Decentralized Multi-Agent Reinforcement Learning for Collective Transport

by

Apratim Mukherjee

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Robotics Engineering

May 2023

APPROVED:

Prof. Carlo Pinciroli, Advisor

Prof. Greg Lewin, Professor, Robotics Engineering

Prof. Nitin Sanket, Professor, Robotics Engineering

Abstract

Deep Reinforcement Learning (DRL) has seen recent success in controlling individual robots, but extending to multi-robot systems presents substantial challenges. Non-stationarity is a critical issue that arises when multiple robots learn concurrently, resulting in an interdependent training process without guaranteed convergence. Addressing non-stationarity often demands unrealistic assumptions such as global information. In this thesis, we investigate four vanilla Reinforcement Learning algorithms applied to a multi-robot system where all of the robots are rigidly connected, commonly referred to as a robot aggregate. We limit the sensing capabilities of the robots to proximity sensors and remove the ability to directly communicate. Our approach is validated by using a collective transport task where robots are pre-attached to an object that must be transported to a predetermined location. We assume that the robots are minimalistic, capable of sensing the target location and nearby obstacles, but without explicit communication abilities, such as message-passing. Instead, they communicate implicitly through the aggregate push-and-pull forces exerted on the object. We apply Centralized Training Decentralized Execution to analyze the coordination capabilities of four prominent deep reinforcement learning algorithms (DQN, DDQN, DDPG, and TD3), investigating the scalability, resilience, and obstacle avoidance capabilities of the robot aggregate in a simulated multi-agent environment. Through a comprehensive study with our experiments, we measure the performance as the successful transport of the object to a goal location within a desired timeframe, highlighting the strengths and weaknesses of each algorithm.

Acknowledgements

I would like to express my sincere gratitude to my advisor, Prof. Carlo Pinciroli, for his continuous guidance, invaluable feedback, and unwavering support throughout my time at WPI. His extensive knowledge and insightful advice has been pivotal not only in shaping my thesis, but also my life after grad school, for which I will be eternally grateful. I would also like to thank my committee members, Prof. Greg Lewin, and Prof. Nitin Sanket, for their constructive feedback on my thesis.

I am also profoundly grateful to Prof. Emmanuel Agu, who provided me the opportunity to work with him as a research student in 2019. His support and guidance have been instrumental in shaping my research journey, and I owe my current position to his mentorship.

Furthermore, I would like to acknowledge the incredible work done by Ph.D. candidate Joshua Bloom, whose exceptional work laid the foundation for this work. His collaboration and motivation has made this journey truly enjoyable, and his initial work on setting up this entire idea and bringing me on as a contributor is what made this possible. Josh's significant contributions spanned formulating the problem statement, designing and setting up the initial simulations, and testing baseline value-based networks. These efforts provided a solid foundation for our collaborative expansion on various other models and experiments. Moreover, Josh's eye for detail and visualization skills were instrumental in creating the environment setup diagrams and trajectory plots that effectively showcase our work. I'm deeply grateful for his invaluable support and for the opportunity to work alongside such a brilliant researcher.

I am also grateful to my fellow members at NEST Lab for their stimulating discussions, thought-provoking questions, and camaraderie. Their insightful discussions, thought-provoking questions, and invaluable feedback have been essential in shaping and refining my work. The collaborative spirit fostered by everyone in NEST Lab have made my time there truly enriching and memorable. I am immensely thankful to each and every member, especially Brian Francis, Khai Yi Chin, Davis Catherman, and Ashay Aswale, for their contributions to my growth as a researcher and for being a part of this fulfilling journey.

In addition, my friends provided me with continuous support and encouragement during this challenging yet rewarding process, offering a helping hand whenever I needed it. Their kindness, understanding, and companionship have provided me with the strength and motivation to persevere and overcome obstacles along the way. I am truly blessed to have such an incredible group of individuals in my life, and their presence has been instrumental in maintaining a sense of balance and well-being. I cannot thank them enough for their friendship and the invaluable role they have played in my personal and professional journey.

I would also like to extend my heartfelt gratitude to my teammates at Tesla, namely Gabrielle Trotter, Jeff Stoll, Jitesh Shah, Tristan Seroff, and Jacob Murphy, with whom I had the pleasure of working alongside for over six months during my internship. Their dedication, expertise, and collaborative spirit have not only enriched my knowledge and skills but also inspired me to strive for excellence in every aspect of my work. I will forever cherish the memories we forged during our time together.

I would be remiss if I didn't express my gratitude to my teammates at my undergraduate robotics team, Project MANAS, which ignited my passion for robotics and paved the way for my current pursuits. I am deeply thankful for the opportunity to have been a part of such an incredible team, and I will always carry the lessons learned and the friendships formed during that time with great fondness and appreciation.

Lastly, I am eternally indebted to my family for their unwavering love, faith, and support. My parents have always believed in my potential, and my brother has always provided me with constant encouragement. I have always had the love and support necessary to follow my passion, and they have been my pillars of strength since the very beginning. They have been the bedrock of my success, and have always stood by me. The values they have instilled in me and the sacrifices they have made have been instrumental in shaping the person I am today. I dedicate this accomplishment to them as a testament to the power of their immense faith and support, and I love them more than they will ever know.

This thesis would not have been possible without the contributions of all the incredible individuals in my life, and I am truly grateful for each and every one of them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Reinforcement learning (RL) [1] is a sub-field of machine learning that focuses on an agent and its ability to learn the optimal behaviour. The intended behaviour is to make correct decisions in an unknown environment to maximize its cumulative reward and achieve the desired goal. The reward quantifies the desirability of an agent's action in a given state. It can be thought of as a feedback mechanism that assigns a numerical value to the agent's decisions, enabling it to understand the consequences of its actions within the environment. High rewards signify that the agent has taken favorable actions, while low rewards indicate less desirable outcomes. The ultimate goal of the agent is to maximize the cumulative rewards it receives over time, which leads to the discovery of an optimal strategy or policy for navigating the environment and achieving its objectives. This policy is learned purely by interacting with the environment and receiving continuous feedback in the form of a reward signal. RL has been the subject of study for decades, however, recent advancements in computing power have opened the door to complex function approximators allowing for a broad application of techniques.

In recent years, the integration of deep learning techniques with RL has led to significant breakthroughs in various domains, such as game playing [2], robotics [3], natural language processing [4], and autonomous systems [5]. RL provides a framework for learning optimal decision-making in complex, dynamic, and uncertain environments. Unlike supervised learning, RL does not rely on labeled data in most cases, which enables agents to learn from interactions with the environment, making it particularly suitable for applications where the agents need to be agnostic to the model of the world around them and figure out the rules purely through a feedback loop.

Although RL has been used successfully in the domain of single agent control problems [2], multi-agent settings pose significant challenges for these algorithms. One of the most critical problems is that of non-stationary environments. Non-stationarity arises due to the dynamic nature of these environments, where the behaviour of each agent changes concurrently as they learn and adapt their policies. This causes the environment to shift continuously, making it difficult to achieve convergence as agents get stuck in a loop of learning sub-optimal policies, also called the moving target problem. There are other pitfalls as well, such as the need for coordination, communication, and dealing with partially observable environments. These problems are often tackled by using advanced training schemes and complex message passing between the robots [6], [7], [8] . Multi-agent reinforcement learning (MARL) provides a powerful framework for enabling agents to learn how to cooperate and coordinate autonomously in such decentralized systems.

MARL is a very active area of research as it can lead to potential applications in the fields of swarm robotics, autonomous vehicles, distributed control systems, communication networks, and more. Continuing to push the boundaries of MARL can lead to improvements in these areas, resulting in more efficient and robust systems. For example, MARL has been successfully applied to cooperative control in robotic systems [9] and coordination of autonomous vehicles [10], [11].

Exploring MARL can provide insights into the behavior of complex systems, such as natural systems (e.g., animal groups [12], ecosystems [13]) and social systems (e.g., markets [14], human organizations [15]). By modeling these systems as multi-agent systems, researchers can gain a better understanding of their dynamics and develop more effective strategies for managing them [16].

MARL also offers the opportunity to study the emergence of collective behaviors and understand how individual agents can learn to cooperate and coordinate their actions effectively. Understanding the process of learning these emergent behaviors can provide insights into the design of better algorithms and systems for swarm intelligence and collective decision-making.

In recent years, MARL has been applied to swarm robotics, with a notable example being collective transport [6], [7]. Collective transport is a challenging problem in swarm robotics, where a group of robots cooperatively transports an object to a target location. It is an important task with applications in search and rescue, construction, and logistics. MARL has been applied to the collective transport problem in swarm robotics to enable the robots to learn how to coordinate their actions effectively.

Collective transport tasks often involve coordinating multiple agents with different roles, goals, and capabilities. This complexity makes the problem challenging and requires sophisticated algorithms to learn effective policies and strategies. In many real-world scenarios, it is beneficial to have decentralized control, where agents can make decisions autonomously without relying on a central authority. MARL allows for decentralized decision-making, enabling agents to adapt and respond to changes in the environment more effectively.

Studying collective transport using MARL can help develop algorithms that can scale up to accommodate large numbers of agents, making it suitable for real-world applications where swarm behavior is desirable. Agents also learn to handle uncertainties, such as communication failures or unexpected obstacles. MARL in a distributed setting can provide fault tolerance, as the system can still function even if some agents fail or become unavailable.

1.1 Problem Statement

In this thesis, we address the challenge of employing MARL for the management of robot aggregates, which refers to groups of robots physically linked to one another through mechanical connections. Controlling robot aggregates is not a trivial task, as it requires the robots to learn behaviours that result in coordinated pushing and pulling.

We mention the problem of non-stationarity earlier in the section, and how it affects the training of agents as the environment is constantly changing. A common solution is to use message-passing, however, we show that effective coordination can be achieved with well known RL algorithms even when explicit communication between the robots is not possible. The robots are unable to share their actions and observations, making it a partially observable environment. We consider a collective transport scenario where a swarm of minimalistic robots need to move a target object to a predefined goal location. We assume that robots are able to control their wheel velocities and acquire basic proximity sensor readings, but are unable to communicate this information directly to other robots through message passing. They are also able to sense the direction in which they are heading, i.e., are they moving towards the goal or away from the goal, and also if the object is moving towards or away from the goal. Most importantly, the robots are also physically constrained to the object with the help of grippers which keep them connected to the object throughout the duration of the experiment, unless special circumstances arise and robot failures are introduced.

This task is a compelling test-bed for decentralized multi-agent reinforcement learning as the constraints placed on the group during the experiments force the robots to coordinate transport. The inability to communicate amongst themselves, and the physical attachment to the object that needs to be moved causes the robots to learn to push and pull together, and model the behaviour and actions of other robots in the aggregate. This is a form of implicit communication which effectively circumvents the non-stationarity problem, since training for individual robots is agnostic of the observation spaces and actions of others.

There are common schemes to train the algorithms in a multi-agent setting, with a few common ones being Centralized Training Centralized Execution (CTCE), Centralized Training Decentralized Execution (CTDE), and Decentralized Training Decentralized

Execution (DTDE). CTCE trains a single policy on global information and produces a joint action based on the state which includes observations from all agents in the environment. CTDE trains a single policy on local information and single agent actions, which is then copied on each agent in the environment during execution, producing individual actions per agent. DTDE trains multiple policies on the local information of the agents independently, and these policies are all executed independently during runtime based on the observations of individual agents.

1.2 Contributions

Our goal is to demonstrate that our insight is not only valid but also practical in the context of RL. We compare the performance of four widely-used RL algorithms to showcase the effectiveness of our approach. The four RL algorithms used for our validation are: Deep Q-Network (DQN) [2], Double Deep Q-Network (DDQN) [17], Deep Deterministic Policy Gradient (DDPG) [18], and Twin Delayed Deep Deterministic Policy Gradient (TD3) [19]. The methods above are trained by means of CTDE, where one model is trained with the help of shared experiences of all robots (centralized training) which is then deployed as individual instances on every single agent (decentralized execution).

We show that implicit communication occurs between the robots in the form of natural push and pull actions, and the robots learn to coordinate because of the discrepancy between their actuation and applied force on the object, and the resulting movement of the transported object. This form of communication is sufficient for the robot aggregate to complete the task with relatively high success, and manage to learn a control policy that is capable of emergent coordination. The coordination of the forces applied to the object by the robots is not explicitly modelled, and the agents learn to synchronize their actions purely by modulating their wheel speeds in the simulations. The push and pull forces are implicitly modelled and learnt by the agents as they try to minimize the distance of the robot aggregate over the period of a training episode.

Our findings demonstrate that, even with the simplistic nature of the robots, utilizing the object as a means for communication and coordination leads to favorable outcomes and solutions exhibiting scalability, robustness against failures, and strategies for evading obstacles. We analyze the behavior and efficacy of the strategies generated by the RL algorithms through an extensive series of experiments and benchmarks.

We also establish an initial framework to explore the concept of predicting the next state of the object, and using it to improve the control strategies. The robots avoid global state knowledge, but use minimal local information from other robots to try and predict the state of the object in the next time step. This has lower communication requirements than global information sharing, but the results show that it helps boost the performance of the robot aggregate in transporting the object to the desired goal.

Chapter 2

Background and Related Work

2.1 Reinforcement Learning (RL)

RL has its roots in the concept of Markov Decision Processes (MDPs), which are mathematical frameworks used to model decision-making in situations where outcomes are uncertain. An MDP is defined by a tuple $\langle S, A, R, T \rangle$, where S represents the set of states, A is the set of actions, $R(s,a,s') : S \times A \times S \to \mathbb{R}$ is the reward function, and $T(s'|s,a)$ is a probabilistic function mapping states and actions to a new state. MDPs are typically solved using the Bellman Equation, which defines the value of a state-action pair, $Q(s,a)$, as the expected future rewards, taking into account a discount factor, γ.

$$Q(s,a) = \sum_{s'} T(s'|s,a)[R(s,a,s') + \gamma \max_{a'} Q(s',a')] \tag{1}$$

When applying RL to robotics, complexities arise due to the robots' reliance on sensors to perceive their environment. In most cases, these sensors only provide a partial observation of the true state of the environment, necessitating an extension of MDPs to Partially Observable MDPs (POMDPs). POMDPs are represented by the tuple $\langle S, A, R, T, \Omega, O \rangle$, where $o \in \Omega(s)$ is a partial observation of the full state $s \in S$ according to a probabilistic function $O(o|s)$.

This does not translate very well into the real-world though, where the state-action space can be enormous, which makes it challenging to find the true values of Q. Recently, neural networks (NNs) have emerged as a promising method for approximating the value function $Q(s, a)$. Two of the most common approaches in the field of RL are value-based and policy-based methods. These techniques have been instrumental in advancing the application of RL to complex problems in robotics and other domains, paving the way for further innovations and discoveries.

Mnih *et al.* [2] made a significant breakthrough in the field of deep reinforcement learning by introducing the Deep Q-Network (DQN) algorithm. DQN employs a deep neural network to learn the mapping between states and the expected future rewards of actions, as described by Equation 1. This innovative method demonstrated remarkable success, achieving human-level performance in several Atari games. However, DQN suffers from overestimation bias of the Q-values, leading to unstable convergence in partially observable settings. Overestimation bias refers to the tendency of certain algorithms to overestimate the expected future rewards associated with taking specific actions in given states. This can lead the agent to develop an overly optimistic view of its actions, causing it to prioritize certain decisions based on inflated expectations. Consequently, this bias can hinder the learning process and negatively impact the agent's ability to converge to an optimal policy. Handling overestimation bias is crucial for improving the robustness and performance of RL algorithms in diverse environments.

Van Hasselt *et al.* [17] addressed this issue by developing the Double Deep Q-Network (DDQN), which employs a decoupled copy of the current Q-network, referred to as Q'. This separate network is used to estimate the best action to take and is updated frequently to match the current weights of the primary Q-network. While value-based methods, such as DQN and DDQN, are effective at learning complex mappings between states and actions, they may be hindered by long convergence times.

In contrast, other methods such as hybrid policy-value networks work directly on the mapping from states $s \in S$ to actions $a \in A$, known as the policy $a = \mu(s)$. Lillicrap *et al.* [18] applied neural networks to the Deterministic Policy Gradient (DDPG) algorithm, which uses an actor to learn a policy μ and a critic to evaluate the actor's policy through a value function $Q(s,a)$. Similar to DQN, DDPG is prone to overestimation bias in its value function, Q, resulting in inflated estimates of expected future rewards.

To mitigate this overestimation bias, Fujimoto *et al.* [19] introduced the Twin Delayed DDPG (TD3) algorithm, which incorporates a two-critic evaluation system. When evaluating the policy μ, two separate critics calculate evaluation values, and the minimum value between them is chosen. This approach helps minimize overestimation bias, leading to more accurate and stable learning in deep reinforcement learning tasks.

2.2 Multi-Agent Reinforcement Learning (MARL)

MARL is an intricate area of study, focusing on the interactions between multiple agents within an environment. It is a complex domain that can be formalized as a Decentralized Partially Observable Markov Decision Process (Dec-POMDP), represented by the tuple $\langle I, S, \{A_i\}, R, T, \{\Omega_i\}, O\rangle$. Here, I represents the set of agents, $A = \times_i A_i$ denotes the set of joint actions, and $\Omega = \times_i \Omega_i$ is the set of joint observations.

Dec-POMDPs often assume communication, synchronization, or observability across all agents in the environment to tackle the issue of non-stationarity. This presents a significant challenge in MARL as non-stationarity arises when agents treat their neighbors as elements of the environment. This perception leads to an interdependence in the agents' behaviors, causing convergence issues during learning. To mitigate non-stationarity, researchers often propose solutions that involve information sharing among agents.

For instance, Amato *et al.* [6] addressed non-stationarity by facilitating consensus on a joint action space, enabling agents to coordinate their actions. Similarly, Foerster *et al.* [7] introduced the idea of communicating differentiable units, providing a feedback mechanism to aid in training. Another approach, proposed by Wu *et al.* [8], involved constructing a belief state using spatial intention maps, which capture the physical locations of nearby agents over time.

In contrast to these explicit information-sharing methods, aggregates offer an intriguing alternative. By providing a physical connection between agents, aggregates allow for an indirect form of communication. This eliminates the need for explicit communication or direct observation of other agents, presenting a novel and unexplored coordination avenue in RL.

2.3 MARL for Collective Transport

The growing interest in applying RL to aggregate and modular robots for collective transport tasks is a testament to the potential benefits of incorporating deep learning in robotics. In recent studies, researchers have explored various RL techniques to develop effective control strategies for multi-agent systems in different transport settings.

Zhang *et al*. [20] pioneered the use of Double Deep Q-Network (DDQN) in a two-robot transportation task where robots had to remain attached to an oversized rod. The robots were given a set of four predefined actions, such as forward-left and backward-left, to choose from. By assuming perfect knowledge of other agents' pose and velocity, the researchers were able to produce control strategies that facilitated successful rod transportation through a door obstacle. This work highlighted the potential for applying deep RL techniques in multi-agent systems, paving the way for further investigation into their applicability in complex transport tasks.

Alternative approaches to collective transport, where robots are not constrained to the object and instead rely on pushing mechanisms, have also gained attention in the literature. Studies such as those conducted by Wang *et al*. [21] and Rahimi *et al*. [22] demonstrated the potential of Q-learning in box-pushing tasks. In these experiments, robots had to learn the most effective locations to push the box from a discrete set of predefined positions. These findings suggest that RL can help robots develop an understanding of their environment, ultimately improving their ability to coordinate and collaborate during transport tasks.

Eoh *et al*. [23] took this idea further by implementing Deep Q-Network (DQN) in a multi-agent box-pushing task. The researchers compared the performance of independent learners with a policy-reuse scheme, highlighting the importance of considering different learning strategies when designing multi-agent systems. This study underscored the need to explore various RL techniques and learning frameworks to overcome the non-stationary nature of multi-agent transport problems, which arises from the continuous changes in robots' policies during training.

The emerging body of literature on RL applications in aggregate and modular robots for collective transport tasks showcases the potential of these techniques in developing sophisticated control strategies for multi-agent systems. Further exploration of various RL algorithms, learning frameworks, and multi-agent coordination mechanisms will be crucial to overcoming the challenges posed by non-stationary environments and unlocking the full potential of deep RL in robot aggregates.

Chapter 3

Methodology

3.1 Problem Formulation

3.1.1 Collective Transport

The main goal of our simulations is to test the different RL algorithms in a multi-agent environment for collective transport. To do this, we assume a cylindrical object to be transported to a goal location in simulations. The robots are distributed evenly and symmetrically around the the cylinder, and attach to it with the help of a gripper. The gripping action between the robot and the cylinder introduces the physical constraint in our experiments, and lets the robots move it based on their individual actuation and exerted forces on the cylinder. The forces are symmetric because of the nature of how the robots are distributed around the cylinder, but the robots can also fail during the experiments

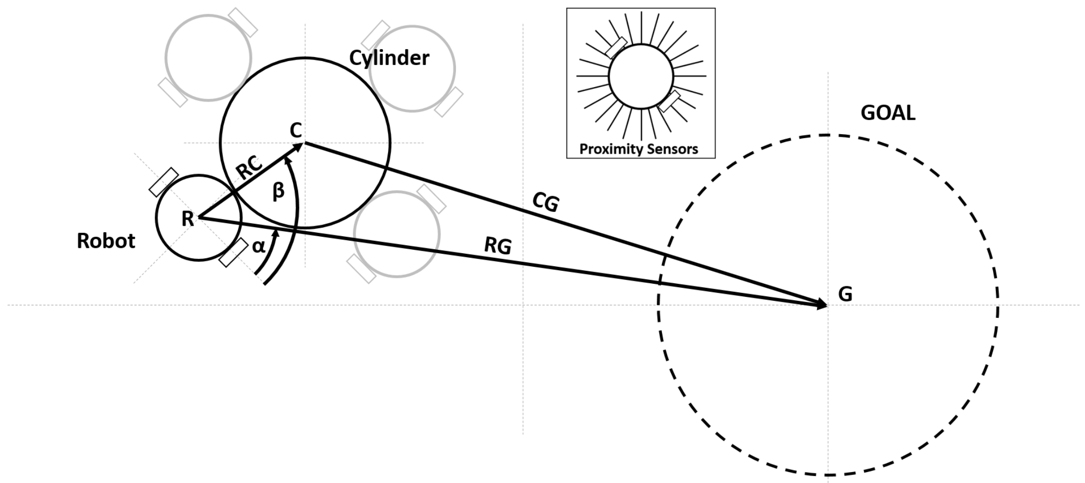


Figure 1: The robots acquire spatial information including the distances and angles to the cylinder and the goal from themselves. They also know the distance from the cylinder to the goal and an array of 24 proximity sensor readings that are uniformly distributed around the robot.

which introduces asymmetry during runtime.

3.1.2 Robot Sensing

All robots are identical and can each individually observe their local space in the form of a tuple $\langle \overrightarrow{RG}, \overrightarrow{RC}, \overrightarrow{CG}, \mathbf{p} \rangle$, according to their local frame of reference as shown in Figure 1. These observations are used in the training of the models, as well as evaluating the reward function (Equation 2) in the current state. $\overrightarrow{RG}$ is the vector that they observe from themselves to the goal, $\overrightarrow{RC}$ is the vector from themselves to the cylinder that needs to be moved, $\overrightarrow{CG}$ is the distance vector between the cylinder and the goal, and an array of proximity values $\mathbf{p}$. Each robot has 24 proximity sensors which produce the array $\mathbf{p}$, with values ranging between 0 and 1. The sensors have a range of 2m, with 0 implying that there is nothing in their immediate proximity and 1 meaning that they are in contact with an object in the environment. They also keep track of a few other values, such as $\mathbf{v}$, α, and β along with the the initial tuple. $\mathbf{v}$ indicates the wheel speeds of the robot, α is the angle between the axis of the robot and the vector $\overrightarrow{RG}$, and β is the angle between the robot axis and the vector $\overrightarrow{RC}$. These values together form the local observation space of an individual robot. When trying to predict the next state of the object with the help of limited communication, the robot also perceives additional values from the other actors in the aggregate in the form of their average proximity values $\sum \mathbf{p}/|\mathbf{p}|$.

3.1.3 Implicit Communication

Each robot in the system operates independently and lacks explicit communication channels to share information with its counterparts. Instead, the robots are rigidly connected to the object being transported, which serves as an implicit means of communication.

From the perspective of an individual agent, the actions taken by other robots can be perceived as deviations from its own intended action and the resulting state of the object. By considering the overall dynamics of the object, it is possible to model the combined effect of all the agents as a single actor with an associated error term. This error term reflects the discrepancy between the desired and actual actions due to the presence of multiple agents.

Given that all the agents share the same goal and reward function, their collective objective is to minimize this error term. In this context, the agents need to learn how to coordinate their actions effectively to reduce the error and achieve efficient, stable, and coordinated collective transport. The methodology adopted should enable the agents to optimize their policies in a decentralized manner, despite the absence of explicit communication.

In this setting, the agents can use their local observations, which include the effects of other agents' actions on the object, to update the global policy. By iteratively refining this policy, the agents can gradually converge towards a coordinated and efficient collective transport strategy, effectively minimizing the error term and achieving the shared goal.

3.1.4 Robot Control

We employ the foot-bot [24] in our simulations, a differential-drive robot equipped with a non-actuated turret connected to an actuated gripper. The design of the turret allows for independent rotation of the base with respect to the gripper. It is important to note that the gripper is not actively controlled by the robot during the collective transport task; it is only actuated during the initialization process or in case of a failure.

The foot-bot robots can be controlled by adjusting their wheel velocities, denoted as v, where $v \in (-10, 10)$ cm/s. The agents choose a change in wheel velocity, Δv, according to the type of reinforcement learning method being employed. For value-based methods, the agents select a Δv from a discrete set of $\Delta v \in \{-0.1, 0, 0.1\}$ cm/s, while for actor-critic methods, the agents can choose any Δv within the continuous range of $\Delta v \in [-0.1, 0.1]$ cm/s. This difference reflects the distinction between discretized action spaces in value-based methods and continuous action spaces in policy-based and actor-critic methods.

In the case of value-based methods, it is necessary to consider all possible combinations of Δv for both wheels, resulting in an action space represented by the square of the cardinality of Δv ($|\Delta v|^2$). To simplify the control scheme, we choose a cardinality of 3 for Δv, which leads to a manageable action space size while still allowing for adequate control granularity.

By using this control scheme, the foot-bot robots can learn to coordinate their actions effectively in the collective transport task, adapting their wheel velocities to ensure smooth and efficient transport of the object while maintaining the necessary cooperation among the agents.

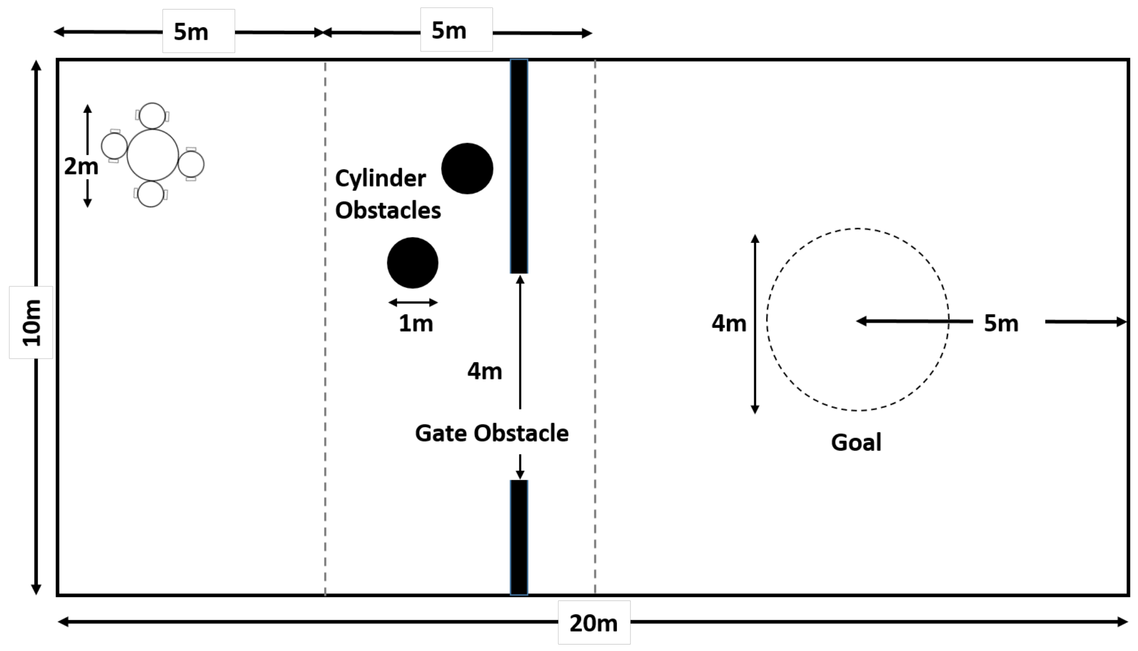


Figure 2: The environment with the two types of obstacles we considered: cylinders and gate. The experiments feature only one type of obstacle. The robots and the object to transport are generated in the region on the left of the obstacles; the goal is generated in the region on the right.

3.1.5 Obstacles

In the proposed environment, obstacles are introduced to increase the complexity and challenge of the collective transport task. As depicted in Figure 2, these obstacles are stationary, non-movable, and rigid objects that the robots must navigate around during the transport process. The obstacles are generated following a uniform distribution, ensuring a random and diverse layout in each trial.

Two types of obstacles are considered: cylindrical obstacles and a gate-like structure. To generate the cylindrical obstacles, we randomly select their (x, y) coordinates within the allowable environment boundaries, ensuring that the robots encounter varying obstacle configurations in different trials. This encourages the development of robust and adaptable strategies for navigating around obstacles.

The gate structure is generated by first randomly selecting its x coordinate and then determining the opening position, represented by the y coordinate. The gate serves as a more complex obstacle, requiring the robots to navigate through a narrow passage while maintaining coordination and control over the transported object. This added challenge emphasizes the importance of developing efficient and cooperative strategies that can handle diverse and dynamic environments.

By incorporating both cylindrical obstacles and a gate structure, we ensure that the robots are exposed to a variety of scenarios, and expect the development of versatile and robust policies that can be applied in cases where unforeseen obstacles and environmental constraints are likely to be encountered.

3.2 Approach

3.2.1 Training

For our training, we adopt the Centralized Training for Decentralized Execution (CTDE) approach. This paradigm involves training a single centralized model using experiences collected from multiple agents, which is then deployed for decentralized execution by individual agents during runtime. The CTDE framework enables the agents to learn cooperative strategies while maintaining the benefits of decentralized control during execution.

During the training phase, the agents generate experiences by interacting with the environment, and these experiences are accumulated and collectively contribute to the replay buffer. This replay buffer is then utilized to train a global model, effectively capturing the collective knowledge and cooperative strategies developed by the agents. The training process incorporates local observations from all participating agents, ensuring that the learned policy is generalizable and adaptable to different team sizes and configurations.

Algorithm 1 Centralized Training Decentralized Execution (CTDE)

```
Initialize Policy $\pi_0$ and Buffer $B$
for each episode do
    Initialize Robots, Object, and Obstacles
    Receive initial observations $O_t$
    while not done at timestep $t$ do
        for each robot $i$ do
            $a_t^i \leftarrow \pi_t(o_t^i)$
        Execute actions $A_t$ and receive ($O_{t+1}$, $R_t$, Done)
        for each robot $i$ do
            $B \leftarrow (o_t^i, a_t^i, r_t^i, o_{t+1}^i$, Done)
        $\pi_{t+1} \leftarrow$ Update Policy $\pi_t$
        $o_t \leftarrow o_{t+1}$
```

Once the centralized model is trained, it is directly copied to each individual agent for decentralized execution. During runtime, the agents rely solely on their local experiences and observations to make decisions, utilizing the policies learned by the global model. As the centralized model was trained using local observations, it remains agnostic to the number of agents present during execution, making it applicable in a variety of scenarios with different team sizes.

By implementing the CTDE approach as proposed in Algorithm 1, we can achieve effective coordination and cooperation among the agents in collective transport tasks while preserving the advantages of decentralized control. This methodology allows for scalable and robust solutions, capable of adapting to diverse environments and agent configurations, thus making it suitable for real-world applications in the domain of swarm robotics and beyond.

3.2.2 Curriculum Learning

In order to equip our robots with the ability to learn to effectively navigate through the gate obstacle, we employ a curriculum learning approach [25]. This method involves gradually increasing the complexity of the learning environment, allowing the agents to develop and refine their skills in a step-by-step manner.

We initiate the curriculum learning process by setting the gate opening equal to the width of the environment, which effectively removes the obstacle from the learning scenario. This initial stage allows the agents to focus on learning the fundamental aspects of cooperation and coordination without the added complexity of navigating through the gate.

Once the agents have developed a baseline understanding of the cooperative transport task, we progressively decrease the width of the gate opening by 0.5m every 50 episodes, introducing the challenge of navigating through the narrow passage. This gradual increase in difficulty enables the robots to incrementally adapt their strategies and fine-tune their policies to accommodate the more constrained environment.

The curriculum learning process continues until we reach the desired minimum gate opening, which is set to double the max length of the robot-cylinder aggregate. This width is chosen to ensure that the robots have sufficient room to maneuver the object through the gate while maintaining coordination and control. By progressively increasing the complexity of the learning environment, the agents are better equipped to handle real-world scenarios where diverse obstacles and dynamic conditions are likely.

3.2.3 Rewards

It is crucial to design a reward structure that promotes successful learning in environments with failures and obstacles. To achieve this, we explored and evaluated various reward structures, ultimately settling on a composite reward function that strikes a balance between promoting goal-oriented behavior and penalizing undesirable actions.

The chosen reward structure presented in Equation 2 consists of two main components. The first component rewards agents for moving in the direction of the goal, encouraging them to maintain a trajectory towards the target destination. This element of the reward function guides the agents towards the primary objective of the task which is getting the object to the desired destination.

$$R_{i,t} = -2 + \frac{\overrightarrow{CG} \cdot \Big(C(x_t, y_t) - C(x_{t-1}, y_{t-1})\Big)}{|\overrightarrow{CG}| \left| \Big(C(x_t, y_t) - C(x_{t-1}, y_{t-1})\Big) \right|} - \frac{\sum \mathbf{p}}{|\mathbf{p}|} \tag{2}$$

The second component of the reward structure is designed to penalize proximity to obstacles and the time taken to complete the task. This is achieved by incorporating an array of proximity values, denoted as p, which quantifies the agents' closeness to obstacles and other undesirable states. By penalizing proximity and task duration, the agents are encouraged to maintain a safe distance from obstacles and optimize their path to minimize the time spent on the task.

The combination of the goal-directed reward and the proximity-based penalty in the overall reward structure allows the agents to learn effectively and let the algorithm scale in diverse environments. As demonstrated in Section IV, this reward function enables the agents to exhibit resilience in the face of failures and to navigate around obstacles while maintaining coordination and control over the transported object.

3.2.4 Deep Q-Network (DQN)

Deep Q-Network (DQN) is a reinforcement learning algorithm that combines Q-learning with deep neural networks to learn an optimal policy for decision-making in complex environments. DQN addresses the limitations of traditional Q-learning methods in dealing with high-dimensional state and action spaces by approximating the Q-value function using a deep neural network.

We implement a DQN architecture consisting of three fully connected layers. The input layer takes in 31 observations representing the state of the environment and is followed by a hidden layer with 64 neurons, 128 neurons and the output layer as shown in Figure 3. The output layer comprises 9 nodes, accounting for the action space discretization, which allows the network to choose among discrete actions.

The Rectified Linear Unit (ReLU) [26] activation function is applied to both the input and hidden layers, providing non-linearity and improved training performance. The network is optimized using the Adaptive Moment Estimation (ADAM) optimizer [27], which is a popular choice in deep learning for its ability to adapt learning rates for each parameter individually, resulting in faster convergence and improved training efficiency.

By employing DQN with the described network configuration, we aim to effectively learn the optimal policy for our problem, leveraging the power of deep learning to handle high-dimensional state and action spaces.

3.2.5 Double Deep Q-Network (DDQN)

Double Deep Q-Network (DDQN) is an extension of the Deep Q-Network (DQN) algorithm that addresses the issue of overestimation bias of the Q-values, which can hinder learning efficiency and policy quality. DDQN introduces a modification to the Q-value

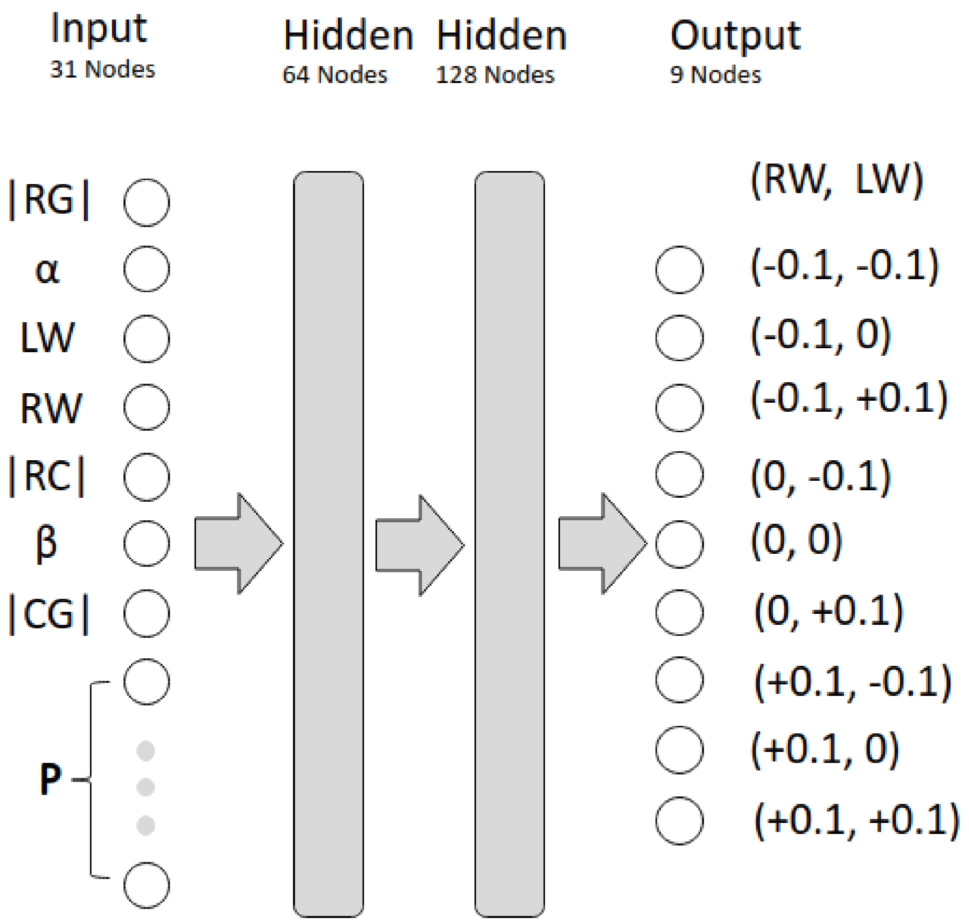


Figure 3: The Q-network architectures for DQN and DDQN are represented here, with 2 hidden layers (64, 128 nodes) and an output layer with 9 discrete actions.

update rule used in DQN by incorporating two separate networks: a target network and an online network.

The target network and the online network have identical architectures but different sets of weights. The online network is used for selecting actions during the learning process, while the target network is employed for generating target Q-values for the update rule. The target network's weights are periodically updated by copying the online network's weights, ensuring that the target Q-values remain stable during the learning process.

By decoupling action selection and Q-value estimation, DDQN effectively reduces the overestimation bias inherent in the original DQN algorithm. This improvement leads to more accurate Q-value estimations, faster convergence, and better policy quality.

During training, the online network is trained using experiences sampled from the replay buffer, which stores the agents' past experiences in the form of state, action, reward, and next state tuples. This experience replay mechanism helps to stabilize training by breaking correlations between sequential experiences and reusing past experiences for multiple updates.

We use the same architecture as the one proposed for DQN and Figure 3, with 3 fully connected hidden layers with 31 inputs and 9 output neurons for our discrete action space.

3.2.6 Deep Deterministic Policy Gradient (DDPG)

Deep Deterministic Policy Gradient (DDPG) is a model-free, off-policy actor-critic algorithm designed for continuous control tasks in reinforcement learning. It combines the strengths of the Deterministic Policy Gradient (DPG) algorithm and deep neural networks to learn optimal policies in high-dimensional continuous action spaces.

DDPG utilizes two separate networks: an actor network and a critic network. The actor network learns a deterministic policy, mapping states to actions, while the critic network learns a state-action value function (Q-value) to evaluate the quality of the actions proposed by the actor network. DDPG also employs a target network for both the actor and critic networks to stabilize training.

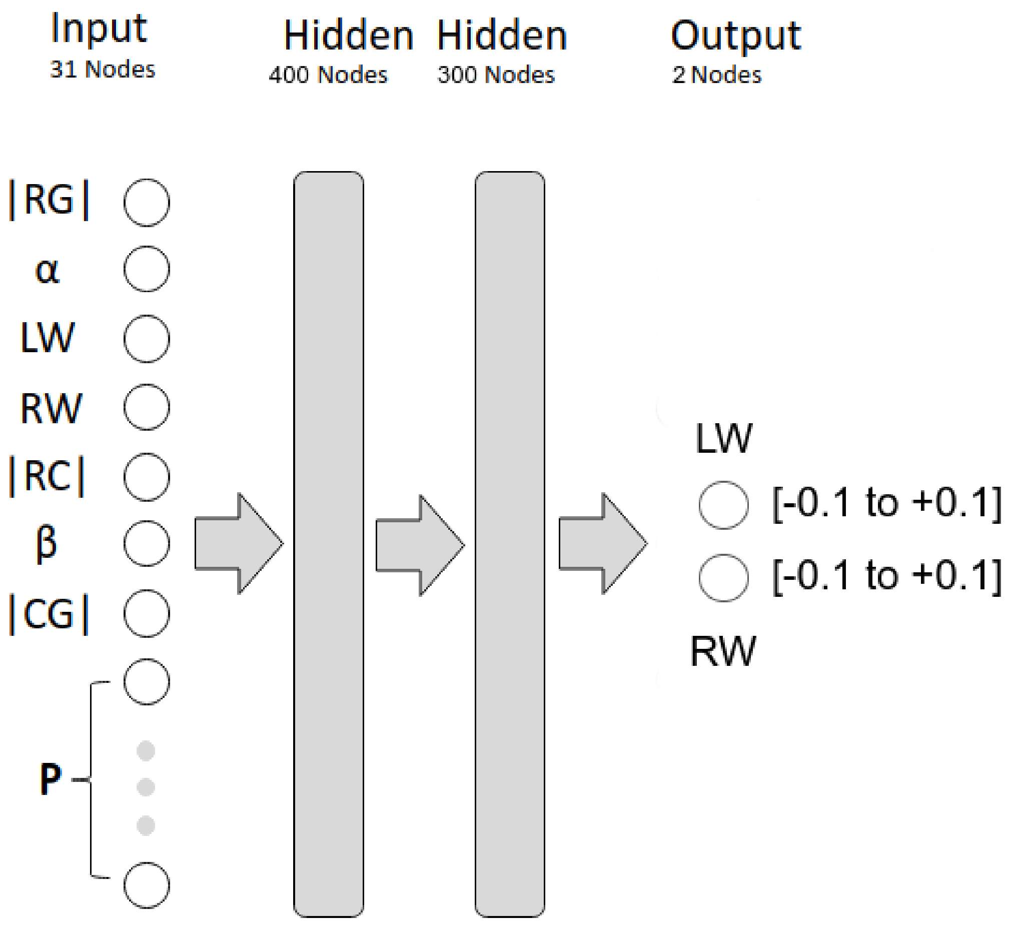


Figure 4: The network architectures for DDPG and TD3 are represented here, with 2 hidden layers (400, 300 nodes) and an output layer with continuous actions for 2 wheels.

During training, DDPG alternates between updating the actor and critic networks. The critic network is trained using a temporal difference (TD) error, which represents the difference between the estimated Q-value and the target Q-value computed using the target networks and the observed reward. The actor network is trained using the policy gradient, which is computed using the gradient of the critic network with respect to the actions.

The architecture for actor-critic methods is inspired by the design presented in [18] and [19], consisting of 3 fully connected layers as presented in Figure 4. The input layer does not change, using 31 nodes to accommodate the state representation. Following the input layer is a hidden layer with 400 input nodes and 300 output nodes, allowing for a higher degree of complexity in the learned policy. The output layer contains two nodes, representing the actions for each wheel of the robot in a continuous action space.

Similar to the value based architecture, the ReLU activation function is used to introduce non-linearities into the network while training. In contrast, the output layer employs the hyperbolic tangent (TanH) [28] activation function, ensuring the output actions lie within a bounded continuous range suitable for the robot's wheel actions. To optimize the network, the Adaptive Moment Estimation (ADAM) optimizer is used, as it leads to faster convergence and improved training performance.

3.2.7 Twin Delayed Deep Deterministic Policy Gradient (TD3)

Twin Delayed Deep Deterministic Policy Gradient (TD3) is an extension of DDPG that introduces major improvements to address function approximation errors and enhance learning stability. TD3 uses two separate critic networks and their corresponding target networks. The minimum of the two target Q-value estimates is used to compute the target Q-value, reducing overestimation bias and improving stability. The actor network and its target network are updated less frequently than the critic networks. This delay allows the critic networks to provide more accurate Q-value estimates before updating the policy, leading to better policy learning. Gaussian noise is added to the target actions during the critic network updates. This encourages the actor network to explore a smoother region of the action space and prevents overfitting to the deterministic policy learned by the actor network.

Both DDPG and TD3 use experience replay as well, which stores past experiences of the robots in a buffer and samples them randomly for training updates to break correlations between sequential experiences and enhances learning stability.

We use the same architecture for TD3 as the one mentioned for DDPG and Figure 4, with hidden layers consisting of 400 nodes, 300 nodes, and 2 outputs. The neurons are activated similarly as well, with ReLU being used as the activation function for the hidden layers barring the output nodes, which use the TanH activation function to bound the wheel speeds between the desired values and each node corresponding to the actuation of the individual wheels of the robot.

Chapter 4

Evaluation

4.1 Experimental Setup

All simulations were conducted on a computer equipped with an Intel i7 processor and an NVIDIA RTX 3070 graphics card, ensuring sufficient computational power for the simulation and training processes. We employed the ARGoS Multi-Robot Simulator [29] for creating multi-agent environments, and the Buzz swarm programming language [30] for implementing swarm behavior and control. The PyTorch [31] deep learning library was utilized for designing and training the reinforcement learning models.

During the simulated experiments, ARGoS communicated observations from the simulated environment to a Python server via ZeroMQ, a high-performance messaging library. The Python server processed these observations using PyTorch to learn and select appropriate actions. The chosen actions were then sent back to ARGoS and executed through the Buzz programming language.

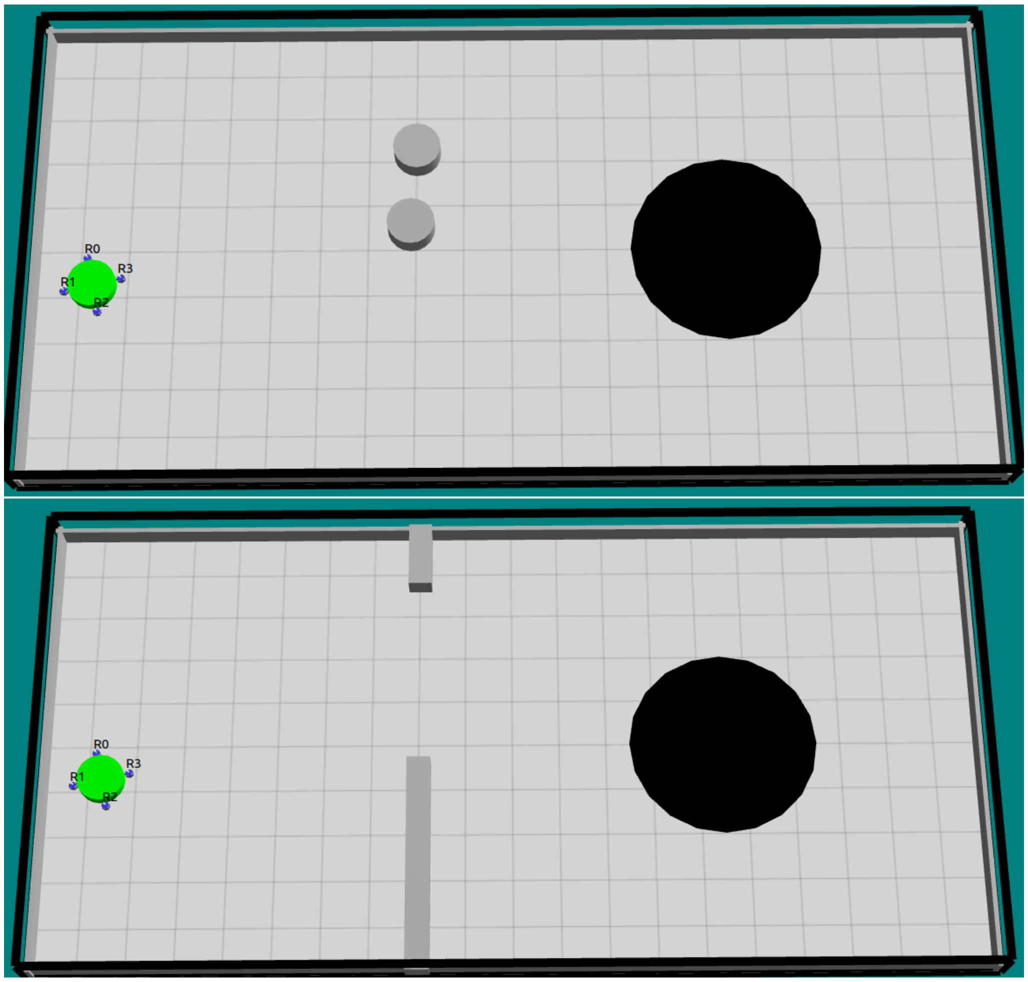


Figure 5: Examples of the cylindrical obstacles (top) and gate obstacles (bottom) in ARGoS are represented in this figure, along with the robots (blue), the goal location (black), and the object to be transported (green).

The training process spanned 1,000 episodes, with each episode terminating either upon reaching the goal or exceeding a time limit of 4,500 time-steps. To monitor the progress and facilitate model selection, the current weights of the learning model were saved every 10 episodes. After completing the training phase, the best-performing model was identified based on their goal completion percentages and subjected to further evaluation through 500 test cases to assess its generalization capabilities and overall performance.

The hyperparameters for our reinforcement learning models were similar to the ones mentioned in [2]. Specifically, we employed a discount factor γ of 0.99997 to balance the trade-off between immediate and future rewards. Additionally, we set a learning rate η of 10^{-4}, which controls the step size of the weight updates during the optimization process.

To facilitate the learning process, we implemented mini-batch learning with an experience replay memory capable of storing 10^6 experiences. Each experience consists of the current state, the action taken, the obtained reward, the subsequent state, and a boolean terminal flag indicating if the episode has reached its termination condition. We sampled mini-batches of 100 experiences to update the model during training.

We adopted an ε-greedy exploration strategy, which balances exploration and exploitation. The ε value decreases linearly with a decrement rate of 10^{-6} and has a minimum threshold of 0.01 to ensure continued exploration. This adaptive approach enables the agent to explore the environment initially and gradually shift towards exploiting the learned knowledge.

Learning took place at every time step throughout the training process, ensuring continuous adaptation and improvement of the agent's performance. The target network, responsible for generating stable target values during raining, was updated every 1,000 learning iterations to maintain consistency with the primary network. This approach mitigates the risk of oscillations and divergence during training.

4.2 Analysis

4.2.1 Scalability

Scalability is a crucial aspect of collective transport, as it directly influences the ability to transport objects of various sizes, larger or smaller than those encountered during training. To investigate this property, we trained two models for each method, one consisting of 4 robots and the other of 8 robots. We evaluated the performance of these models in configurations involving 2, 4, 6, 8, and 10 robots. Each model was subjected to 500 randomly generated test scenarios, with the success criterion being the object's arrival at the goal location. The results are illustrated in Figure 6.

A striking observation from the results is the inferior performance of DQN compared

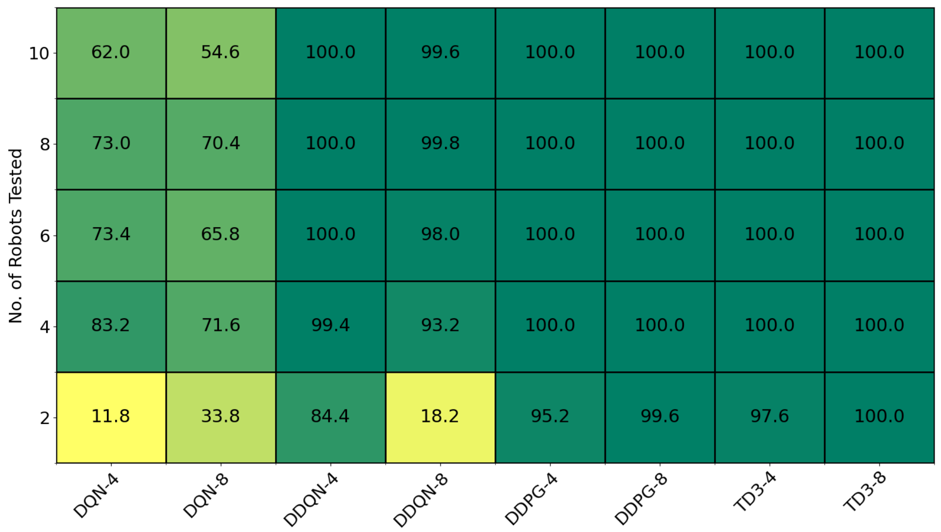


Figure 6: Scalability study results visualized as a heat map: The x-axis displays trained models along with their designated number of robots, while the y-axis indicates the number of robots tested with each model. The color spectrum ranges from yellow for worse scores, through light green for mediocre scores, to green for good scores.

to the other methods. DQN exhibits a significant decline in performance when scaling down to 2 robots and also experiences a drop in performance when scaling up. We hypothesize that this issue arises from DQN's inherent overestimation of Q-values, which renders a trained model unable to adapt effectively to a slightly modified configuration. In contrast, DDQN demonstrates the capacity to scale to a higher number of robots, although its performance when scaling down, especially in the case of the model trained with 8 robots, is also hindered. These findings suggest that value-based methods can scale to handle larger degrees of freedom than encountered during training, but struggle to adapt to under-actuated systems.

Both DDPG and TD3 exhibit robust performance with respect to scalability. However, a minor performance drop-off is observed when scaling down to 2 robots, which could be attributed to DDPG's tendency to overestimate state-action values.

Based on our analysis, we infer that DQN and DDQN, the value-based methods, face challenges when scaling to different configurations, with DQN performing poorly in most cases. DDPG and TD3, actor-critic methods, demonstrate more consistent performance when scaling, making them better candidates for scalable collective transport tasks.

4.2.2 Resilience

We explore the robustness of our models against robot failures through an attrition analysis, examining the consequences of losing up to 25%, 50%, and 75% of robots during the evaluation process. To prevent over-fitting on a specific number of failures, we randomly select the number of failures that will occur at the beginning of each episode, ranging from a minimum of 0 to the set theoretical maximum. For example, if we are training a model with 8 robots and allow for a 50% failure rate, each episode can have at most 4 failures, but can also have fewer. A failure is categorized as complete loss of power, which causes the gripper to release and the robot to become detached from the object being transported, rendering it unable to exert any further influence on the object.

Once a robot is determined to fail, we randomly generate the time-step at which the failure will occur. We trained each method with 4 and 8 robots and with 0% and 50% robot failures. Figure 7 presents our results, where the top and bottom plots correspond to 4 and 8 robots, respectively. Each plot contains two entries per method: one with 0% and the

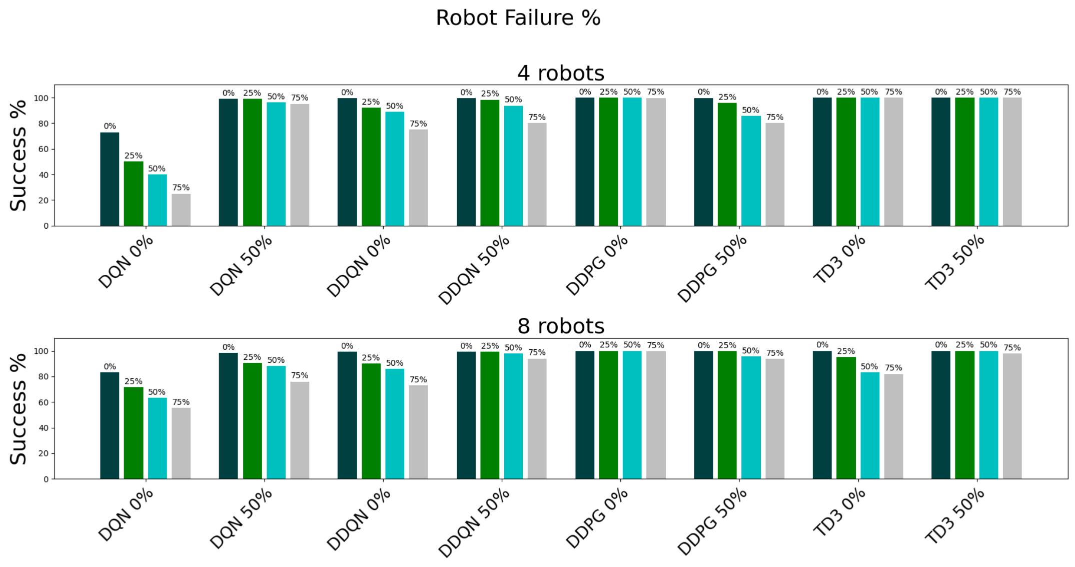


Figure 7: Resilience study results: The top and bottom plots depict models trained on 4 and 8 robots, respectively. The x-axis displays sets of 4 bars, with each set signifying a model trained under 0% or 50% failure conditions. Within these sets, the 4 individual bars represent tests with failure rates from 0% to 75%. The y-axis indicates the success rate, where success is defined by reaching the goal.

other with 50% allowed failures during training. Within each bar set, we display the results from testing on 0%, 25%, 50%, and 75% allowed failures.

Actor-critic methods, regardless of whether they are trained with failures or not, outperform value-based methods, particularly as the failure percentage increases. Among value-based methods, models trained with failures demonstrate better performance than those without. The actor-critic results are less definitive, but still exhibit resilience to failure across all configurations. While failure testing provides valuable insights into resilience, it also enables us to examine non-symmetrical systems. Our results suggest that actor-critic methods may be effectively extended to non-symmetrical transport conditions, whereas value-based methods could struggle in such scenarios.

Based on our resilience analysis, we infer that actor-critic methods, such as DDPG and TD3, demonstrate greater resilience to robot failures compared to value-based methods, like DQN and DDQN. This resilience makes actor-critic methods more suitable for applications that require robust performance in the presence of failures.

4.2.3 Obstacle Avoidance

Obstacle avoidance constitutes a critical aspect of any collective transport algorithm. In this study, we investigate model adaptability by introducing two types of obstacles, as depicted in Figure 2.

Cylinder Obstacles: These obstacles, with dimensions identical to the transported cylinder, are randomly generated within the area between the starting positions of the robots and the

Table 1: Success Rates with Cylinder Obstacles

Obstacles	DQN-0	DQN-2	DDQN-0	DDQN-2	DDPG-0	DDPG-2	TD3-0	TD3-2
2	64.4%	73.2%	93.6%	90.4%	90.2%	91.4%	92.4%	91%
4	48.8%	56.2%	79.8%	81.2%	83.2%	86.4%	84%	82%

target location. We train the models in environments with 0 and 2 obstacles and examine their effectiveness in environments containing both 2 and 4 obstacles. Table 1 displays our results, indicating that DQN-0 represents DQN trained in an obstacle-free environment, while DQN-2 refers to DQN trained in a setting containing 2 obstacles. All methods demonstrate a decline in performance in the 4-obstacle environment compared to the 2-obstacle environment. This deterioration can be attributed to the increased obstacle density within the environment. As more obstacles are added, the probability of introducing non-convex obstacles rises due to the close proximity between two cylinder obstacles. Non-convex obstacles pose a significant challenge, as evidenced in the literature [32].

DQN underperforms in both training configurations and in both obstacle environments, exhibiting a drop in performance as the number of obstacles increases. DDQN, DDPG, and TD3 display relatively similar performance. Figure 8 illustrates trajectories from the 4-obstacle environment. The top plot highlights the difference in learned behaviors: DQN and TD3 maneuver towards the top of the environment as they approach the goal, whereas DDQN and DDPG gravitate towards the bottom. This divergent behavior only emerges in environments with obstacles; in all other environments, the methods exhibit similar strategies.

We also observe variations in trajectory smoothness, with DQN displaying the least

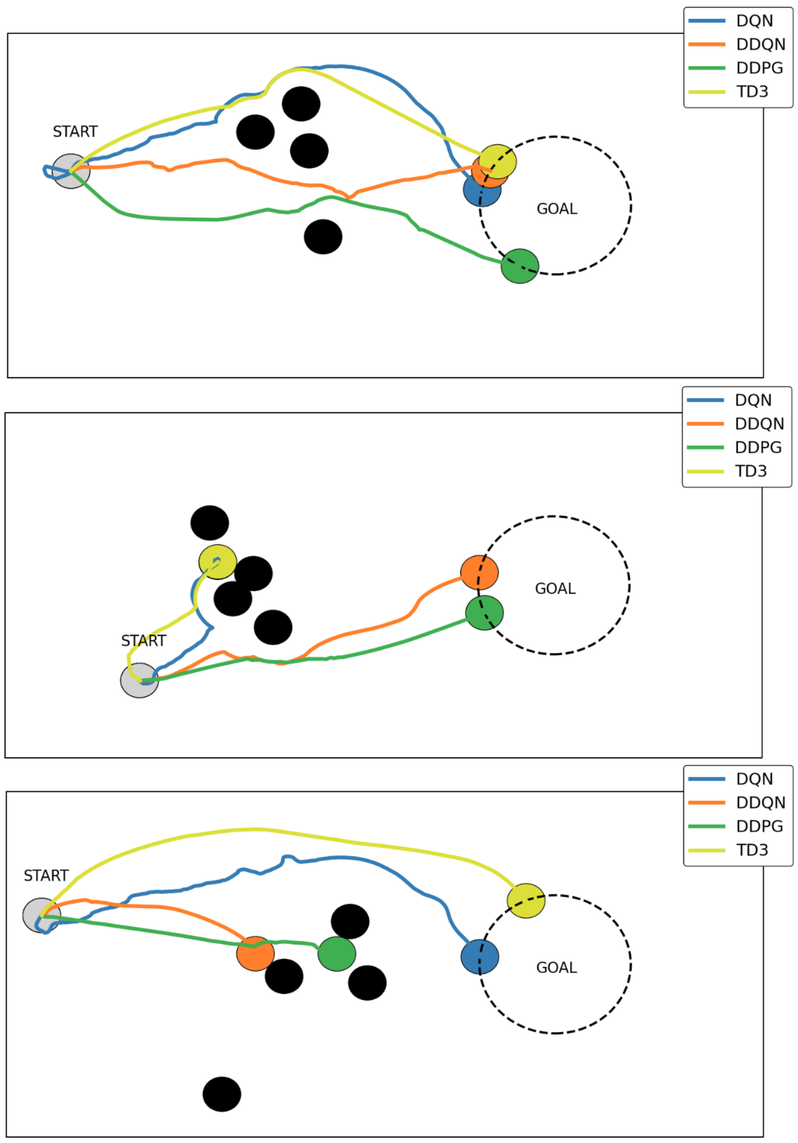


Figure 8: Best neural network trajectories trained with 2 obstacles and tested with 4: Top plot displays successful trials for each learning algorithm, with DQN and TD3 opting for strategies moving above obstacles, while DDQN and DDPG choose strategies navigating below. Middle plot demonstrates failed attempts with DQN and TD3, and the bottom plot illustrates failures for DDQN and DDPG.

smoothness, followed by DDQN, DDPG, and TD3. This implies an increased level of implicit coordination among the robots. Less smooth trajectories indicate disagreement in obstacle trajectory between robots, leading them to apply forces in different directions.

Table 2: Success Rates with Gate Obstacle

DQN	DDQN	DDPG	TD3
72.4%	74.4%	77%	76.4%

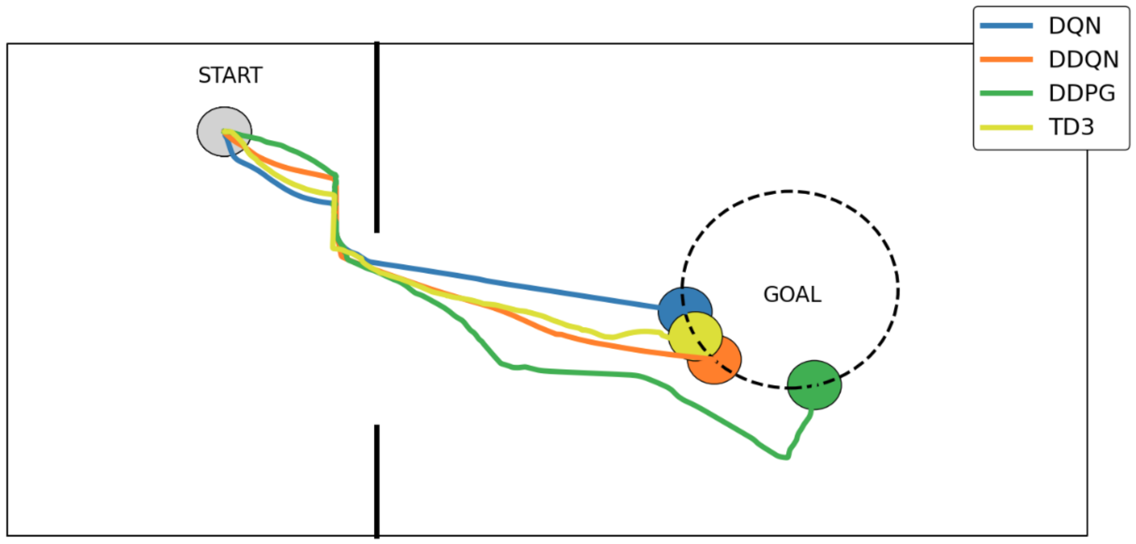
In contrast, smoother trajectories suggest that all robots apply comparable forces on the object by adjusting their wheel speeds in a similar manner, resulting in a higher level of implicit coordination. The middle and bottom charts in Figure 8 depict examples of a non-convex obstacle created by the proximity of two cylinder obstacles being too small for the robots to pass through, yet large enough that the robots cannot pass over it. This type of obstacle poses a challenge for all methods, as none of them are able to handle it effectively.

Gate Obstacle: The gate obstacle presents a more challenging scenario, as the robots must identify an opening in the wall structure to reach the goal. Without the ability to explicitly communicate, a single robot sensing the opening may not guarantee a successful trial. The gate obstacle introduces an additional layer of complexity to the task, as the optimal policy is no longer solely focused on moving towards the goal. The models must now learn to follow the wall in order to find the opening and subsequently locate the goal. Table 2 shows the performance of the algorithms when presented with the gate obstacle. DQN and DDQN achieved success rates of 72.4% and 74.4% respectively, while DDPG and TD3 performed better, reaching the goal 77% and 76.4% of the time, respectively. Figure 9 demonstrates the behaviors learned in the gate obstacle environment.

In the top chart, we observe a distinct wall-following behavior exhibited by all the methods. DQN adopts the most direct path to the goal after navigating the obstacle, which can be attributed to a naive policy of constantly moving towards the goal. DDQN employs a similar policy. Although this policy may prove effective in some configurations, it fails when the gate opening is located on the opposite side of the midpoint of the goal relative to the robots. This limitation is evident in the bottom plot of Figure 9. TD3 and DDPG are capable of continuing in the upward direction until they find the opening, whereas DQN and DDQN halt at the midpoint of the goal.

Based on the analysis provided above, we infer that obstacle avoidance remains a challenging aspect of collective transport algorithms. The increased obstacle density, particularly non-convex obstacles, leads to a decline in performance across all methods. However, certain methods, like DDPG and TD3, show more promising results in terms of adaptability and implicit coordination. Future research could explore new strategies and algorithms to improve obstacle avoidance, especially in the case of non-convex obstacles, to enhance the overall performance of collective transport systems.

We can also infer that the gate obstacle environment highlights the differences in the methods' adaptability and learning capabilities. DDPG and TD3 demonstrate better performance in finding the opening and reaching the goal, while DQN and DDQN struggle in certain configurations due to their naive policies. Developing more sophisticated policies that take into account the spatial complexity of the environment could potentially improve the success rates of collective transport systems when faced with gate obstacles.



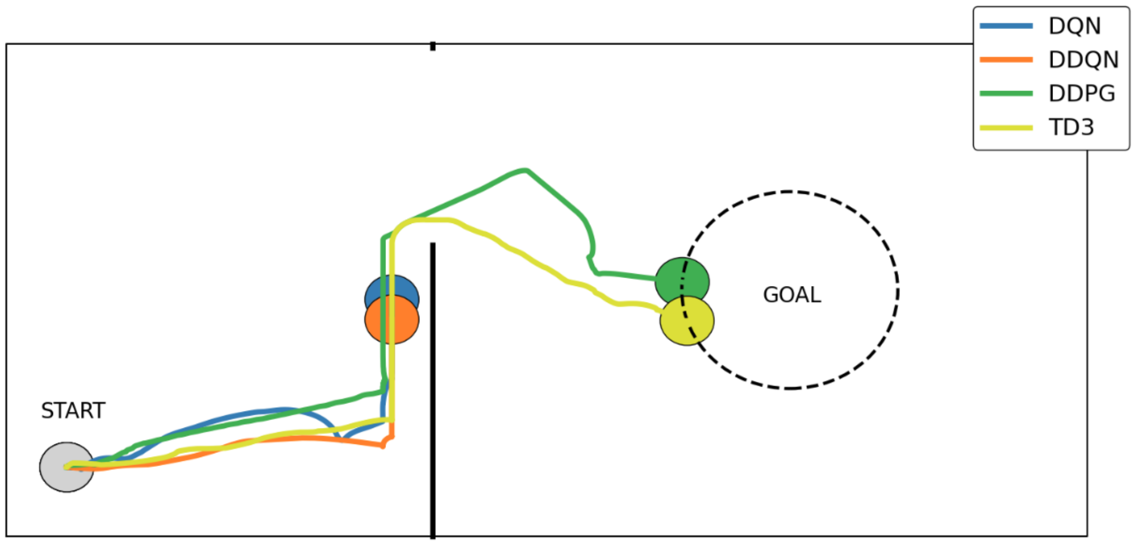


Figure 9: Best neural network trajectories for gate obstacle training: Top plot illustrates wall-following behavior to navigate the obstacle and reach the goal, while the bottom plot highlights the differences between value-based and actor-critic methods. Value-based methods struggle to overcome the obstacle when the opening is past the goal's midpoint opposite the starting position.

Chapter 5

Conclusion

5.1 Summary

We presented a study of four well known RL algorithms applied to aggregates of minimalistic robots which address the problem of non-stationarity in MARL with the help of implicit communication. We show that this is a viable method to solve the problem of multi-agent collective transport, and demonstrate this insight in multi-agent collective transport scenario, in which the robots are physically constrained to the object that needs to be moved.

The different control strategies that emerged from the algorithms are compared and analyzed, showing the ability to scale, be resilient to failures, and coordinate to avoid obstacles with different levels of success. The experiments give weight to our initial hypothesis, which states that implicit communication is sufficient for robot aggregates to achieve coordination and perform collective transport, even in the presence of obstacles and robot failures. We also show that providing the robots with minimal information in the form of accumulated partial local observations of the aggregate to predict the future state of the object improves the performance and success rate for achieving collective transport. The analysis of the emergent behaviours which drive the coordination is compelling and future work will aim to develop it further.

5.2 Future Work

In this paper, we only considered objects whose shapes were radially symmetric. But this method could be expanded to support shapes that are arbitrary, as well as deformable. In our experiments, the objects were completely rigid, which lends a certain type of deterministic property to the system when forces act on it. The robots achieve a sense of perfect implicit communication because of this rigidity, as they are able to feel the forces of other robots distinctly, and do not have to worry about modelling the inherent physics that the elasticity and rigidity of an object can introduce into the system. Future work would involve working with deformable objects to introduce an improved understanding of the object dynamics for the robots, which could lead to better and more complex control strategies.

The work can be expanded by benchmarking other RL algorithms as well, adding better policy based and actor-critic methods such as Trust Region Policy Optimization (TRPO), Proximal Policy Optimization (PPO), Soft Actor-Critic (SAC), and Asynchronous Advantage Actor Critic (A3C), and comparing their performances with the ones studied in this thesis. Investigating the state prediction module for the object further is imperative as well, and trying out different combinations of RL models to evaluate which ones work the best would be an interesting ablation study.

Finally, this methodology could be applied to other types of robot aggregates and tasks beyond collective transport, such as modular robot platforms, collective motion, and multi-robot foraging. The concept of solving non-stationarity through implicit communication, as well as using partial aggregated knowledge is one that definitely has real world applications and warrants deeper studies. Solving the problem of multi-agent RL and transferring the methodology into the real world, and onto real robots, is one that should be pursued aggressively as it could have positive implications in the field of swarm robotics and autonomous vehicles.

Bibliography

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

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[3] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[4] Víctor Uc-Cetina, Nicolás Navarro-Guerrero, Anabel Martín-González, Cornelius Weber, and Stefan Wermter. Survey on reinforcement learning for language processing. *CoRR*, abs/2104.05565, 2021.

[5] Angelos Filos, Panagiotis Tigkas, Rowan McAllister, Nicholas Rhinehart, Sergey Levine, and Yarin Gal. Can autonomous vehicles identify, recover from, and adapt to distribution shifts? In *International Conference on Machine Learning*, pages 3145–3153. PMLR, 2020.

[6] Christopher Amato, George Konidaris, and Leslie P Kaelbling. Modeling and Planning with Macro-Actions in Decentralized POMDPs. *Journal of Artificial Intelligence Research*, 64:817–859, 2019.

[7] Jakob N. Foerster, Yannis M. Assael, Nando de Freitas, and Shimon Whiteson. Learning to Communicate with Deep Multi-Agent Reinforcement Learning. *Advances in neural information processing systems*, 29, May 2016. arXiv: 1605.06676.

[8] Jimmy Wu, Xingyuan Sun, Andy Zeng, Shuran Song, Szymon Rusinkiewicz, and Thomas Funkhouser. Spatial Intention Maps for Multi-Agent Mobile Manipulation. *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8749–8756, 2021.

[9] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 66–83. Springer, 2017.

[10] W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrun. Collaborative multi-robot exploration. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 1, pages 476–481 vol.1, 2000.

[11] Chao Yu, Xin Wang, Xin Xu, Minjie Zhang, Hongwei Ge, Jiankang Ren, Liang Sun, Bingcai Chen, and Guozhen Tan. Distributed multiagent coordinated learning for autonomous driving in highways based on dynamic coordination graphs. *Ieee transactions on intelligent transportation systems*, 21(2):735–748, 2019.

[12] Kevin J. McGowan and Glen E. Woolfenden. A sentinel system in the florida scrub jay. *Animal Behaviour*, 37:1000–1006, 1989.

[13] Robert Heinsohn and Craig Packer. Complex cooperative strategies in group-territorial african lions. *Science*, 269(5228):1260–1262, 1995.

[14] Sumitra Ganesh, Nelson Vadori, Mengda Xu, Hua Zheng, Prashant Reddy, and Manuela Veloso. Reinforcement learning for market making in a multi-agent dealer market, 2019.

[15] Kevin R. McKee, Edward Hughes, Tina O. Zhu, Martin J. Chadwick, Raphael Koster, Antonio Garcia Castaneda, Charlie Beattie, Thore Graepel, Matt Botvinick, and Joel Z. Leibo. A multi-agent reinforcement learning model of reputation and cooperation in human groups, 2023.

[16] Peter Stone, Gal Kaminka, Sarit Kraus, and Jeffrey Rosenschein. Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, pages 1504–1509, 2010.

[17] Hado van Hasselt, Arthur Guez, and David Silver. Deep Reinforcement Learning with Double Q-Learning. *Proceedings of the AAAI conference on artificial intelligence*, 30(1), 2016.

[18] 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:1509.02971 [cs, stat]*, 2015. arXiv: 1509.02971.

[19] Scott Fujimoto. Addressing Function Approximation Error in Actor-Critic Methods. page 10.

[20] Lin Zhang, Yufeng Sun, Andrew Barth, and Ou Ma. Decentralized Control of Multi-Robot System in Cooperative Object Transportation Using Deep Reinforcement Learning. *IEEE Access*, 8:184109–184119, 2020.

[21] Ying Wang and Clarence De Silva. Multi-robot Box-pushing: Single-Agent Q-Learning vs. Team Q-Learning. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3694–3699, Beijing, China, October 2006. IEEE.

[22] Mehdi Rahimi, Spencer Gibb, Yantao Shen, and Hung Manh La. A Comparison of Various Approaches to Reinforcement Learning Algorithms for Multi-robot Box Pushing. In Hamido Fujita, Duy Cuong Nguyen, Ngoc Pi Vu, Tien Long Banh, and Hermann Horst Puta, editors, *Advances in Engineering Research and Application*, volume 63, pages 16–30. Springer International Publishing, Cham, 2019. Series Title: Lecture Notes in Networks and Systems.

[23] Gyuho Eoh and Tae-Hyoung Park. Cooperative Object Transportation Using Curriculum-Based Deep Reinforcement Learning. *Sensors*, 21(14):4780, July 2021.

[24] M Bonani, V Longchamp, Stéphane Magnenat, Philippe Rétornaz, D Burnier, G Roulet, F Vaussard, H Bleuler, and F Mondada. The marXbot, a miniature mobile robot opening new perspectives for the collective-robotic research. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4187–4193, Taipei, October 2010. IEEE.

[25] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*, pages 1–8, Montreal, Quebec, Canada, 2009. ACM Press.

[26] Abien Fred Agarap. Deep Learning using Rectified Linear Units (ReLU), February 2019. arXiv:1803.08375 [cs, stat].

[27] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, January 2017. arXiv:1412.6980 [cs].

[28] Yann LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–50. Springer, 2002.

[29] Carlo Pinciroli, Vito Trianni, Rehan O'Grady, Giovanni Pini, Arne Brutschy, Manuele Brambilla, Nithin Mathews, Eliseo Ferrante, Gianni Di Caro, Frederick Ducatelle, Mauro Birattari, Luca Maria Gambardella, and Marco Dorigo. ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intelligence*, 6(4):271–295, 2012.

[30] Carlo Pinciroli and Giovanni Beltrame. Buzz: An extensible programming language for heterogeneous swarm robotics. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3794–3800, Daejeon, South Korea, October 2016. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).

[31] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[32] Hamed Farivarnejad and Spring Berman. Multirobot Control Strategies for Collective Transport. *Annual Review of Control, Robotics, and Autonomous Systems*, 5(1):annurev–control–042920–095844, May 2022.