task. Despite their results, several points remain open as the supervised learning approach is limited by the dataset size. Additionally, the results do not consider the Success Rate for all situations, especially for the *Swap*.

Using an interesting approach with deep reinforcement learning, Long *et. al* (2018) [51] address the collision avoidance task using a decentralized algorithm for a multi-robot system and local view based on the Proximal Policy Optimization (PPO) [71]. The authors expose the agents to different obstacle scenarios to achieve scalability and generalization. The agent objective is to avoid collisions while navigating through the environment. Moreover, they use a machine learning concept called *Curriculum Learning* proposed by Bengio *et. al* [14]. This strategy aims to train a machine learning or reinforcement learning model in an easier data or environment and increase the complexity of the samples imitating the human learning behavior. In other words, the first training part is done using simpler scenarios with a lower complexity of obstacles. In the second part, the complexity is increased, and the agent is exposed to a more complex scenario. The authors compare their results with the NH-ORCA proposed by Alonso *et. al* [4], a variant of the ORCA algorithm. The success rate of their methodology for the collision avoidance task is almost 100% for all setups. Only the setup with 20 robots, in the simulation, has the success rate decreases to 96.5%. Although the author's results are expressive, they do not consider the segregated navigation task in their experiments nor complex configurations with a considerable number of groups sharing the same environment.

Another important work was presented by Brito *et. al* [16]. The authors propose a deep reinforcement learning algorithm using LSTM layers to recommend a subgoal for a Model Predictive Control (MPC). This subgoal is expected to help the robot to go in the direction of its target position. Aside from the learning robot, the other robots that share the environment are ruled by the RVO algorithm. This work is different from the two previous works presented because it uses an MPC to generate locally optimal commands and avoid collisions. Additionally, the authors train the neural network using the PPO algorithm, similar to Long *et. al* (2018) [51]. The experiments considered a different number of robots on each simulation with 6, 8, and 10 robots, and the *Swap* task is applied to exchange the robot's position avoiding collisions. The results presented by the authors are impressive reducing the time to goal metric (time that a robot takes to achieve its target position) and the traveled distance on average. Moreover, the failure rate presented is 0% which shows the efficiency of the author's algorithm considering the tasks. This work considers a small number of robots in each simulation and the concept of a group is not covered since each robot has its own goal, albeit its interesting success in terms of success rate metric.

Although successful, the aforementioned methods focus on collision avoidance tasks, and the complexity of their work increases with different obstacles or the num-

ber of robots in the simulation. Moreover, the methodology proposed by Long *et. al* (2017) [52] uses a Convolutional Neural Network and a dataset of observable frames from previous simulations, configuring a Supervised Learning approach. On the other hand, the second approach uses the Reinforcement Learning paradigm, but they focus on collision avoidance, and the number of groups is limited to two groups, more specifically in the *swap* task. Finally, Brito *et. al* [16] show an interesting methodology using LSTM layers for a subgoal recommendation to help the robot in its task, which is in a sense, similar to what we propose in Chapter 4 to support the learning robot during the simulation execution to stay inside its group ellipse as a subgoal on each time step.

Despite of some similarities with the aforementioned methodologies, our work investigates not only the collision avoidance task but also the segregated navigation task, and it focuses on the FL-ORCA algorithm. We also use a sensor-based state representation in Chapter 3 and explore the success rate and the main failure reasons (*Lost* and *Captured* robot) considering the robotics swarm in the environment.

Furthermore, we extend the sensor-based observation of the environment to a more complex state space in Chapter 4. We consider an ellipse to model or encapsulate the group of robots to compute the state representation of a given time step. Interestingly, the ellipse representation using a collision avoidance strategy outperforms our sensor-based approach with a higher success rate.

Chapter 3

Sensor State Representation

This Chapter presents to the reader the first methodology called M_{sensor}. We propose a state representation that derives the environment perception from the robot's sensors in different regions. The goal is to navigate the group of robots from an initial position to the final position, with the entire group avoiding the congestion caused by the agglomeration of robots in the same region during the simulation and also the collision with other robots from its group or different groups in the simulation.

We investigate if a learning robot r_l trained in a simpler environment with the robots from its group may use the learned policy in more complex environments.

Here, we define a more complex scenario as a simulation setup in which the learning robot r_l is challenged to apply the learned policy in a different and more complex setup where it was first trained with more robots per group and additional unknown groups.

We name *Fixed* the simulations where the training and testing configurations are the same. For example, if we train with 5 robots per group and 4 groups, we also test with the same number of robots and groups. Conversely, the *Extrapolation* considers training with a given configuration, for example, 5 robots per group and 2 groups, but a testing scenario with 10 robots per group and 4 groups. The *Extrapolation* aims to increase the simulation complexity by adding more robots and groups, while the *Fixed* evaluates the same configuration during the testing period.

Interestingly, some *Extrapolation* results outperform the simpler situation of *Fixed* configuration. We highlight some possible causes of the presented behavior and study reasons that might explain the situation in terms of explored states and non-success causes.

3.1 Task Formulation

We consider a scenario in which a swarm, represented as a set $\mathcal{S} = \{r_1, r_2, ..., r_n\}$ of n holonomic robot, navigates in a 2D environment without obstacles. Each robot r_i

is represented by its pose $q_i = [x_i, y_i]$, with kinematic model given by $\dot{q}_i = u_i$. The entire swarm is formed by m distinct types (groups) of robots, which we represent by the partition $\Gamma = \{\Gamma_1, ..., \Gamma_m\}$, where each Γ_k contains all robots of type k and $|\Gamma_k| = n/m$. We assume that $\forall j, k : j \neq k \rightarrow \Gamma_j \cap \Gamma_k = \emptyset$, i.e., each robot is uniquely assigned to a single group. For notation purposes, the robot that is going through the learning process is defined as r_l where $\{l \in \mathbb{N} \mid 1 \leq l \leq n\}$ and the group to which r_l belongs is $\Gamma_d = \{d \in \mathbb{N} \mid 1 \leq d \leq m\}$.

Additionally, except the learning robot r_l, all other robots are ruled by the FL-ORCA, and each group Γ_k with robots of type k is also a group for the FL-ORCA. Each robot executes the Algorithm 3 to evaluate its preferred velocity. As a decentralized algorithm, the FL-ORCA assumes that each robot knows its neighborhood, its goal position, and which group the given robot belongs. That said, in this representation, the robots ruled by the FL-ORCA are aware of the learning robot r_l presence and even to which group it belongs, and the robots treat it as a member of their group while evaluating its preferred velocity. Nevertheless, r_l is the learning robot that is ruled by our methodology and the executed velocity will depend on the learned policy.

We also consider that, at the beginning of the task execution, each group $\Gamma_k \in \Gamma$ is segregated. To check if two groups are segregated, we calculate the average distances between robots from the same group and robots from different groups [48]. Thus, two groups of robots, for example, A and B, are considered segregated if the average distance between robots of the same group (group A or group B) is less than the average distance between robots of different groups (that is, between robots in group A and group B). Formally, we have $d_{XX} < d_{XY}$ and $d_{YY} < d_{XY}$ where d_{XY} is the average distance between the robots of groups X and Y:

$$d_{XY} = \frac{1}{|\Gamma_X|} \sum_{i \in \Gamma_X} \left(\frac{1}{|\Gamma_Y|} \sum_{j \in \Gamma_Y} (q_i - q_j) \right). \tag{3.1}$$

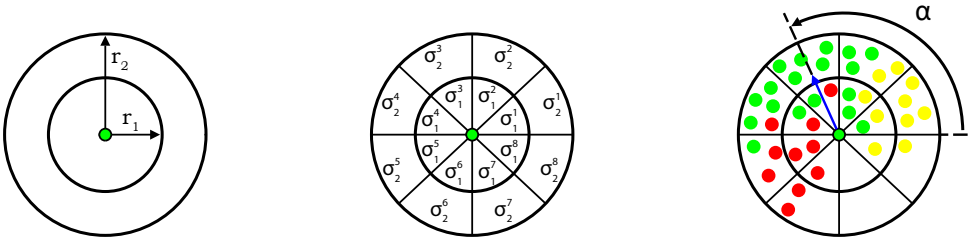
That said, we can define the problem addressed as follows: *make a robot learn to behave as part of a heterogeneous swarm formed by an arbitrary number of robots that navigate towards a goal in a shared environment while maintaining the condition of segregation between the swarm groups.*

3.2 Markov Decision Process Formulation

The robot's environment is shaped as Markov Decision Process (*MDP*) [1] represented by a tuple (S, A, P, R, λ) where S is a set of states, A denotes a set of discrete actions, P: S x A x S $\Rightarrow$ $\{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$ is the unknown transition probability function, R is the reward function, and λ is the discount factor applied.

We consider that robots have a local perception of the environment. We represent the robot's local sensing by a circular area and an external ring represented in Figure 3.1(a), divided into q sectors of equal size, as shown in Figure 3.1(b). Hence, we define region (circular area) $\sigma_1 = \{\sigma_1^1, \sigma_1^2, ..., \sigma_1^q\}$ delimited by radius R_1 and a second region (external ring) $\sigma_2 = \{\sigma_2^1, \sigma_2^2, ..., \sigma_2^q\}$ delimited by radius R_1 and R_2 around the robot. We also consider that the robot knows the group to which it belongs and the goal position.

Any robot or obstacle perceived within a distance R_1 is considered in a "closer zone" by the learning robot. On the other hand, if the distance to another robot or obstacle is farther than R_2, it cannot be detected by the learning robot representing the "external zone". A "mid-term zone" is represented by the external ring where all sensed robots are in the area limited by R_1 and R_2.



(a) (b) (c)

Figure 3.1: a) The two sense regions; b) Regions used for state representation; c) An example of the selected action (blue arrow) based on the state configuration.

3.2.1 State Space

We represent the state as a vector $S = \{s_1^1, ..., s_q^1, s_1^2, ..., s_q^2\}$ where s_j^i is the number of robots of the learning robot group (Γ_d) in sector j of the region i, subtracted by the number of robots of different groups sensed by r_l in the same region.

[1] Despite formally being an *POMDP*, due to the local observations, we abstract the environment representation as a Markov Decision Process (*MDP*) to be able to use the Double-DQN algorithm.

In Figure 3.1(c), we can see an example in which robots of distinct groups are represented by different colors, Γ_d is comprised of the green robots, and r_l is the green robot at the center. Note that r_l is surrounded by robots of three distinct groups (yellow, red, and green). The blue arrow represents the potential best action that can be selected by the robot since the sector corresponding to this action has the largest net amount of individuals of Γ_d.

Additionally, we consider a terminal state when the learning robot reaches the goal position $goal_d = [x_d, y_d]$ with its group, which means that all robots from Γ_d, including r_l, have the average distance $\delta_{X,goal}$ to the goal position (Equation 3.2) less than R_2. Other terminal states are reached when r_l no longer detects any other robot from its group in its sensing area or collides with another robot in the environment.

$$\delta_{X,goal} = \frac{1}{|\Gamma_X|} \sum_{i \in \Gamma_X} (q_i - goal_X). \tag{3.2}$$

3.2.2 Action Space

The action space is the set of allowed directions in a discrete space, where each action points towards the center of the sectors from regions σ_1 and σ_2. Given an action, $\{a \in A \mid 1 \leq a \leq q + 1\}$, we calculate an angle α_i towards the respective region as represented by Figure 3.1(c). This angle represents the direction from the origin to the region where most of the robots from its group are. Also, it may be calculated by Equation 3.3. Here, q represents the number of regions as previously defined and $\alpha_i \in [-\pi, \pi]$.

$$\alpha_i = \frac{\pi}{q} + \frac{a_i \cdot 2\pi}{q} - \pi. \tag{3.3}$$

We use a constant magnitude v for the performed velocity by r_l. The constant magnitude is set to simplify the state representation since our representation only takes into consideration the regions where the other robots are.

Finally, Equation 3.4 sets the applied velocity u_l to the learning robot.

$$u_l = (v \cdot \cos\alpha, v \cdot \sin\alpha) \tag{3.4}$$

3.2.3 Reward Design

The reward has a positive sign if the actions lead the robot to the desired states and a negative otherwise. More specifically, reaching states closer to the goal, a $+\tau/\eta$ reward is assigned where τ is a constant we set for all experiments. On the other hand, reaching states that do not represent a closer position to the goal, a reward $-\tau/\eta$ is assigned. Here, η represents the current step number of encouraging the robot to reach the goal faster. In other words, the faster the robot reaches the goal, the higher the reward.

We also set reward values for terminal states. If the terminal state is that the robot reached the goal with its group, $+\tau$ is assigned. Otherwise, if the terminal state is that the robot is lost (no longer senses any robot from its group) or there was a collision, the $-\tau$ is assigned. Table 3.1 illustrates the possible states and the assigned rewards for each case.

State	**Reward**
Closer to the Goal	$+\tau/\eta$
Further to the Goal	$-\tau/\eta$
Lost / Collision	$-\tau$
Goal reached	$+\tau$

Table 3.1: Reward Design for All Possible States Considering the Sensor State Representation.

3.3 Experimental Setup

As mentioned, we investigate if a robot that belongs to an FL-ORCA group but is not ruled by this algorithm can learn the group behavior and reach the goal position by avoiding collisions.

To evaluate our methodology, we test the learned policy in two different ways: The simple scenario is called *Fixed* and consists of the testing simulation using the same number of robots and groups of the training session. For example, we trained using 10 robots per group and 2 groups during the training session and tested the policy using the same number of robots per group and groups in the simulation.

Furthermore, the more complex scenario is called *Extrapolation* and represents a situation where we train in a simpler scenario with fewer robots per group and fewer groups but test in a more complex scenario with more robots and/or more groups. As an example, if we train the robot in an environment with 5 robots in each group and 2 different groups, we test in a more complex scenario with 4 groups of 5 robots (increasing the number of groups) or even the same 2 groups but with 10 robots on each group (increasing the number of robots).

In order to explore different setups, we use a different number of robots $(5, 10, 15, 20, 25)$ and groups $(1, 2, 3, 4, 8)$ when training. We start each training with the r_l placed in a random group Γ_k and randomly selected among the robots of Γ_k. With this random selection, we expect a better exploration among all possible start positions and want the robot to experience different difficulty levels due to its start position inside the group.

As an example, Figure 3.2 demonstrates the simulation during one task episode. In the first time step, the eight groups are in their start positions with all robots. In Figure 3.2, the learning robot r_l is represented by the black circle, while the goal position is the black square. The learning robot's group is the green one. Then, at each time step, the robots navigate the environment to achieve their goals while the learning robot r_l follows its group (green). The shaded area represents the sensor's area with radius R_2 (Figure 3.1(a)) and is represented only for r_l. In the final time step, the entire group reaches the goal position, and the episode is finished.

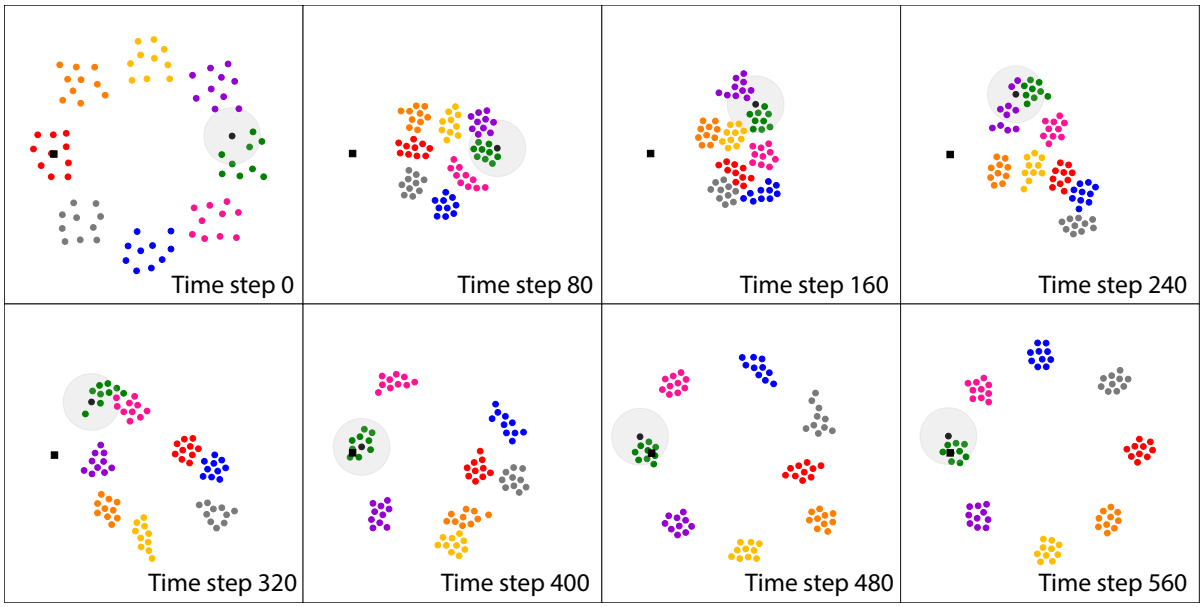


Figure 3.2: Simulation step by step. The learning robot r_l (black circle) is part of the green group, and we represent the group's target as the black square. The shaded area is the region sensed by r_l and is delimited by R_2 as represented in Figure 3.1(a).

The Algorithm 4 presents the pseudocode of the methodology M_{sensor}. At every experiment, we initialize the simulation with a set of robots $\mathcal{S}$ and a set of groups Γ

and all necessary parameters (Table 3.2) where λ is the discount factor used to update the action-value function Q according to the DQN presented by V Mnih *et. al* [58] and discussed in Chapter 2. We use the ε-greedy strategy for balancing between exploration and exploitation. Aside from the Reinforcement Learning parameters, we also set some simulation parameters that the learning robot uses. The η parameter sets the maximum number of steps each episode might have, the v parameter is used in the Equation 3.4, and the R_1 & R_2 are the sensor radius used by the r_l. Finally, the τ is used according to Table 3.1.

Finally, we conducted the experiments simulating each combination of number of robots and the number of groups, for 30 interactions. Each interaction were evaluated 100 times for statistical measures, thus a total of 3000 experiments each setup were made. In each process, we compute the success rate of the setup.

Parameter	**Value**	**Unit**
λ	0.001	-
ϵ	0.05	-
η	500	-
v	1	m/s
τ	1	-
R_1	0.5	m
R_2	1.5	m

Table 3.2: Parameters used during training for M_{sensor}.

Algorithm 4: M_{sensor}

Inputs: Set of Robots $\mathcal{S}$, Set of Groups Γ
Parameters: $\lambda, \epsilon, \tau, \eta, v, r_1, r_2$
Initialize action-value function Q with random weights
Set the initial state at time step 0

foreach *episode* **do**
  Choose a new learning robot $r_l \in \mathcal{S}$;
  foreach *step of episode* **do**
    Sense the environment and set the state space S
    Choose $\alpha \in A$ given S using policy derived from Q using ε-greedy;
    Calculate $\overrightarrow{u_l}$ using Equation 3.3 given α
    Execute $\overrightarrow{u_l}$, observe S';
    if S' *is terminal* **then**
      Receive final reward τ or $-\tau$
      Stop current episode;
    else
      Receive reward τ/η or $-\tau/\eta$;
    end if
    Update the action-value function Q according to Algorithm 1
  end foreach
end foreach

Algorithm 3.1: M_{sensor} Algorithm.

3.4 Results

This Section brings into focus the main results for Chapter 3, where the methodology introduced considers a state representation using the robot's local sense and the robots in the neighborhood. We also investigated the extrapolation of a simple training scenario applied to a more complex scenario. A sample of the executions for this representation can be found Local View Representation Videos.

The results are presented for a single group training scenario where we train the r_l on a single group and deploy it into a different group size. Then, we also present the multi-group scenario with groups ranging from 1 to 8.

In all figures, "r" shows the number of robots, and "gr" is the number of groups. As an example, in Figure 3.4 (a), the first two grouped bars (purple and red) present the results for 5 robots and 1 group in training time and test time. The second two grouped

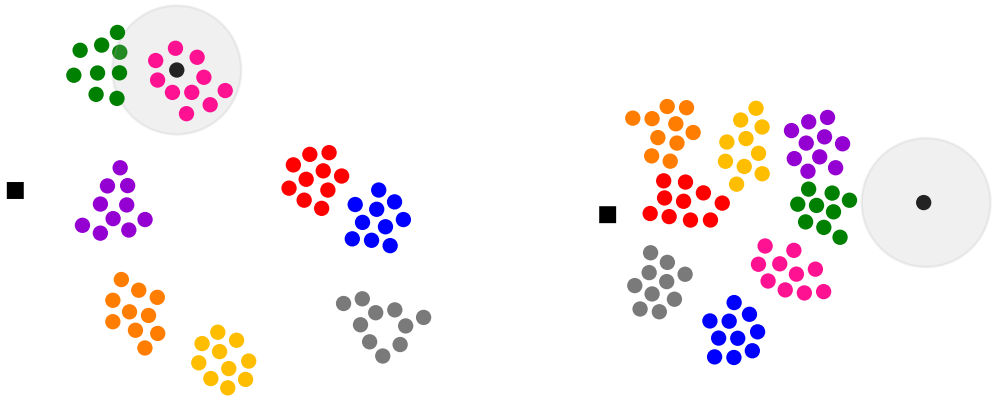
bars present the results for 10 robots and 1 group, and so on.

Moreover, the analysis should be made using Fixed/Extrapolation bars. Every *Fixed* bar states for the training setup described in the chart's x-axis and can be considered as a reference. All **purple** bars in Figure 3.4(a) were trained and tested with the x-axis setup, and as an example, the last grouped bars, the **purple** bar states for the training scenario with 25 robots and 1 group.

On the other hand, the *Extrapolation* bars (**red**) are results considering the training setup described by the caption, and the testing scenario used the x-axis setup. As an example, in Figure 3.4(a), the last grouped bars states for the training setup using 5 robots and 1 group and the testing scenario, named as *Extrapolation*, considers 25 robots and 1 group. That said, the **red** bar for this example demonstrates the results when training with 5 robots with 1 groups and testing the learned policy with 25 robots while the **purple** bar is the *Fixed* results where the results represent the training and testing setup as described in the x-axis.

With the configuration of this bar chart, it is possible to understand how the *Extrapolation* learned policy performs over the *Fixed* policy in the same setup scenario. This will be used for analyzing the results throughout this section.

Finally, we analyze the possible failures based on two situations: *Lost Robot* & *Captured Robot*. The *Captured Robot* is a robot surrounded by other robots that do not belong to its group. Figure 3.3(a) shows an example where the learning robot from the green group is captured by robots from another group (pink). *Lost Robot* is the Robot that can no longer detect since robots of its group inside its sensing region. This is exemplified in Figure 3.3(b).



(a) (b)

Figure 3.3: Example of (a) learning robot (black) from the green group being captured by the pink group; (b) learning robot (black) from the green group being lost from its group.

3.4.1 Single Group Training

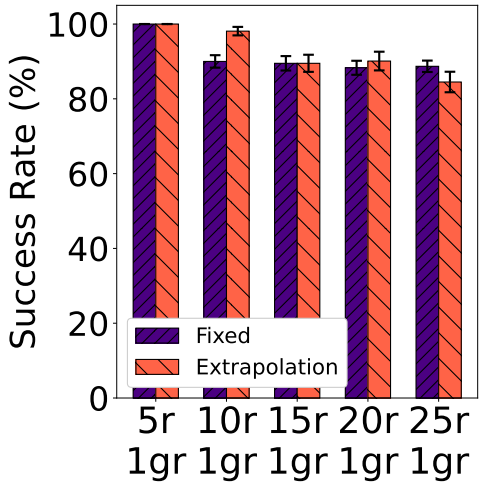
The results when training a single group are presented in Figure 3.4. Figure 3.4(a) and Figure 3.4(c) are situations where the training setup was performed using 5 robots and 1 group while Figures 3.4(b) and 3.4(d) are setups with 10 robots and 1 group.

The *Fixed* situation demonstrates that increasing the complexity in any case (raising the number of robots or groups) reduces the success rate, but the setup with 10 robots shows more stability, and increasing the number of robots instead of the groups causes less damage to the results. As an example, Figure 3.4(c) the success rate goes from 100% with one group to 50% with 8 groups while Figure 3.4(a) increasing the number of robots the success rate is more than 80%. On the other hand, the same analysis with 10 robots setup demonstrates less damage to the success rate: Figure 3.4(b) stays around 85% for all configurations Figure 3.4(d) it may be observed a decrease, but smaller when compared to Figure 3.4(c).

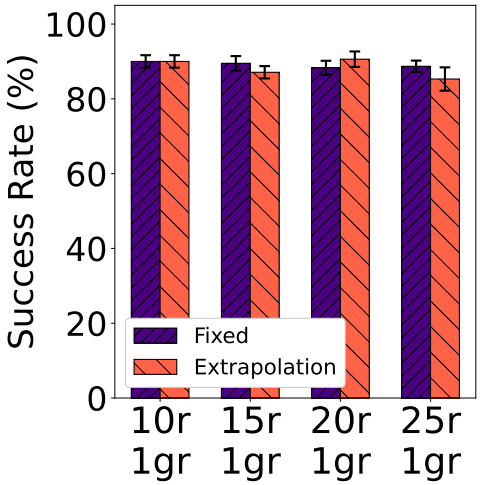
For the *Extrapolation* analysis, it is remarkable that the success rate is equivalent or even better in some cases. In Figures, 3.4(a) and (c), the second grouped bar (10 robots, 1 group & 5 robots, 2 groups) exemplify situations where the *Extrapolation* is better than the *Fixed* setup. Other setups also show similar results, such as the configuration with 5 or 10 robots and 3 groups (Figures 3.4(c) & 3.4(d)).

Moreover, considering the *Extrapolation* results, we understand that increasing the number of robots in the simulation might help the learning robot as there will be more robots in the simulation so it can follow and create a better state representation of the environment. The state representation is based on the sensed region around the learning robot limited by R_2, and the more robot we have in the simulation, the easier to find the way to the goal position using a follower approach.

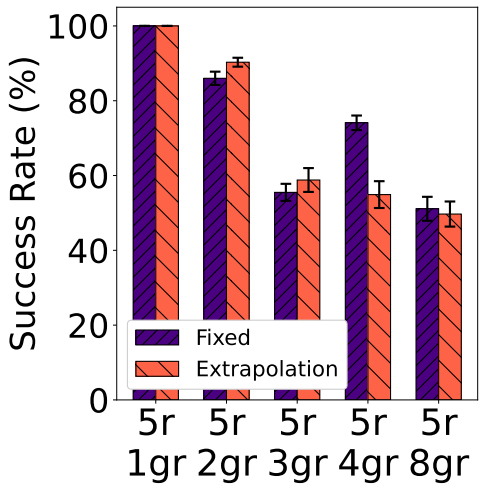
One important point about the *Single Group*, as we only have one group in the simulation, we do not observe the segregation problem or the *Captured Robot* problem. As the problem is due to different groups mixing up during the simulation, with only one group, the learning robot and its group go directly to the goal position. Moreover, the second problem also happens with two or more groups in the simulation. Eventually, the only possible reason to finish the episode and the learning robot receives a bad reward is when it gets lost from its group (*Lost Robot*).



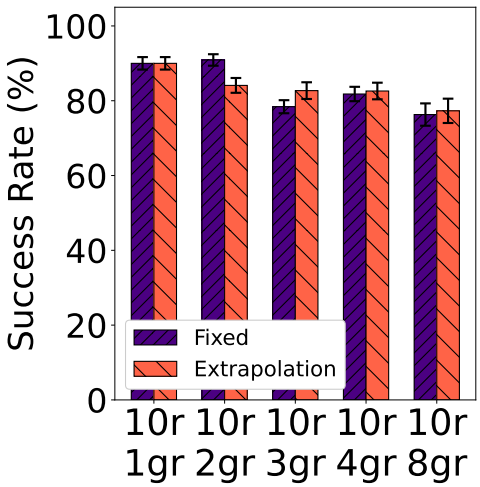
(a) Training setup: 5 robots & 1 Group Extrapolating the number of robots.



(b) Training setup: 10 robots & 1 Group Extrapolating the number of robots.



(c) Training setup: 5 robots & 1 Group Extrapolating the number of groups.



(d) Training setup: 10 robots & 1 Group Extrapolating the number of groups.

Figure 3.4: Results for setup with 5 and 10 robots with 1 group during the training. Figures (a) and (b): increasing the number of robots. Figures (c) and (d): increasing the number of groups.

3.4.2 Multi Group Training

The results presented in Figure 3.5 and Figure 3.6 are focused on setups with multiple groups. While the first figure illustrates the results when increasing the number of robots from 5 to 25 robots per group, the second figure compares the results when increasing the number of groups from 2 to 8.

The same idea for *Fixed* & *Extrapolation* applies to these grouped bars. The **purple** bars represent the *Fixed* results when the training scenario is described by the chart x-axis. The **red** bars are the *Extrapolation* results, and the training was performed using the figure caption scenario and the testing with the x-axis information.

Increasing the complexity in terms of the number of robots shows a decay in the success rate for the *Fixed* strategy (**purple** bars). In Figure 3.5(c), increasing the number of robots reduces the performance from 90% when training with 10 robots and 2 groups to almost 60% when the setup is 25 robots and the same 2 groups. Figures 3.5(b), (c), and (d) also show similar results with a decrease in the success rate when the number of robots is increased. Curiously, 3.5(a), (b), and (d) shows also a common behavior: the first two grouped bars show a lower success rate than the second grouped bar. That means that the success rate in the simplest scenario analyzed is lower than one step more in complexity (increasing the number of robots by 5 units).

We analyzed these behaviors, and increasing the number of robots may also increase the complexity for the learning robot to learn the group behavior. But there are also drawbacks when there are too few robots in the simulation because with fewer robots per group, the state might not be well structured, and the learning robot r_l may be lost or even captured by another group.

When the setup is with 25 robots, the complexity increases, and the robot may not be able to follow the optimal policy towards its goal. With less complexity, but a certain amount of robots to follow, the learning robot r_l achieves a success rate of around 80% on every setup with 10 robots per group.

Figure 3.6 presents similar results: increasing the number of groups also leads to a lower success rate. While Figure 3.6(c) the success rate is at 90% for the setup with 10 robots and 2 groups, increasing the number of groups to 8, the success rate drops to a rate lower than 80%. Also, for Figure 3.6(a) the drop in the success rate is even greater from 90% to 55% for the same difference in the number of groups.

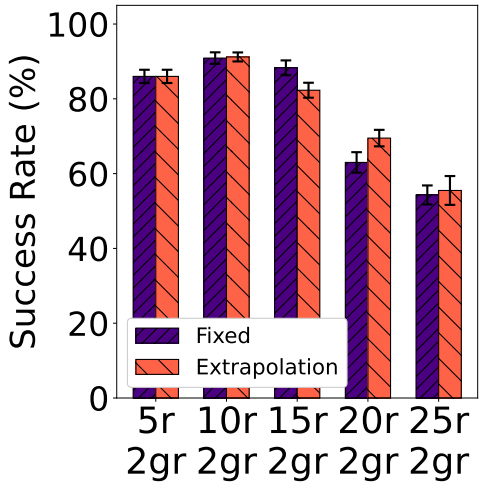
Looking at the *Extrapolation* results, almost every situation outperforms the *Fixed* setup, even with more robots or groups during the testing time. Figures 3.5(a), (c) have an important example when training with 5 and 10 robots with 2 groups and testing in a scenario with 25 robots and the same 2 groups is better when compared with the *Fixed* scenario.

Important results were also observed in Figure 3.6. This scenario presents the results when the training setup was with 10 robots and 2 groups (Figure 3.6(c)) All results shows that the *Extrapolation* results are better than the *Fixed* scenario. It means that even when the training setup was with 10 robots and 2 groups, using the learned policy from 3, 4, or 8 groups, the policy outperforms the *Fixed* training setup with the same 3, 4, or 8 groups. The same behavior may also be observed in Figure 3.6(d) and for some setups in Figures 3.6(a) and (b) the *Extrapolation* (**red** bars) outperform the *Fixed* (**purple** bars).

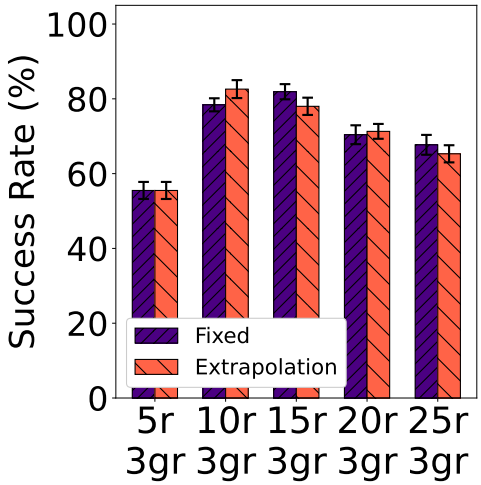
These results are interesting because the complexity and possibility of *Captured Robot* increase with the number of groups in the environment. On the other hand, the success rate with 5 robots presented in Figures 3.6(a) and (b) are worse when compared to Figures 3.6(c) and (d) that present the results with 10 robots. We analyze the results and discuss the idea that increasing the number of robots might help the learning robot build its state representation to follow its group and achieve the expected results.

We use the success rate to evaluate the M_{sensor} methodology, but also streamline the failure causes based on the *Lost* and *Captured* robot (Figure 3.3). As shown in Figure 3.7, the overall failure rate increases with the number of robots (a) and also with the number of groups (b). It is remarkable that increasing the number of robots, the *Captured robot* reason increases while the *Lost robot* decreases from 5 to 10 robots, keeping at a constant level the failure rate from 10 to 15 and then, increases linearly from 15 to 25 robots (Figure 3.7(a)). These results reinforce the idea that with fewer robots, e.g. 5, the learning robot does not have enough information to represent the state using the methodology M_{sensor}, while with a certain number of robots, 10 and 15, this failure reason is minimal and seems to have its optimality at this level.

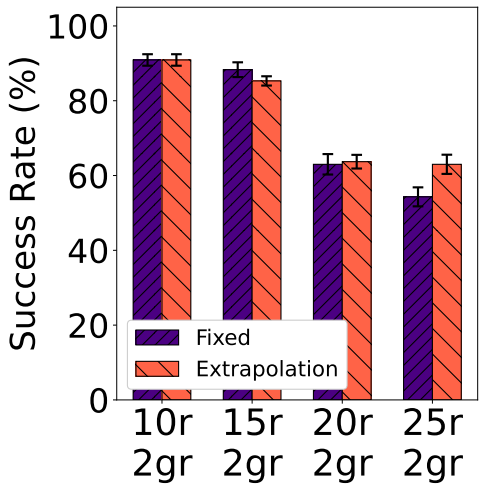
On the other hand, Figure 3.7(b) shows the *Lost* and *Captured* robot when increasing the number of groups. Despite the differences between the results of *Lost* and *Captured*, where most of the time the *Lost robot* is the main reason for failures when we have 8 groups in the simulation, both reasons reach similar failure levels.



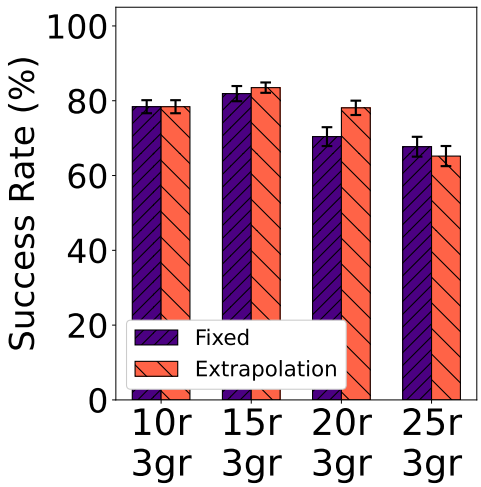
(a) Training setup: 5 robots & 2 Group Extrapolating the number of robots.



(b) Training setup: 5 robots & 3 Group Extrapolating the number of robots.

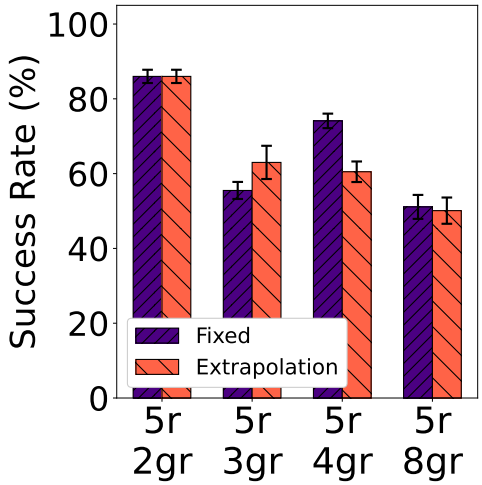


(c) Training setup: 10 robots & 2 Group Extrapolating the number of robots.

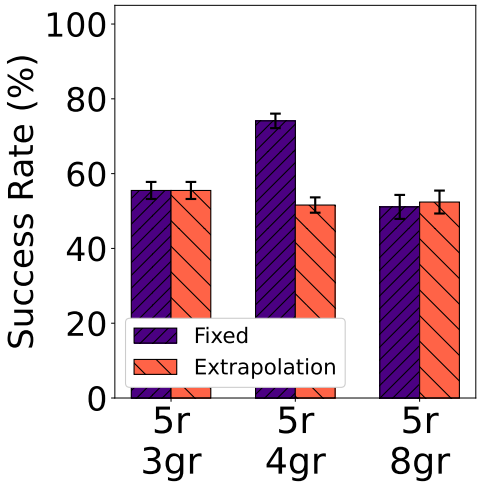


(d) Training setup: 10 robots & 3 Group Extrapolating the number of robots.

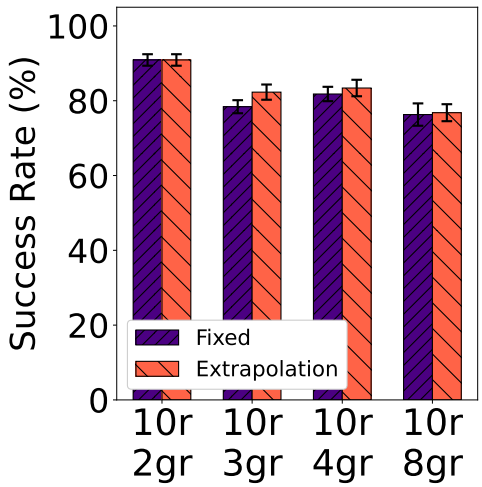
Figure 3.5: Results for setup with 5 and 10 robots with 2 or 3 groups in training time and evaluating the policy for the *Extrapolation* strategy. Here, we demonstrate the *Extrapolation* when increasing the number of robots.



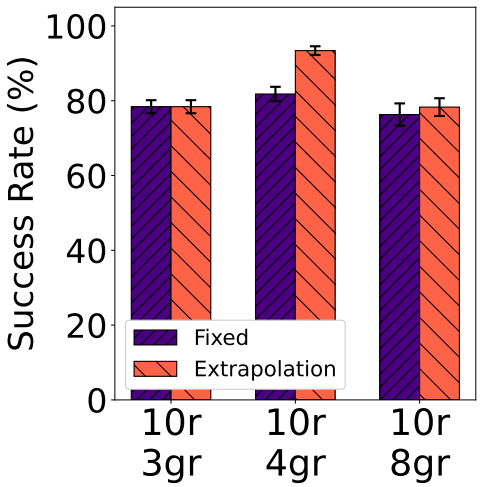
(a) Training setup: 5 robots & 2 Group Extrapolating the number of groups.



(b) Training setup: 5 robots & 3 Group Extrapolating the number of groups.

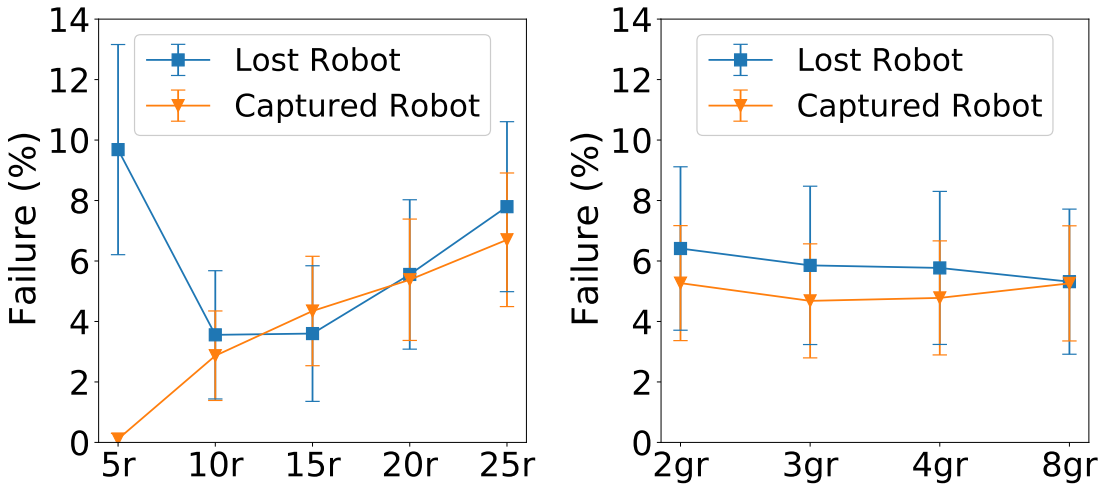


(c) Training setup: 10 robots & 2 Group Extrapolating the number of groups.



(d) Training setup: 10 robots & 3 Group Extrapolating the number of groups.

Figure 3.6: Results for setup with 5 and 10 robots during training time and evaluating the policy for the *Extrapolation* strategy. Here, we demonstrate the *Extrapolation* when increasing the number of Groups.



(a) Increasing number of robots. (b) Increasing number of groups.

Figure 3.7: Analyzing the *Lost* and *Captured* failure rate.

3.5 Conclusion

During the development of the M_{sensor} methodology, discussed in this Chapter, we presented the Markov Decision Process (MDP) using the learning robot r_l sensors limited by R_1 and R_2 that uses the location of the other robots in the neighbor of r_l to create the state vector S.

We conducted several experiments considering a range of robots from 5 to 25 per group and a maximum of 8 groups. During the experiments, we evaluate the learned policy considering a *Fixed* and *Extrapolation* using a simpler training setup during the training, deploying the policy into a more complex setup increasing the number of robots and the number of groups.

The results were interesting, and for a majority of the setups, our methodology had a reasonable success rate in the *Fixed* evaluation, but also when compared to the *Extrapolation* scenario. Additionally, we demonstrate the main reasons for failure and how they compare to the simulation. The *Lost* and *Captured* robot are the two causes of failure during the experiments, and increasing the number of robots from 10 to 25, for example, was notable the increase in both failure reasons.

Finally, besides being simple, our methodology depends on the FL-ORCA algorithm as the learning robot r_l is known by the other robots that are ruled by the FL-ORCA as a member of their group. So, part of the responsibility for not colliding is of the other robots in the simulation and this constraint is notable we propose in the next Chapter some possible approaches to eliminate this constraint.

Chapter 4

Ellipse Representation

In this Chapter, we try to overcome the difficulties and enhance the results presented in Chapter 3. As we have seen in the first methodology, the robot can learn the group behavior ruled by the FL-ORCA, and the success rate is satisfactory. Furthermore, the M_{sensor} presents a constraint that the learning robot r_l is part of the FL-ORCA group and is known as a member of its group. In other words, during the execution, the robots in Chapter 3 recognize the learning robot r_l as a member in the FL-ORCA execution, and they will take into account the r_l position and velocity to perform the safe navigation and the congestion part of the simulation, while in this chapter we remove this constraint.

Here, we investigate a different approach using an ellipse to model or encapsulate each group $\Gamma_k \in \Gamma$. The inspiration to use this abstraction comes from the authors in [65], where they use an *Hierarchical Abstraction (HAS)* to model the swarm based on statistical models.

As the learning robot r_l is no longer a member of the FL-ORCA, and all robots in the simulation do not recognize the r_l as part of their group or the simulation, we address the collision problem using Artificial Potential Fields (APF) [44]. Then, we split the ellipse approach into two different methodologies called $M_{INforce}$ and $M_{OUTforce}$.

The first approach considers the APF as an input in the state representation of the environment and is used in the Neural Network (DQN algorithm), while in the second approach, the APF is used as a vector to be summed up to the Neural Network velocity vector and the $robot_l$ executed velocity will be a vector sum of these two velocities.

As a definition, we describe the two approaches:

1. We compute the APF velocity u_{apf} and use it as an input to the state vector $S_{M_{in}}$. The two additional parameters represent the velocity in the x and y directions. For simplicity, we name this method as $M_{INforce}$.

2. We compute the APF velocity vector u_{apf} and sum it up to the Neural Network velocity vector u_{nn}. As a result, we have a final velocity u_l applied to the robot. In this case, we have a state vector $S_{M_{out}}$, and for simplicity, we name this method as $M_{OUTforce}$.

We also compare the results from the first methodology presented in Chapter 3 here as M_{sensor}.

4.1 Task Formulation

The task to be achieved in this chapter is very similar to the task presented in 3. Here, we also consider a set of $\mathcal{S} = \{r_1, r_2, ..., r_n\}$ robots that share and navigate in an environment. The learning robot is also defined as r_l where $\{l \in \mathbb{N} \mid 1 \leq l \leq n\}$ and the group to which r_l belongs is $\Gamma_d = \{d \in \mathbb{N} \mid 1 \leq d \leq m\}$.

The aforementioned group Γ_d has the learning robot r_l ($r_l \in \Gamma_d$), but the other robots from the same group Γ_d do not sense r_l as a member of their FL-ORCA defined as Υ_d ($r_l \notin \Upsilon_d$).

Finally, the problem addressed is the same as defined in section 3.1, where we want a robot to learn to behave as part of a heterogeneous swarm formed by an arbitrary number of robots that navigate toward a goal in a shared environment while maintaining the condition of segregation between the swarm groups.

We use a different group abstraction where we model each group as an ellipse that tries to cover all robots within a group. During the task, the robot must reach the goal position and keep its position inside the given ellipse. We better describe this formulation in the next Section 4.2.3.

4.2 Markov Decision Process Formulation

The MDP formulation of this approach follows a similar idea presented in chapter 3. Section 3.2.3 presents the previous MDP formulation where consider a local perception and the robot sense is represented by a circular area with an external ring divided into sectors. We use the same perception model as described in Figure 3.1 to sense the environment that surrounds the learning robot r_l and evaluate the APF considering the other robots in the neighborhood.

Additionally, to compute the ellipse of each group, we assume that every robot of a given group knows the other robot's position to build the ellipse representation. That said, in $M_{INforce}$ we use the APF velocity and the group ellipse from all robot positions

from the same group to build the learning robot's current state. On the other hand, $M_{OUTforce}$ the group ellipse from all robot positions from the same group is used to build the state representation, and APF velocity is not considered to compute the state.

4.2.1 *State Space*

Chapter 3 presents a state space based on the local sensing of the robot. By contrast, this formulation uses ellipses to model each group $\Gamma_k \in \Gamma$. At each time-step, we compute all robot's poses in the simulation $q_n = (x_n, y_n) \, \forall \, r_i \in \mathcal{S}$ to assign to each group a matrix Z to calculate all ellipses parameters. The general ellipse equation is given by Equation 4.1 where x_c and y_c represent the ellipse's center and x and y a point in a 2D space. This Equation is a restriction to evaluate if a given robot is inside the ellipse, and this constraint should be met to assign the robot to that specific ellipse and to set the learning robot reward.

$$\frac{(x - x_c)^2}{a^2} + \frac{(y - y_c)^2}{b^2} \leq 1 \tag{4.1}$$

Figure 4.1 demonstrates an ellipse example with all robots from group Γ_k and the ellipse's parameters a, b and θ.

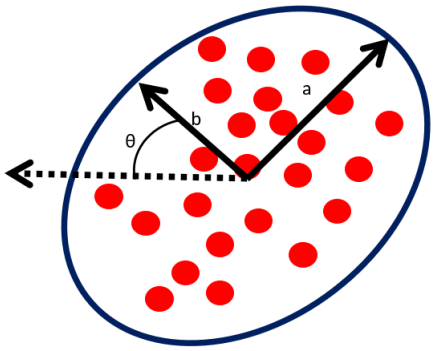


Figure 4.1: Ellipse representation given the parameters a, b and θ for all robots (red) in group $\Gamma_k \in \Gamma$.

During the simulation, the group behavior dynamically changes due to the FL-ORCA algorithm, the robots avoid collisions and overcome the congestion while achieving their goal. Then, the ellipse that encapsulates the group might rotate over an angle θ. Thus, we use a transformed version of the Equation 4.1 given by the Equation.

$$\frac{((x-x_c)\times\cos\theta-(y-y_c)\times\sin\theta)^2}{a^2}+\frac{((x-x_c)\times\sin\theta+(y-y_c)\times\cos\theta))^2}{b^2}\leq 1 \quad (4.2)$$

At each time step, we assign all robots to a matrix Z representing the robots pose q and the group $\Gamma_k \in \Gamma$ a given robot $r_b$elongs.

$$Z=\begin{pmatrix} r_{1x} & r_{1_y} & \Gamma_k\in\Gamma \\ r_{2x} & r_{2y} & \Gamma_k\in\Gamma \\ \vdots & \vdots & \vdots \\ r_{nx} & r_{ny} & \Gamma_k\in\Gamma \end{pmatrix}$$

Then, for each group Γ_k we assume that the positions are a sample from a 2D Gaussian Distribution, and we calculate the Covariance Matrix C considering that the positions x and y are uncorrelated:

$$C=\begin{pmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{pmatrix}$$

The *eigenvectors* of C $(\overrightarrow{e1}, \overrightarrow{e2})$ point towards the largest spread of the data in the original matrix Z and the *eigenvalues* represents the variance along those directions. To get a certain confidence interval, we can multiply the normalized axes by a constant c, resulting in a confidence interval from a cumulative distribution function χ^2 (CDF) with 2 degrees of freedom.

Therefore, the parameters of the ellipse a and b may be expressed in Equations 4.3 and 4.4 while the ellipse center (x_c, y_c) are defined by Equations 4.5 and 4.6 where n is the number of robots in each group $\Gamma_k \in \Gamma$. Finally, the last ellipse parameter θ is defined as the angle between the largest *eigenvector* $\overrightarrow{e1}$ and the x-axis as Equation 4.7.

$$a=\frac{\sqrt{c\|\overrightarrow{e2}\|}}{2} \quad (4.3)$$

$$b=\frac{\sqrt{c\|\overrightarrow{e1}\|}}{2} \quad (4.4)$$

$$x_c=\frac{1}{n}\sum_{i=1}^{n} r_{ix}=\frac{r_{1x}+r_{2x}+\cdots+r_{nx}}{n} \quad (4.5)$$

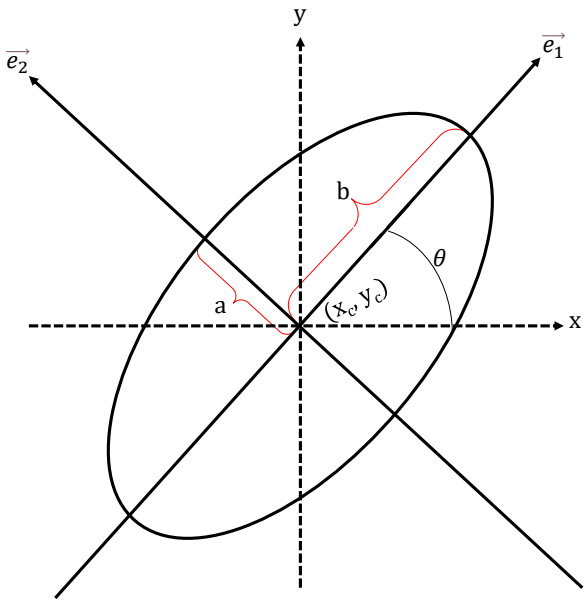


Figure 4.2: Example of an ellipse and its parameters when we model the coverage for all robots that belong to a group at each state.

$$y_c = \frac{1}{n}\sum_{i=1}^{n} r_{iy} = \frac{r_{1y} + r_{2y} + \cdots + r_{ny}}{n} \tag{4.6}$$

$$\theta = \frac{\overrightarrow{e1}}{\overrightarrow{e2}} \tag{4.7}$$

We use a state space representation for the proposed methodology as a matrix $S_{M_{in}}$, and it includes the APF velocity. The last two arguments are the calculated APF velocity in the x and y directions while the other parameters are the ellipse parameters evaluated using the presented equations. Each line of the given matrix S has tge ellipse parameters for all m groups.

$$S_{M_{in}} = \begin{pmatrix} a_1 & b_1 & x_{c1} & y_{c1} & \theta_1 & x_{g1} & y_{g1} & u_{apf1_x} & u_{apf1_y} \\ a_2 & b_2 & x_{c2} & y_{c2} & \theta_2 & x_{g2} & y_{g2} & u_{apf2_x} & u_{apf2_y} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_m & b_m & x_{cm} & y_{cm} & \theta_m & x_{gm} & y_{gm} & u_{apfm_x} & u_{apfm_y} \end{pmatrix}$$

Hence, we represent the state space as a new matrix $S_{M_{out}}$ where each line of the matrix S has ellipse parameters that aim to model all groups $\Gamma_k \in \Gamma$, and the two last arguments are the goal position. Again, we consider a terminal state when the learning

robot reaches the goal position $goal_d = [x_d, y_d]$ with its group, which means that all robots from Γ_d, including r_l, have the average distance $\delta_{X,goal}$ to the goal position less than R_2 (Equation 3.2).

$$S_{M_{out}} = \begin{pmatrix} a_1 & b_1 & x_{c1} & y_{c1} & \theta_1 & x_{g1} & y_{g1} \\ a_2 & b_2 & x_{c2} & y_{c2} & \theta_2 & x_{g2} & y_{g2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_m & b_m & x_{cm} & y_{cm} & \theta_m & x_{gm} & y_{gm} \end{pmatrix}$$

A terminal state is reached when r_l does not satisfies the Equation 4.2 for φ consecutive time-steps, when it reaches the goal position or, when it collides against another robot in the environment. In other words, if the learning robot r_l stays outside its group ellipse for φ consecutive time-steps, it is considered a terminal state, and the episode is finished. Likewise, if the robot reaches its goal with the group, a terminal state is also reached.

4.2.2 *Action Space*

The action space for this approach considers a set of allowed directions in a continuous space as established in section 3.2.3. We choose the same set of actions and are given an action $\{a \in A \mid 1 \leq a \leq q+1\}$ we calculate an angle α_i towards the respective region.

For the state space $S_{M_{out}}$, we use Equation 3.4 to compute the Neural Network velocity u_{nn} output. We use a second velocity u_{apf} that aims to avoid collisions using the potential fields. Finally, the executed velocity is a vector sum of the two velocities $u_l = u_{nn} + u_{apf}$.

As an example of how these velocities are set and how the action space actually works in our method, Figure 4.3 demonstrates a single step where a Neural Network velocity and an APF velocity are summed up and final velocity u_l is set.

One may note that the action that is the output from the Neural Network is the u_{nn}. As our state space only considers the ellipses and goal parameters, it is important to add the APF velocity as a collision avoidance method. Otherwise, the success rate would be close to zero, and the goal would be never reached.

On the other hand, for the state space $S_{M_{in}}$ instead of summing up the velocities as demonstrated in Figure 4.3, we include the u_{apf} as part of the state representation and

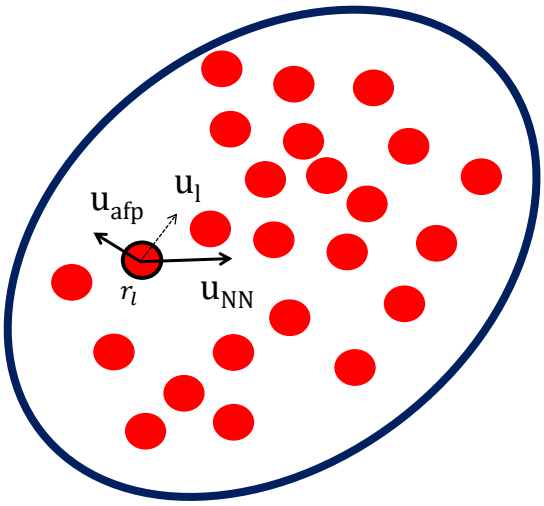


Figure 4.3: Example of u_l as a sum of u_{afp} and u_{nn}.

expect that the Neural Network maps the actions to avoid the collisions while the group target is achieved.

4.2.3 *Reward Design*

The reward designed for the present approach is different from Chapter 3. The target position is represented inside the state $S_{M_{in}}$ and $S_{M_{out}}$ as x_g and y_g, and it represents a long-term achievement, but the short-term goal at every time step is to keep the learning robot inside the group ellipse to keep the group cohesion. Hence, we set a terminal reward $+\tau$ when the learning robot reaches the goal alongside its group Γ_d and a $-\tau$ when a collision between r_l and any other robot or when r_l does not satisfies the Equation 4.2 for φ consecutive time-steps. Likewise, a reward $+\frac{\tau}{100}$ is assign to it if satisfies Equation 4.2 (inside group ellipse) and $-\frac{\tau}{100}$ otherwise. We summarize the reward design in Table 4.1.

State	Reward
Inside the Ellipse	$+\tau/100$
Outside the Ellipse	$-\tau/100$
Lost / Collision	$-\tau$
Outside Ellipse for φ steps	$-\tau$
Goal reached	$+\tau$

Table 4.1: Reward Design for All Possible States Considering the $S_{M_{in}}$ and $S_{M_{out}}$ Representation.

4.3 Experimental Setup

The experiments in this Chapter focus on accomplishing the task rather than extrapolating more difficult scenarios as presented in Chapter 3. We want here to compare both $M_{INforce}$ & $M_{OUTforce}$ with the first methodology Sensor State Representation. For simplicity, the approach described in Chapter 3 is called here M_{sensor}

Similarly to the Experimental Setup in Chapter 3, we use a different number of robots $(5, 10, 15, 20, 25)$ and groups $(1, 2, 3, 4, 8)$ when training. We start each training with r_l placed in a random group Γ_k and randomly selected among the robots of Γ_k. With this random selection, we expect a better exploration among all possible start positions and want the robot to experience different difficulty levels due to its start position inside the group.

Algorithm 4.1 presents the pseudocode for the $M_{INforce}$ approach. Again, at every experiment, we initialize the simulation with a set of robots $\mathcal{S}$ and a set of groups Γ and all necessary parameters (Table 4.2) where λ is the discount factor used to update the action-value function Q according to [58]. We use the ε-greedy strategy for balancing between exploration and exploitation. Aside from the Reinforcement Learning parameters, we also set some simulation parameters that the learning robot uses. The η parameter sets the maximum number of steps each episode might have, the v parameter is used in the Equation 3.4, and the R_1 & R_2 are the sensor radius used by the r_l. The τ is set to 1. Finally, we use a cumulative distribution function χ^2 to set a confidence interval on the ellipse parameters.

One important step of Algorithm 4.1 is that it calculates u_{apf} when sensing the environment to set as one of the $S_{M_{in}}$ inputs. Our objective using this information is that the action-value function Q will be able to generalize those states and map the input to reliable actions that support the learning robot to accomplish its task.

On the other hand, Algorithm 4.2 shows the pseudocode for the $M_{OUTforce}$ method.

Parameter	Value	Unit
C	5.991	-
χ^2	90	%
λ	0.001	-
ϵ	0.05	-
η	500	-
v	1	m/s
τ	1	-
R_1	0.5	m
R_2	1.5	m

Table 4.2: Parameters used during training for $M_{INforce}$ & $M_{OUTforce}$.

The main difference here from algorithm 4.1 is the u_l computation which uses u_{nn} and u_{apf} as a vector sum instead of using the artificial potential fields velocity as input in $S_{M_{out}}$. We create here a combined method where the action-value function Q is responsible for keeping the learning robot bound to its group keeping the cohesion even when the congestion part happens, while the u_{apf} avoids the collisions between the learning robot and the other robots in the simulation that do not sense the learning robot.

We conducted the experiments simulating the methodologies $M_{OUTforce}$ & $M_{INforce}$. On each combination setup, the number of robots and groups were evaluated for 30 interactions. Each interaction was evaluated 100 times for statistical measures, thus a total of 3000 experiments for each setup. Following a similar idea of Chapter 3, in each process, we compute the success rate of the given setup.

4.4 Results

The graphs presented here use the acronyms of the names in Chapter 3. A sample of the executions for this representation can be found Ellipses Representation Videos.

The "r" represents the number of robots, and the "gr" the number of groups. The grouped bar charts show the results for the three methodologies presented in this dissertation:

1. M_{sensor} represents the results of Chapter 3. As one of the objectives of Chapter 4 is to improve some points of the previous methodology, it is important to compare

all results.

2. $M_{INforce}$ represents the results when u_{apf} is used as a part of the state space $S_{M_{in}}$ and is an input to the Neural Network.

3. Finally, $M_{OUTforce}$ represents the results when the learning robot velocity is computed as $u_l = u_{nn}+u_{apf}$. It is related to the state space $S_{M_{out}}$.

Figure 4.4 demonstrates the results for all setups. Every plot is related to the number of robots used (from 5 to 25 robots), and the three grouped bars represent the comparison among the methodologies.

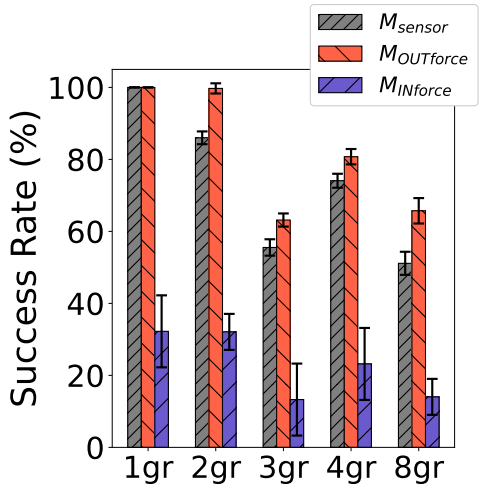
For all situations, the methodology $M_{OUTforce}$ outperforms all others, achieving interesting results. On the other hand, $M_{INforce}$ demonstrates a poor performance with a success rate lower than 50% in all cases.

In Figures, 4.4(a), (d), and (e), one can note that the setup with 1 group shows a success rate close to 100% for $M_{OUTforce}$, but this metric decays fast as the number of groups increases. This indicates that the learning robot r_l can use the state representation to follow its group without any prior information of the other robot's behavior, but the increase in the complexity with more groups (not necessarily adversaries, but they compete for space during the simulation) the success rate decreases.

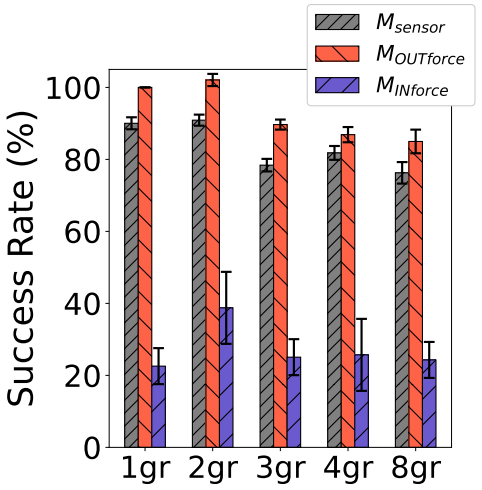
The increase in the number of groups also increases the difficulty, and in the first case (5 robots), the learning robot gets lost during the testing time. When the setup is 20 or 25 robots per group in the simulation, the complexity is due to the total number of robots, especially in the center of the simulation frame, when a situation called congestion happens and may be defined as the increase of the number of robots in the same area of the simulation. The robots should be able to manage this situation and overcome the congestion issue. In those situations, the learning robot may be suffering from the *Captured Robot* reason instead of being lost.

In contrast, Figures 4.4(b) and (c) for $M_{OUTforce}$ show the decay is smother, and the success rate is above 80%, except for the setup with 15 robots and 8 groups. The same behavior is noted for M_{sensor} results. Nevertheless, $M_{INforce}$ still demonstrates a bad performance in all situations. The reason $M_{INforce}$ presents a worse success rate compared to the other methods may be the neural network not learning how to deal with the collisions. Even with the u_{apf} being input in the state representation, the Neural Network cannot generalize and avoid collisions. By contrast, the $M_{OUTforce}$ approach uses u_{apf} in a sum with the u_{nn}, which is more reliable when dealing with the collision avoidance task.

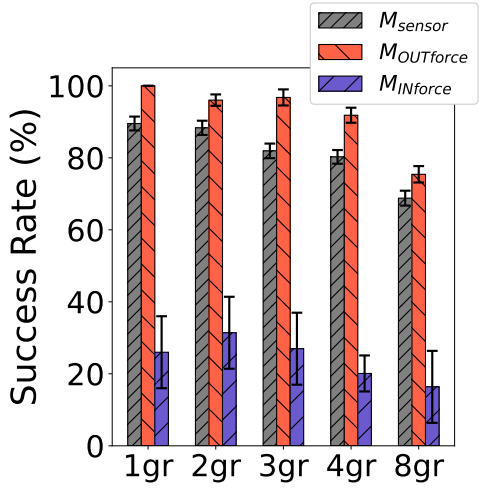
The Captured and Lost robot situations show similar behavior for both methodologies ($M_{OUTforce}$ & M_{sensor}). While a setup with 5 robots in each group, the *Captured Robot* failure does not happen, as the number of robots increases in the simulation, the



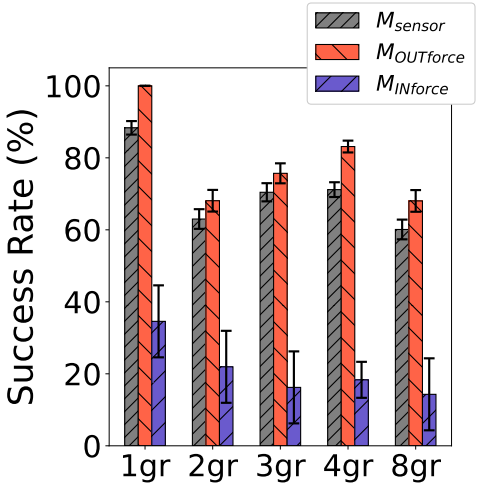
(a) 5 Robots.



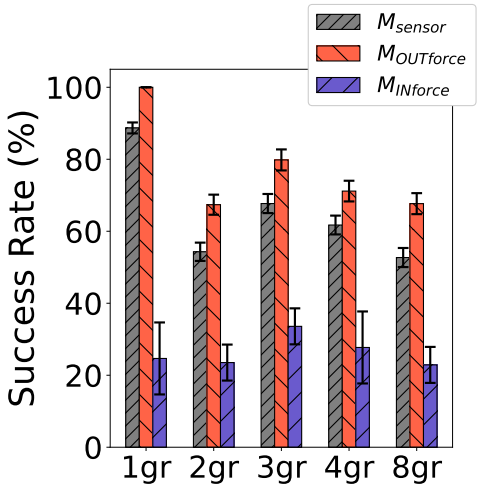
(b) 10 Robots.



(c) 15 Robots.



(d) 20 Robots.



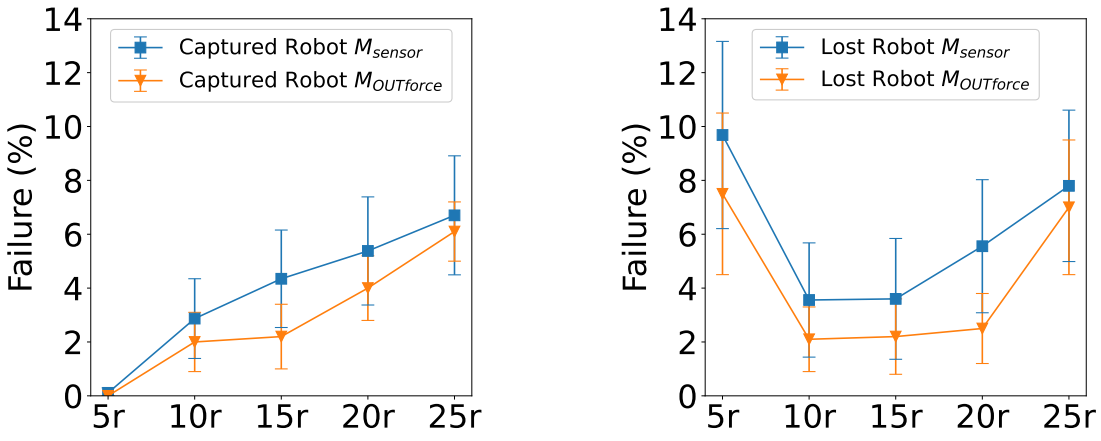
(e) 25 Robots.

Figure 4.4: Results for all methodologies: M_{sensor}, $M_{INforce}$ & $M_{OUTforce}$. Every chart represents a setup with a specific number of robots, and each grouped bar is the number of groups.

percentage of failure due to this reason also increases (Figure 4.5). For the M_{sensor} and $M_{OUTforce}$ methods, it reaches 6% of failure, which can be explained mainly in the congestion part of the simulation (e.g. Figure 3.2 steps 160 and 240).

On the other hand, the *Lost Robot* failure percentage decreases from 5 to 10 robots. As we have more robots to guide the r_l during the task, the failure due to the learning robot being lost is reduced. Also, from 10 to 15 robots per group, the *Lost Robot* failure percentage keeps at the same level, but the failure percentage increases when observing the number of robots per group from 20 to 25. It seems here that there is an optimal level to reduce the Lost Robot situation with 10 to 15 robots, and going beyond that, the number of robots reduces the success rate. Increasing the complexity of the simulation with more robots per group also increases the failure rate.

In addition to the previous analysis, we can compare both methods and exemplify why the $M_{OUTforce}$ demonstrates a higher success rate than M_{sensor}. The trending behavior of the *Captured* and *Lost* robot is similar for both cases, but for all results presented in Figure 4.5, the $M_{OUTforce}$ presents a lower failure rate, and as a consequence a higher success rate. In Figure 4.5(a), increasing the number of robots from 10 to 15 does not imply a higher *Captured* failure reason for $M_{OUTforce}$. The same behavior is presented in Figure 4.5(b); when we increase the number of robots from 10 to 20, the *Lost* robot reason does not increase significantly for $M_{OUTforce}$ while the M_{sensor} approach increases from 4% to 8% the failure rate. Additionally, as the r_l is not considered by all other robots in the simulation as an FL-ORCA member, the robots do not change their path to avoid collisions with the robot. Instead of that, the learning robot using u_l and the component u_{apf} is responsible for avoiding collisions and also achieving the group goal.



(a) Captured Robot failure results.

(b) Lost Robot failure results.

Figure 4.5: M_{sensor} & $M_{OUTforce}$ failure percentage when increasing the number of robots per group. They present a similar behavior, but $M_{OUTforce}$ demonstrates better performance. Here we do not present the $M_{INforce}$ as its results are considerably worse than both presented.

Despite the FL-ORCA constraint that highlights the better results in $M_{OUTforce}$, the r_l has to know the ellipse parameter from its group. Even if the other robots do not modify their trajectories to avoid collisions, the r_l still needs to know all its group members position to evaluate all necessary parameters to build the state representation.

4.5 Conclusion

During the development of Chapter 4 we presented two methodologies to build the state space: $M_{INforce}$ & $M_{OUTforce}$. While the first use the APF velocity u_{apf} as an input in the Neural Network, in the second methodology this velocity is summed up to the Neural Network velocity $u_l = u_{nn} + u_{apf}$ to avoid collisions among the robots.

The experiments were conducted using the range of robots per group from 5 to 25 while the number of groups was from 1 (single group) to 8 groups. As one of the objectives of this chapter is to improve the performance from the previous chapter, we compare $M_{INforce}$ and $M_{OUTforce}$ with the M_{sensor} approach using the success rate as a metric of comparison.

We observed that $M_{INforce}$ demonstrated poor performance in terms of success rate compared to the $M_{OUTforce}$ and one of the main reasons is when using the u_{apf} as an input to the Neural Network, it is not able to avoid collisions and the main failure reason is the collisions among the robots. On the other hand, the $M_{OUTforce}$ uses $u_l = u_{nn} + u_{apf}$ as shown in Figure 4.3 and its results outperform the first methodology, M_{sensor}, for all experiments.

Finally, besides eliminating the FL-ORCA constraint, and the learning robot is no longer part of the FL-ORCA group, another constrain was inserted to compute $S_{M_{out}}$: The robots from the same group should know the position of each robot to compute the ellipse's parameters and build $S_{M_{out}}$.

Algorithm 5: $M_{INforce}$

Inputs: Set of Robots $\mathcal{S}$, Set of Groups Γ
Parameters: $\lambda, \epsilon, \tau, \eta, \upsilon$,$\mathrm{r}_1$, r_2, C, χ^2
Initialize action-value function Q with random weights
Set the initial state at time step 0

for *Each episode* **do**
  Choose a new learning robot $r_l \in \mathcal{S}$;
  for *Each step of the current episode* **do**
    Sense the environment calculating u_{apf} from the learning robot sensor.
    Set the state space $S_{M_{in}}$ using C from the cumulative distribution function χ^2.
    Calculate ellipse parameters using Equations 4.3, 4.4, 4.5, 4.6, 4.7.
    Choose $\alpha \in A$ given $S_{M_{in}}$ using policy derived from Q using ε-greedy;
    Calculate u_l using Equation 3.3 given α
    Execute u_l, observe $S_{M'_{in}}$;
    if $S_{M'_{in}}$ *is terminal or current step* $\geq \eta$ **then**
      Receive final reward τ *or* $-\tau$
      Stop the current episode;
    else
      Receive reward $\tau/100$ *or* $-\tau/100$*;*
    end if
    Update the action-value function Q *according to Algorithm 1*
  end for
end for

Algorithm 4.1: $M_{INforce}$ Algorithm.

Algorithm 6: $M_{OUTforce}$

Inputs: Set of Robots $\mathcal{S}$, Set of Groups Γ
Parameters: $\lambda, \epsilon, \tau, \eta, v$,r$_1$, C, χ^2
Initialize action-value function Q with random weights
Set the initial state at time step 0

for *Each episode* **do**
  Choose a new learning robot $r_l \in \mathcal{S}$;
  for *Each step of the current episode* **do**
    Sense the environment and set the state space $S_{M_{out}}$ using C from the cumulative distribution function χ^2.
    Calculate ellipse parameters using Equations 4.3, 4.4, 4.5, 4.6, 4.7.
    Choose $\alpha \in A$ given $S_{M_{out}}$ using policy derived from Q using ε-greedy;
    Calculate u_{apf} using R_1 from learning robot sensor
    Calculate $u_l = u_{nn} + u_{apf}$.
    Execute u_l and observe $S_{M'_{out}}$
    if $S_{M'_{out}}$ *is terminal or current step* $\geq \eta$ **then**
      Receive final reward τ *or* $-\tau$
      Stop the current episode;
    else
      Receive reward $\tau/100$ *or* $-\tau/100$*;*
    end if
    Update the action-value function Q *according to Algorithm 1*
  end for
end for

Algorithm 4.2: $M_{OUTforce}$ Algorithm.

Chapter 5

Conclusions and Future Work

During the development of this work, several methods were tested to investigate algorithms and representations that best suit the segregated navigation task using a Reinforcement Learning approach to a swarm of robots. In Chapter 3, we used a state representation that uses the robot sensors to compute a state representation of the environment with the regions surrounding the learning robot. The experiments in this approach considered two situations: *Fixed* and *Extrapolation*. The *Fixed* method was simply training and testing with the same setup configuration, while the *Extrapolation* considers a training scenario with fewer robots per group than in the testing scenario. For example, in the *Extrapolation*, we trained a setup with 5 robots per group and tested in a scenario with 20 robots per group. The results demonstrated that at some level, it is possible to use the proposed method M_{sensor} to *Extrapolate* simple training setups to a more complex without vanishing the success rate.

One disadvantage of the proposed methodology in Chapter 3 is that the learning robot r_l is known by the other robots as members of the FL-ORCA swarm. The consequence is that r_l has only half the responsibility for not colliding, as the other robot in the environment will avoid collisions with the swarm members.

In order to address this constraint, we use a different methodology in Chapter 4. We proposed $M_{INforce}$ and $M_{OUTforce}$, where the state representation is an ellipse abstraction to model or encapsulate the group of robots inside an ellipse. The $M_{INforce}$ sets the state representation as a matrix $S_{M_{in}}$ where the u_{apf} (Artificial Potential Fields velocity) is set as an input to the Neural Network. On the other hand, $M_{OUTforce}$ uses the matrix $S_{M_{out}}$ as a state representation, and for this case, the u_{apf} is used in a vector sum of the Neural Network velocity and the Potential Fields velocity.

The results from both methodologies diverge considerably since the $M_{INforce}$ cannot generalize the problem, and the success rate is significantly worse than the $M_{OUTforce}$ results. We also compare the $M_{OUTforce}$ and M_{sensor} methodologies as their results are interesting in terms of success rate. One can observe that the $M_{OUTforce}$ outperforms all scenarios compared to M_{sensor}. When evaluating the failure reasons, we also observe that the *Lost* and *Captured* robot situation is lower for the $M_{OUTforce}$ approach.

In conclusion to this work, both methodologies, $M_{OUTforce}$, and M_{sensor}, presented

important results demonstrating their potential in using Reinforcement Learning to use a learning robot without prior knowledge of the environment or the other robots. Each method has its own constraints and may be addressed or evaluated critically and deployed to a real robot evaluation.

Although we have improved the results in the $M_{OUTforce}$ methodology, several future directions may be possible in future research:

- In methods $M_{INforce}$ and $M_{OUTforce}$, the ellipse evaluation considers that the learning robot has knowledge of the parameters of the ellipse. Although broadcast communication might fill this gap, the data latency and even data loss may reduce the success rate due to a lack of precision. We may understand the impact of noisy data during the parameter recognition, for example, dropping the position for one or many robots from the r_l group.

- Testing the strategy on real robots to investigate possible gaps in the methodologies is also a possible future work. We have developed a simulation environment that emulates some real situations, but those methods can be extended in a more complex simulator like Gazebo.

References

[1] Stefano V. Albrecht and Subramanian Ramamoorthy. A game-theoretic model and best-response learning method for ad hoc coordination in multiagent systems. *CoRR*, 2015.

[2] Stefano V. Albrecht and Subramanian Ramamoorthy. Are you doing what I think you are doing? criticising uncertain agent models. *CoRR*, 2019.

[3] Stefano V. Albrecht and Peter Stone. Autonomous agents modelling other agents: A comprehensive survey and open problems. *Artificial Intelligence*, 258:66–95, 2018.

[4] Javier Alonso-Mora, Andreas Breitenmoser, Martin Rufli, Paul Beardsley, and Roland Siegwart. *Optimal Reciprocal Collision Avoidance for Multiple Non-Holonomic Robots*, pages 203–216. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[5] Stefanie Angerer, Christoph Strassmair, Max Staehr, Maren Roettenbacher, and Neil M. Robertson. Give me a hand — the potential of mobile assistive robots in automotive logistics and assembly applications. In *2012 IEEE International Conference on Technologies for Practical Robot Applications (TePRA)*, pages 111–116, 2012.

[6] Ross Arnold, Kevin Carey, Benjamin Abruzzo, and Christopher Korpela. What is a robot swarm: A definition for swarming robotics. In *2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, pages 0074–0081, 2019.

[7] Farshad Arvin, Khairulmizam Samsudin, Abdul Rahman Ramli, and Masoud Bekravi. Imitation of honeybee aggregation with collective behavior of swarm robots. *International Journal of Computational Intelligence Systems*, 4(4):739–748, 2011.

[8] Farshad Arvin, Ali Emre Turgut, Farhad Bazyari, Kutluk Bilge Arikan, Nicola Bellotto, and Shigang Yue. Cue-based aggregation with a mobile robot swarm: a novel fuzzy-based method. *Adaptive Behavior*, 22(3):189–206, 2014.

[9] Gianluca Baldassarre, Stefano Nolfi, and Domenico Parisi. Evolving mobile robots able to display collective behaviors. *Artificial life*, 9(3):255–267, 2003.

[10] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846, 1983.

[11] Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.

[12] Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[13] Calin Belta, Guilherme AS Pereira, and Vijay Kumar. Abstraction and control for swarms of robots. In *Robotics Research. The Eleventh International Symposium*, pages 224–233. Springer, 2005.

[14] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48, New York, NY, USA, 2009. Association for Computing Machinery.

[15] Jur van den Berg, Stephen J Guy, Ming Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. In *Robotics research*, pages 3–19. Springer, 2011.

[16] Bruno Brito, Michael Everett, Jonathan P. How, and Javier Alonso-Mora. Where to go next: Learning a subgoal recommendation policy for navigation in dynamic environments. *IEEE Robotics and Automation Letters*, 6(3):4616–4623, 2021.

[17] Alejandro Camacho, Rodrigo Toro Icarte, Toryn Q Klassen, and Roberto Valenzano. Ltl and beyond: Formal languages for reward function specification in reinforcement learning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 840–846. IJCAI, 2019.

[18] Gustavo A Cardona and Juan M Calderon. Robot swarm navigation and victim detection using rendezvous consensus in search and rescue operations. *Applied Sciences*, 9(8):1702, 2019.

[19] Yu Fan Chen, Miao Liu, Michael Everett, and Jonathan P How. Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 285–292. IEEE, 2017.

[20] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[21] Rati Devidze, Goran Radanovic, Yuan Sun, Kumar Sricharan, Xiaojin Zhu, Jianyu Chen, and Carl Woodside. Explicable reward design for reinforcement learning agents. In *Advances in Neural Information Processing Systems*, 2021.

[22] Daniel Dewey. Reinforcement learning and the reward engineering principle. In *2014 AAAI Spring Symposium Series*, 2014.

[23] Gregory Dudek, Michael RM Jenkin, Evangelos Milios, and David Wilkes. A taxonomy for multi-agent robotics. *Autonomous Robots*, 3(4):375–397, 1996.

[24] Razvan C Fetecau and Justin Meskas. A nonlocal kinetic model for predator–prey interactions. *Swarm Intelligence*, 7(4):279–305, 2013.

[25] Edson B. F. Filho and Luciano C. A. Pimenta. Segregating multiple groups of heterogeneous units in robot swarms using abstractions. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 401–406, 2015.

[26] Paolo Fiorini and Zvi Shiller. Motion planning in dynamic environments using velocity obstacles. *The international journal of robotics research*, 17(7):760–772, 1998.

[27] Antonio Franchi, Paolo Stegagno, and A. Franchi. Decentralized multi-robot encirclement of a 3d target with guaranteed collision avoidance.

[28] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. *An Introduction to Deep Reinforcement Learning*, volume 11. 2018.

[29] Katie Genter, T. Laue, and P. Stone. Three years of the robocup standard platform league drop-in player competition. *Autonomous Agents and Multi-Agent Systems*, 31:790–820, 2016.

[30] Katie Genter and Peter Stone. Adding influencing agents to a flock. In *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-16)*, May 2016.

[31] Julio Godoy, Ioannis Karamouzas, Stephen J Guy, and Maria Gini. Online learning for multi-agent local navigation. In *CAVE Workshop at AAMAS*. sn, 2013.

[32] Julio E Godoy, Ioannis Karamouzas, Stephen J Guy, and Maria Gini. Adaptive learning for multi-agent navigation. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 1577–1585, 2015.

[33] Susan Hackwood and Gerardo Beni. Self-organization of sensors for swarm intelligence. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 819–829, 1992.

[34] Dylan Hadfield-Menell, Smitha Milli, Pieter Abbeel, Stuart J Russell, and Anca D Dragan. Inverse reward design. In *Advances in neural information processing systems*, pages 6765–6774, 2017.

[35] Heiko Hamann, Heinz Worn, Karl Crailsheim, and Thomas Schmickl. Spatial macroscopic models of a bio-inspired robotic swarm algorithm. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1415–1420. IEEE, 2008.

[36] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, page 2094–2100. AAAI Press, 2016.

[37] Pablo Hernandez-Leal, Bilal Kartal, and Matthew E Taylor. A survey and critique of multiagent deep reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 33(6):750–797, 2019.

[38] Nicholas Hoff, Robert Wood, and Radhika Nagpal. *Distributed Colony-Level Algorithm Switching for Robot Swarm Foraging*, pages 417–430. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[39] Zhen Hu, Kun Wan, Xin Gao, and Yuhong Zhai. A dynamic adjusting reward function method for deep reinforcement learning with adjustable parameters. In *Mathematical Problems in Engineering*. Hindawi, 2019.

[40] Fabrício R. Inácio, Douglas G. Macharet, and Luiz Chaimowicz. United we move: Decentralized segregated robotic swarm navigation. In *Distributed Autonomous Robotic Systems*, Springer Tracts in Advanced Robotics. Springer, 2016.

[41] Mohammad R Jahanshahi, Stefano Mascaro, and Ronald Bunker. Reconfigurable swarm robots for structural health monitoring. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[42] Yanshu Jing, Yukun Chen, Minghai Jiao, Jie Huang, Bowen Niu, and Wenbo Zheng. Mobile robot path planning based on improved reinforcement learning optimization. In *Proceedings of the 2019 International Conference on Robotics Systems and Vehicle Technology*, pages 138–143, 2019.

[43] Mohammad Khajenejad, Farzaneh Afshinmanesh, Alireza Marandi, and Babak Nadjar Araabi. Intelligent particle swarm optimization using q-learning. In *Proc. IEEE Swarm Intell. Symp*, pages 7–12. Citeseer, 2006.

[44] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research*, 5(1):90–98, 1986.

[45] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[46] Athanasios Krontiris and Kostas E. Bekris. Using minimal communication to improve decentralized conflict resolution for non-holonomic vehicles. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3235–3240, 2011.

[47] C Ronald Kube and Hong Zhang. Collective robotic intelligence. In *Second International Conference on Simulation of Adaptive Behavior*, pages 460–468, 1992.

[48] Manish Kumar, Devendra P Garg, and Vijay Kumar. Segregation of heterogeneous units in a swarm of robotic agents. *Automatic Control, IEEE Transactions on*, 55(3):743–748, 2010.

[49] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2016.

[50] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Double q-learning for reward learning from simulated human feedback. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017.

[51] Pinxin Long, Tingxiang Fan, Xinyi Liao, Wenxi Liu, Hao Zhang, and Jia Pan. Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning. Ithaca, 2018. Cornell University Library, arXiv.org.

[52] Pinxin Long, Wenxi Liu, and Jia Pan. Deep-learned collision avoidance policy for distributed multiagent navigation. *IEEE Robotics and Automation Letters*, 2(2):656–663, 2017.

[53] Qi Lu and Ryan Luna. Adaptive multiple distributed bidirectional spiral path planning for resource collection in foraging robot swarms. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[54] Leandro Soriano Marcolino and Luiz Chaimowicz. A coordination mechanism for swarm navigation: Experiments and analysis (short paper). In *Proc. of the Seventh Int. Conf. on Autonomous Agents and Multiagent Systems*, pages 1203–1206, 2008.

[55] Laurent Matignon, Guillaume J Laurent, and Nadine Le Fort-Piat. Reward function and initial values: Better choices for accelerated goal-directed reinforcement learning. In *International Conference on Simulation of Adaptive Behavior*, pages 485–494. Springer, 2006.

[56] James McLurkin and Erik D Demaine. A distributed boundary detection algorithm for multi-robot systems. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4791–4798. IEEE, 2009.

[57] Syed Irfan Ali Meerza, Moinul Islam, and Md Mohiuddin Uzzal. Q-learning based particle swarm optimization algorithm for optimal path planning of swarm of mobile robots. In *2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT)*, pages 1–5. IEEE, 2019.

[58] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[59] Alexander Mogilner and Leah Edelstein-Keshet. A non-local model for a swarm. *Journal of mathematical biology*, 38(6):534–570, 1999.

[60] Yogeswaran Mohan and S. G. Ponnambalam. An extensive review of research in swarm robotics. In *2009 World Congress on Nature & Biologically Inspired Computing (NaBIC)*, pages 140–145, 2009.

[61] Andrew Y Ng and Stuart Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 663–670. ICML, 2000.

[62] K.C. Ng and M.M. Trivedi. A neuro-fuzzy controller for mobile robot navigation and multirobot convoying. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 28(6):829–840, 1998.

[63] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[64] Alessandro Panella and Piotr Gmytrasiewicz. Interactive pomdps with finite-state models of other agents. *Autonomous Agents and Multi-Agent Systems*, 31(4):861–904, July 2017.

[65] Lukasz Pelcner, Shaling Li, Matheus Aparecido do Carmo Alves, Leandro Soriano Marcolino, and Alex Collins. Real-time learning and planning in environments with swarms: A hierarchical and a parameter-based simulation approach. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '20, page 1019–1027, Richland, SC, 2020. International Foundation for Autonomous Agents and Multiagent Systems.

[66] Julien Pettre, Pablo de Heras Ciechomski, Jonathan Maïm, Barbara Yersin, Jean-Paul Laumond, and Daniel Thalmann. Real-time navigating crowds: Scalable simulation and rendering. *Computer Animation and Virtual Worlds*, 17:445–455, 07 2006.

[67] Craig W Reynolds. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 25–34, 1987.

[68] Erol Şahin. Swarm robotics: From sources of inspiration to domains of application. In *International workshop on swarm robotics*, pages 10–20. Springer, 2004.

[69] Vinicius Santos, Mario Campos, and Luiz Chaimowicz. On segregative behaviors using flocking and velocity obstacles. *Springer Tracts in Advanced Robotics*, 104:121–133, 01 2014.

[70] Vinicius Graciano Santos, Mario F. M. Campos, and Luiz Chaimowicz. *On Segregative Behaviors Using Flocking and Velocity Obstacles*, pages 121–133. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[71] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[72] Pierre Sermanet, Koray Kavukcuoglu, Soumith Chintala, and Yann LeCun. Pedestrian detection with unsupervised multi-stage feature learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3626–3633, 2013.

[73] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. Deep reinforcement learning for autonomous driving. In *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018.

[74] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv:1611.06594*, 2018.

[75] Bruno Siciliano, Oussama Khatib, and Torsten Kröger. *Springer handbook of robotics*, volume 200. Springer, 2008.

[76] Roland Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. *Introduction to autonomous mobile robots*. MIT press, 2011.

[77] Leandro Soriano Marcolino, Yuri Tavares dos Passos, Alvaro Antonio Fonseca de Souza, Andersoney dos Santos Rodrigues, and Luiz Chaimowicz. Avoiding target congestion on the navigation of robotic swarms. *Autonomous Robots*, 41(6):1297–1320, 2017.

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

[79] L. S. Marcolino T. M. Grabe, F. R. In´acio. Stand by me: Learning to keep cohesion in the navigation of heterogeneous swarms. In *4th International Symposium on Swarm Behavior and Bio-Inspired Robotics (SWARMS 2021)*, 2021.

[80] Sebastian Thrun. Probabilistic robotics. *Commun. ACM*, 45(3):52–57, mar 2002.

[81] Sebastian Thrun and Anton Schwartz. Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum*, volume 6, pages 1–9, 1993.

[82] Y. Tsurumine, Y. Cui, E. Uchibe, and T. Matsubara. Deep reinforcement learning with smooth policy update: Application to robotic cloth manipulation. *Robotics and Autonomous Systems*, 114:80–92, 2019.

[83] Ali E Turgut, Hande Çelikkanat, Fatih Gökçe, and Erol Şahin. Self-organized flocking in mobile robot swarms. *Swarm Intelligence*, 2(2):97–120, 2008.

[84] Jur Van Den Berg, Stephen J Guy, Ming Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. In *Robotics research*, pages 3–19. Springer, 2011.

[85] Jur Van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *2008 IEEE international conference on robotics and automation*, pages 1928–1935. Ieee, 2008.

[86] Jito Vanualailai and Bibhya Sharma. A lagrangian-based swarming behavior in the absence of obstacles. In *Workshop on Mathematical Control Theory, Kobe University*, pages 8–10, 2010.

[87] Harish Verlekar and Kashyap Joshi. Ant & bee inspired foraging swarm robots using computer vision. In *2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT)*, pages 191–195, 2017.

[88] Rich Washington, Keith Golden, John Bresina, David Smith, Corin Anderson, and Trey Smith. Autonomous rovers for mars exploration. volume 1, pages 237 – 251 vol.1, 02 1999.

[89] C. J. C. H. Watkins. Learning from delayed rewards. 1989.

[90] Alfred Wurr and John Anderson. Multi-agent trail making for stigmergic navigation. In *Conference of the Canadian Society for Computational Studies of Intelligence*, pages 422–428. Springer, 2004.

[91] Tong Xie, Shu Zhao, Cheng-Hao Wu, Yan Liu, Qi Luo, and Victor Zhong. Text2reward: Automated dense reward function generation for reinforcement learning. *arXiv preprint arXiv:2309.11489*, 2023.

[92] Dimitri Zarzhitsky, Diana F Spears, and William M Spears. Swarms for chemical plume tracing. In *Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005.*, pages 249–256. IEEE, 2005.

[93] Marius Šumanas, Vytautas Bučinskas, Inga Morkvėnaitė Vilkončienė, Andrius Dzedziskis, and Tadas Lenkutis. Implementation of machine learning algorithms for autonomous robot trajectory resolving. In *2019 Open Conference of Electrical, Electronic and Information Sciences (eStream)*, pages 1–3, 2019.