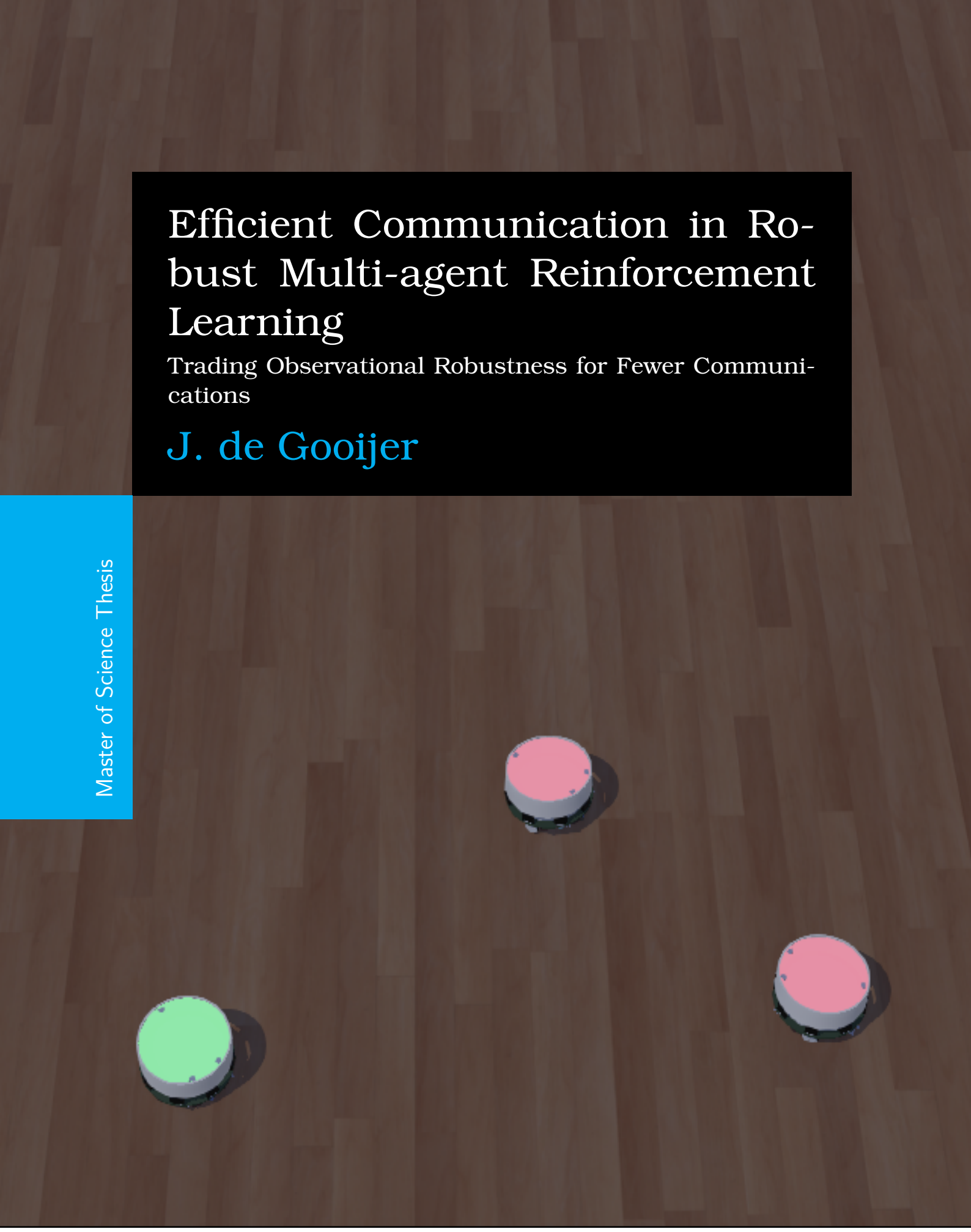
Efficient Communication in Robust Multi-agent Reinforcement Learning

Trading Observational Robustness for Fewer Communications

J. de Gooijer

Master of Science Thesis



TU Delft Delft University of Technology

Delft Center for Systems and Control

Delft Department of Cognitive Robotics

Efficient Communication in Robust Multi-agent Reinforcement Learning

Trading Observational Robustness for Fewer Communications

Master of Science Thesis

For the degree of Master of Science in Systems and Control, Robotics at Delft University of Technology

J. de Gooijer

July 29, 2023

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Abstract

Reinforcement learning, especially deep reinforcement learning, has made many advances in the last decade. Similarly, great strides have been made in multi-agent reinforcement learning. Systems of cooperative autonomous robots are increasingly being used, for which multi-agent reinforcement learning can be used as a training method. However, the curse of dimensionality remains a problem for the computational speed of the learning algorithm and the bandwidth of communication channels. This research will focus mainly on reducing the problem of overloading communication channels by trying to reduce the number of communications. This is possible since it is usually unnecessary for every agent to communicate with every other agent constantly.

To do this, we use some ideas by Daniel Jarne Ornia. The first is to reduce communications in a multi-agent reinforcement learning system by treating it as an event-triggered control problem. This method uses so-called robustness surrogates as an equivalent to a Lyapunov function to determine if a communication can be skipped without decreasing the performance more than some tolerance. The second is a method to increase the observational robustness of a policy by using lexicographic reinforcement learning.

We aim to combine these ideas and trade the additional observational robustness for decreased communications. We also want to test whether additional observational robustness can help mitigate the sim-to-real gap. We implement this method for the multi-agent deep deterministic policy gradient algorithm and perform tests on a variant of the predator-prey domain in increasingly more realistic simulations.

We found that the combination of this robust policy and the robustness surrogates method does enable the agents to achieve the same return while communicating less. Unfortunately, our research shows that the observational robustness obtained using lexicographic reinforcement learning, does not help mitigate the sim-to-real gap.

Table of Contents

Preface **xi**

1 Introduction **1**
1-1 Related Work . . . 2
1-2 Main Contribution . . . 3
1-3 Nomenclature and Notation . . . 4
1-4 Report Structure . . . 4

2 Preliminaries **5**
2-1 Reinforcement Learning . . . 5
2-1-1 Policy Gradient . . . 6
2-1-2 Actor-Critic Architecture . . . 7
2-1-3 Deep Reinforcement Learning . . . 7
2-2 Multi-Agent Reinforcement Learning . . . 7
2-3 Observational robustness . . . 8
2-4 Event-Driven Interactions . . . 8
2-4-1 Introduction to Event-Triggered Control . . . 8
2-4-2 Event-Driven Interactions . . . 9

3 Environment and Algorithm **11**
3-1 Multi-Agent Deep Deterministic Policy Gradient Algorithm . . . 11
3-1-1 Convergence Issues . . . 12
3-2 Environment . . . 13
3-2-1 Reward function . . . 14
3-2-2 Environment Progression . . . 16

4 Extension of the Robustness Surrogates to the MADDPG algorithm **21**
4-1 Original Algorithm and Shortcomings . . . 21
4-2 Algorithm Adaptations . . . 23
4-3 Experimental Validation of Implementation . . . 27
4-3-1 Validation Conclusion . . . 30

5 Lexicographic Optimisation **31**
5-1 Original Algorithm . . . 31
5-2 Algorithm Adaptations . . . 33
5-3 Experimenal Validation of Implementation . . . 34
5-3-1 Validation Conclusion . . . 36

6 Software Architecture **37**
6-1 Flow and Architecture . . . 37
6-2 Class Diagram . . . 37
6-3 Neural Networks . . . 38
6-4 Hyperparameters . . . 39

7 Experimental Results **41**
7-1 Experiments . . . 41
7-2 Results . . . 42
7-2-1 Experiments in Environment A . . . 42
7-2-2 Experiments in Environment B . . . 44
7-2-3 Experiments in Environment C . . . 46
7-2-4 Comparison of return increase per environment . . . 48

8 Conclusion **51**
8-1 Discussion . . . 51
8-2 Conclusion . . . 52
8-3 Future Work . . . 52
8-3-1 Improvements to this research . . . 52
8-3-2 Extensions . . . 53

Bibliography **55**

Glossary **59**
List of Acronyms . . . 59
List of Symbols . . . 59

Index **63**

List of Figures

2-1 Diagram of interaction between the agent and the environment. 5

2-2 Taxonomy of popular Reinforcement Learning (RL) algorithms. Figure from [1]. . 7

2-3 Simple schematic of an Event-Triggered Control (ETC) structure. Figure from [2]. 9

2-4 Simple schematic of EDI structure for state communication. 10

3-1 Overview of the multi-agent actor-critic architecture of [3]. Here, o is an observation, a is an action, π is a policy, and Q is a Q-function. In Multi-Agent Deep Deterministic Policy Gradient (MADDPG), π and Q are neural networks. Figure from [3]. 13

3-2 Figures of the simple-adversary and simple-tag environments. Figures from [3]. . 14

3-3 Plot of communications and adversary returns for different values of ζ using the simple-tag environment with its original reward. 15

3-4 Screenshot of the MPE simple-tag environment, with two red adversaries, one chasing the green agent, the other mirroring its partner's movements. 17

3-5 Graphic top-view representation of a robot and some important variables, such as the robot coordinates and goal coordinates, the definition of the coordinate system and the relevant robot parameters. 20

3-6 Screenshot of the Webots environment, with two red adversaries, one chasing the green agent, the other mirroring its partner's movements. 20

4-1 Visualisation of the use of the robustness surrogates, from [4]. 22

4-2 Diagram of state communication between two agents using robustness surrogates. 25

4-3 Decision process for agent 1 whether to communicate its updated state to agent 2. If $\zeta > \zeta_{\text{th}}$, then agent 2's knowledge of agent 1's state is updated. 26

4-4 Visualisation of the increase of ζ for a larger difference in states. 28

4-5 Graph displaying the change in adversary return and number of communications per episode for varying ζ. 30

5-1 Graphical representation of the Lexicographically Robust Reinforcement Learning (LRRL) algorithm. We can see a visualisation of the inclusion theorem, and the policy converging to a maximally robust policy, while still being within the tolerance from the optimal policy. Figure from [5]. 33

6-1 Diagram of the flow and architecture of the full algorithm. 38

6-2 Simplified class diagram of the software. 39

7-1 Experimental results for the increase of ζ for states further apart for environment A. 42

7-2 Experimental results for the adversary return and number of communications for environment A. 43

7-3 Adversary return versus the number of communications for environment A. . . . 44

7-4 Experimental results for the increase of ζ for states further apart for environment B. 45

7-5 Experimental results for the adversary return and number of communications for environment B. 46

7-6 Adversary return versus the number of communications for environment B. . . . 46

7-7 Experimental results for the increase of ζ for states further apart for environment C. 47

7-8 Experimental results for the adversary return and number of communications for environment C. 48

7-9 Adversary return versus the number of communications for environment C. . . . 48

7-10 Increase in adversary return obtained by LRRL versus number of communications. 49

List of Tables

4-1 Table to indicate the limit in the loss of return, and the actual loss of return for different values of ζ. 29

5-1 Adversary return comparison of Vanilla policy and LRRL policy. 35

6-1 Table of important parameters used for this research. 40

7-1 Baseline values, with Event-Driven Interactions (EDI) mode deactivated for environment A. 43

7-2 Experimental results for the sim-to-real gap experiment for environment B. . . . 44

7-3 Baseline values, with EDI mode deactivated for environment B. 45

7-4 Experimental results for the sim-to-real gap experiment for environment C. . . . 47

7-5 Baseline values, with EDI mode deactivated for environment C. 47

List of Algorithms

1 Multi-Agent Deep Deterministic Policy Gradient for N Agents [3] 12
2 Reward Function 16
3 Simple Controller 19
4 Computation of Robustness Indicator [4] 22
5 Self-triggered State Sharing [4] 23
6 New Computation of Robustness Indicator 24
7 New Self-triggered State Sharing 24
8 Lexicographically robust policy gradient [5] 34
9 New actor loss when using LRRL 35

Preface

Dear reader,
You are reading my master's thesis, the culmination of 6 years of study. After three years of bachelor's degree in Mechanical Engineering and three years of doing a double master's degree in Robotics and Systems & Control at the TU Delft, this is the final result. My academic journey so far has had a lot of ups and downs, but I have learned more than I ever thought possible and met some fantastic people along the way.

When the time came to pick my thesis topic, I knew I wanted something relevant to both my fields of study. When one of my supervisors suggested multi-agent reinforcement learning, I was immediately enthusiastic. I have always been excited by algorithms inspired by biology, such as reinforcement learning, where agents learn by trial and error, or genetic algorithms for optimisation. Before starting this thesis, my knowledge of reinforcement learning was minimal, let alone multi-agent reinforcement learning, with additional constraints to communication!

Nonetheless, I was excited to take on this challenge and diligently delved into it. Everybody experiences good and bad phases during their thesis, which was true for me. I had my fair share of late-night coding sessions and pits of despair when the results of running a 3-day script were useless. Looking back now, though, it was not so bad, and I mostly enjoyed the relentless quest for knowledge.

A lot of people helped me through this thesis, first and foremost my supervisors, Dr. Manuel Mazo Espinosa, Dr. Javier Alonso Mora and Dr. Daniel Jarne Ornia, with special thanks to Dr. Daniel Jarne Ornia, who helped me with a lot of the technical part of the project and sparred with me when I needed it and gave me some invaluable tips. I also want to thank Dr. Jens Kober for being on my graduation committee. Finally, I want to thank my friends and family for their love and support.

And, of course, thank you, my reader. I hope you will enjoy reading this thesis, may you find it useful and provide you with some insights.

Delft, University of Technology — J. de Gooijer
July 29, 2023

Chapter 1

Introduction

Reinforcement Learning (RL) is a rapidly growing field that has gained a lot of interest in recent decades. Similarly, the multi-agent variant of reinforcement learning has also been gaining popularity. Various algorithms have been developed to tackle the problems of scalability and non-stationarity of the multi-agent reinforcement learning system, such as Multi-Agent Deep Deterministic Policy Gradient (MADDPG) proposed by Lowe [3]. When multiple agents have to cooperate to solve a task and gain a reward, it is often necessary that these agents can communicate with each other. This communication can be done periodically, on a clock, or a certain event can trigger it. Communicating only when necessary for maintaining performance can lead to a reduction in communications. Thus, the demand on communication bandwidth, resources, and energy consumption will be lessened, which might be desirable or even necessary when many agents are involved. Also, from an environmental viewpoint, it is always desirable to lessen the energy requirements of a system. However, this reduction in communication will cause some uncertainties for the agents concerning the state or location of the other agents. This means that the algorithm that controls the agents needs to be more robust to these uncertainties, and there will be a trade-off between the number of communications and the robustness of the control algorithm.

There are two papers written by Daniel Jarne Ornia which, when combined, concern both of these problems: Robust Event-Driven Interactions in Cooperative Multi-Agent Learning [4] and Observational Robustness and Invariances in Reinforcement Learning via Lexicographic Objectives [5]. The first of these, [4], concerns maintaining the robustness and performance guarantees of the control algorithm while reducing communications through the use of robustness surrogates. The second, [5], concerns the increase of observational robustness.

This thesis aims to combine the ideas from these papers and test if the increase in observational robustness will enhance the performance of the event-driven interactions algorithm, and to what extent. We aim to perform these tests in a realistic environment like the Webots simulator. This will also allow us to test if the increased observational robustness can help mitigate the sim-to-real gap, regardless of the event-driven interactions.

In relation to these goals, we can define our two main hypotheses:

Hypothesis 1. *Policies that are more observationally robust due to the use of lexicographic reinforcement learning will be able to maintain the same performance while using less communications than policies that are trained in a traditional manner.*

Hypothesis 2. *Policies that are more observationally robust due to the use of lexicographic reinforcement learning will perform better when moving from a simulated environment to a more realistic environment than policies that are trained in a traditional manner.*

1-1 Related Work

Reinforcement Learning The work by Dong et al. [1] is a good and fairly complete introductory work on single-agent deep RL, and discusses many popular single-agent algorithms, both basic ones like Q-learning and deep learning algorithms such as Deep Deterministic Policy Gradient (DDPG).

Most single-agent RL algorithms are not suitable for multi-agent environments. This is due to the non-stationarity that occurs in a multi-agent Markov Decision Process (MDP), also known as a stochastic game [6]. Another main problem is the curse of dimensionality, where the computational complexity rises exponentially with the number of agents. Multi-Agent Reinforcement Learning (MARL) algorithms have been developed to deal with these issues.

For MARL, [6] gives a good overview of some popular algorithms and a classification. Some other MARL algorithms mentioned in literature include [7], which develops a robust algorithm for state-adversarial MARL, since the perturbations caused by state adversaries cannot be fully described by a Partially Observable Markov Decision Process (POMDP). The authors of [8] develop a multi-agent Q-learning algorithm for general-sum stochastic games with a single Nash equilibrium. An extension of the DDPG algorithm to the multi-agent case is proposed in [3]. For papers on evolutionary learning, we can refer to [9] for a survey of evolutionary learning and MARL in a cooperative setting, or one of the many papers by Cees Verdier on controller synthesis via genetic programming: [10, 11, 12, 13].

Deterministic Policy Gradient Methods We have selected the MADDPG algorithm from [3] for our research. This algorithm is suitable since it is claimed to be a general all-purpose MARL algorithm suitable for mixed environments. Notably, it does not make assumptions about environment differentiability or the communication structure. Moreover, during the decentralised execution, the agents only use local information and their own observations.

Although MADDPG uses both a critic and an actor, it is ultimately a Policy Gradient (PG) algorithm and uses the Policy Gradient Theorem from [14], which discusses PG methods for stochastic policies; $\pi(u|x)$. This policy gradient method is adapted by [15] for the deterministic policy case; $u = \mu(x)$. They claim that the crucial practical difference is that while stochastic policy gradients integrate over both state and action spaces, the Deterministic Policy Gradient (DPG) only needs to integrate over the state space. Thus, DPG possibly requires fewer samples.

The deep learning version of DPG, called DDPG, is developed in [16], for which they use techniques also seen in Deep Q-Network (DQN), presented in [17]. These techniques include using a replay buffer from which to sample transition relations to remove correlations in the observation sequence,

and using additional target networks that update slowly or periodically, limiting the effect of fast-changing parameters due to large gradients. These techniques are meant to mitigate the stability issues that reinforcement learning methods using neural networks as function approximators are known to suffer from. According to [16], DQN can handle high-dimensional state spaces, but only low-dimensional action spaces. Furthermore, DQN can only handle discrete domains. DDPG is claimed to be able to handle both high-dimensional action spaces and continuous domains. MADDPG is the multi-agent extension of the DDPG algorithm and will be further explained in Chapter 3.

Event-Driven Interactions On implementing Event-Driven Interactions (EDI) in MARL systems, multiple attempts have been made, such as [18], where agents learn to communicate in the same way they learn to navigate the environment, through a separate communication Q-function. Using a centralised critic, [19] also learns communication protocols, resulting in targeted, multi-round communication. An efficient policy gradient algorithm using parallel learners that can skip sending an updated policy gradient evaluation is developed in [20]. In [21], agents learn to coordinate (and communicate) in a predefined coordinate structure, specific to each state. Another paper by Daniel Jarne Ornia, [22], implements EDI in a MARL system by using robustness surrogate functions, which are supposed to work analogously to Lyapunov functions in Event-Triggered Control (ETC), of which [23] is an excellent introductory work. This function looks at the loss of the Q-value when not communicating. According to [22], this will guarantee the performance to be within a chosen bound of the optimum. This robustness surrogate method is slightly extended in [4].

Observational Robustness Many attempts have been made to make reinforcement learning more observationally robust. During training, adversarial examples are used in [24] to enable robust learning. In [25], certified adversarial robustness is demonstrated for a DQN system. A trained adversary is used in [26] to destabilise the agent or environment. However, according to [5], most methods lose convergence guarantees and are difficult to verify. They claim this is not a problem with their method, since they set robustness as one of the objectives in a lexicographic multi-objective optimisation. Lexicographic Reinforcement Learning (LRL) was introduced in [27], and [5] used this technique to specifically train a policy for robustness, resulting in Lexicographically Robust Reinforcement Learning (LRRL). Both [5] and [27] use Lagrangian relaxation techniques [28] to make LRL efficient. A proof of convergence is also provided by [27], which is adopted by [5]. Some experiments are performed by [5], to test their algorithm for obtaining observational robustness, using different noise kernels and Proximal Policy Optimisation (PPO) and Advantage Actor-Critic (A2C) as algorithms. They found that their algorithm successfully lowers the robustness regrets, while still being able to guarantee the performance of the chosen algorithm.

1-2 Main Contribution

The main contribution of this research is to combine the efforts of [4] and [5] and test them in a more realistic environment, namely the Webots simulator. We find out whether the LRRL method from [5] can make the EDI method from [4] more effective. In order to do this, we provide an extension to the robustness surrogates method from [4] to be more generally applicable. The

method from [5] has thus far only been applied to single-agent algorithms. Therefore we also provide the implementation of this method for the MADDPG algorithm.

We also verify whether or not the additional observational robustness obtained through the method in [5] will help mitigate the problems caused by the sim-to-real gap.

The complete code for this project can be found in our GitHub repository at [29].

1-3 Nomenclature and Notation

For this report, we will use calligraphic capitals, such as $\mathcal{X}$ and $\mathcal{U}$ to indicate sets or random processes, with the exception of the set of real numbers, for which we will use $\mathbb{R}$. $\varnothing$ indicates the empty set.

We shall use bold letters for vectors, such as $\mathbf{x}$, which are transposed by $(.)^\top$, and small letters for scalars, such as n. We will use capitals for functions, like R, and for the number of elements in a set, i.e. we have a set $\mathcal{A}$ of N agents, and individual agents are indicated by: $A^{(n)},\ n = 1, \ldots, N$. We will use $|.|$ as the absolute value and $||.||_p$ as the p-norm of a vector. When not indicating a specific p-norm, we mean the ℓ_2-norm $||.||_2$.

We will use $\mathbb{P}$ to indicate probability and $\mathbb{E}$ for the expected value.

Unless otherwise indicated, we shall use the subscript for time indication and superscript in parentheses for agent enumeration. For example, the state of agent 1 at time t is $x_t^{(1)}$. We shall use $\hat{}$ to indicate an estimated or outdated value. We shall use $(.)'$ to indicate the next or updated state or value. $(.)'$ is also used to indicate a target network.

1-4 Report Structure

This report will outline the steps taken to prove or disprove our hypotheses. First, we shall discuss some necessary preliminary knowledge in Chapter 2, explaining the basics of RL, ETC and observational robustness. Second, Chapter 3 will elaborate on the chosen algorithm and environment. Chapter 4 and Chapter 5 will describe our adaptations of the robustness surrogate algorithm from [4] and the LRRL algorithm from [5] respectively. The architecture of the full software packet will be discussed in Chapter 6. This implementation was thoroughly tested in some experiments, described in Chapter 7, which also presents the results of these experiments. These results are debated in Chapter 8, which also gives the concluding remarks and possible improvements on the research.

Chapter 2

Preliminaries

This chapter will shortly discuss some preliminary knowledge necessary to understand this thesis. Most of the topics will be covered briefly. Please consult the provided references for a more thorough understanding of the topics.

First, the basics of reinforcement learning will be discussed, the single-agent variant in Section 2-1 and the multi-agent variant in Section 2-2. Sequentially, we will introduce the concepts of observational robustness and event-triggered control, focusing on the special case of event-driven interactions in Section 2-3 and Subsection 2-4-1, respectively.

2-1 Reinforcement Learning

Reinforcement Learning (RL) is the process of an autonomous agent learning to solve a task by receiving reward-based feedback when interacting with the environment. The agent observes its state x and selects an action u based on that information. This action will cause the environment to update the state to x' and return a reward r. This loop is visualised in Figure 2-1.

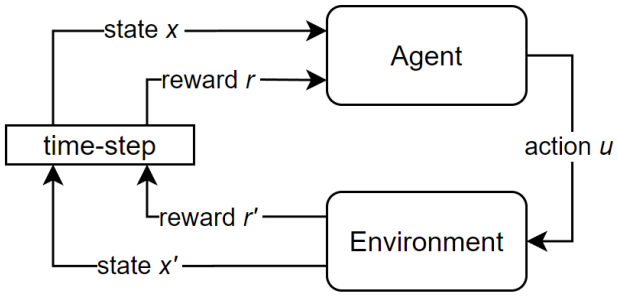


Figure 2-1: Diagram of interaction between the agent and the environment.

This loop and the underlying RL problem can be formalised as a *Markov Decision Process (MDP)*. An MDP is defined by the tuple $(\mathcal{X}, \mathcal{U}, P, R, \gamma)$, where $\mathcal{X}$ is a finite set of states, $\mathcal{U}$ is a set of

actions, $P : \mathcal{U} \times \mathcal{X} \mapsto \mathbb{P}(\mathcal{X})$ is a probabilistic transition relation, mapping the combination of an action in a state to a new state, $R : \mathcal{X} \times \mathcal{U} \times \mathcal{X} \mapsto \mathbb{R}$ is a reward function so that $r = R(x, u, x')$ is the reward obtained when performing action u in state x, which leads to state x', and $\gamma \in [0, 1]$ is the discount factor.

We can now define the cumulative discounted reward, also called the *return*, as $G_t = \sum_{t=0}^{\infty} \gamma^t R(x_t, u_t, x_{t+1})$, where we can see that a higher discount factor puts more relevance on immediate rewards and less relevance on future rewards.

An agent has a *policy* π: $\mathcal{X} \to \mathbb{P}(\mathcal{U})$ that maps the state to the selected action, which can be probabilistic. In that case, we denote $\pi(u \mid x)$ as the probability of selecting action u in the state x. When the policy is deterministic, we denote it as $u = \mu(x)$, a deterministic mapping of state x to action u.

The expected return depends on the state and the policy and is known as the *value function* $V^{\pi}(x) = \mathbb{E}[G_t \mid x_0 = x]$. The value function is a measure of how good any given state is for an agent following policy π. The optimal policy π^* is the policy that maximises the value function: $\pi^* = \text{argmax}_{\pi} \mathbb{E}[G_t]$. Similar to the value function, there is also the action-value function, which represents the quality of an action in a given state while following policy π. This function is better known as the *Q-function*: $Q^{\pi}(x, u) = \mathbb{E}[G_t \mid x_0 = x, u_0 = u]$.

Many algorithms have been developed to solve the RL problem and find a (near)-optimal policy. We shall not discuss algorithms in this report, but readers are encouraged to consult [1] for an overview and explanation of many popular algorithms. We will now discuss some much-used concepts relevant to this research, starting with policy gradient in Subsection 2-1-1, actor-critic in Subsection 2-1-2 and finally, some remarks about deep reinforcement learning in Subsection 2-1-3.

2-1-1 Policy Gradient

Generally speaking, two families of RL algorithms exist: value-based and policy-based. This global taxonomy can be seen in Figure 2-2 from [1]. Within this family of policy-based algorithms is a branch of *Policy Gradient (PG)* algorithms, a popular reinforcement learning approach.

PG algorithms optimise a set of parameters θ of a policy π_θ by maximising the objective function K:

$$\max_{\theta} K(\pi_\theta), \quad K(\pi_\theta) = \mathbb{E}[G_t] \tag{2-1}$$

This maximisation is done through gradient ascent, defining the objective function using the Policy Gradient Theorem from [14] and [15]:

$$\hat{g} = \nabla_\theta K(\pi_\theta) = \int_{\mathcal{X}} \xi^{\pi}(x) \int_{\mathcal{U}} \nabla_\theta \pi_\theta(u \mid x) Q^{\pi}(x, u) \mathrm{d}u \, \mathrm{d}x = \mathbb{E}[\nabla_\theta \log \pi_\theta(u_t|x_t) Q^{\pi_\theta}(x_t, u_t)] \tag{2-2}$$

Where $\hat{g}$ [30] is known as the gradient estimator, and ∇ is the gradient operator. Which leads to the parameter update:

$$\theta_{k+1} = \theta_k + \alpha \hat{g} \tag{2-3}$$

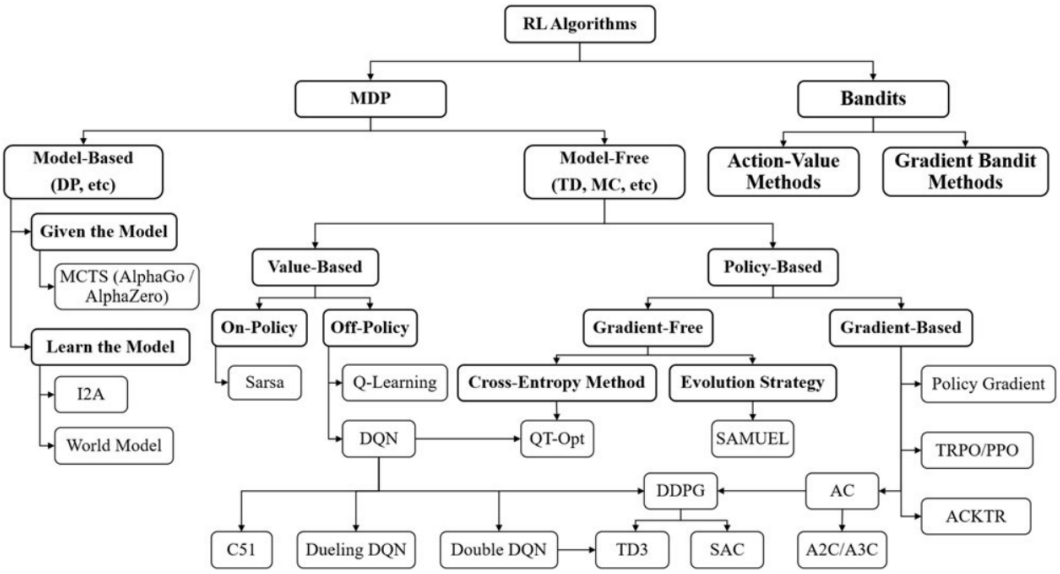


Figure 2-2: Taxonomy of popular RL algorithms. Figure from [1].

2-1-2 Actor-Critic Architecture

The *Actor-Critic (AC)* method, first proposed by [31], combines value-based and policy-based methods. AC algorithms learn functions, or function approximations, of both the policy and Q-function.

2-1-3 Deep Reinforcement Learning

Deep RL is RL using neural networks. This is especially popular for AC methods, using a neural network each for the estimators of the policy and Q-function. RL algorithms using neural networks as function estimators tend to suffer from convergence and stability issues. This issue is solved by [17] for Deep Q-Network (DQN) by using a replay buffer from which to sample transition relations in order to remove correlations in the observation sequence, and by using additional target networks that update slowly or periodically, limiting the effect of fast-changing parameters due to large gradients.

2-2 Multi-Agent Reinforcement Learning

According to [6], the multi-agent case generalisation of an MDP is a *stochastic game*. A stochastic game with N agents can be mathematically described as the tuple $(\mathcal{X}, \mathcal{U}_1, \ldots, \mathcal{U}_N, P, R_1, \ldots, R_N, \gamma)$. Where $\mathcal{U}_n$ is the action chosen by agent n, and R_n is its reward function. $P : \mathcal{U}_1 \times \ldots \times \mathcal{U}_N \times \mathcal{X} \mapsto \mathbb{P}(\mathcal{X})$ is the transition relation.

Stochastic games can be classified as either cooperative, competitive or mixed. A stochastic game is fully *cooperative* if $R_1 = R_2 = \ldots = R_N$, i.e. if all reward functions are the same and the agents have the same goal. A stochastic game is fully *competitive* if $N = 2$ and $R_1 = -R_2$, i.e. there are

two agents with opposite goals [6]. A stochastic game can also be *mixed cooperative-competitive*. A fully cooperative task with a centralized learner reduces back to an MDP, which can be solved with single-agent algorithms. However, when more and more agents are present, the state and action spaces, and thus the problem complexity, will grow exponentially [6]. Therefore, it is often infeasible to solve these types of problems centrally, thus requiring some form of multi-agent algorithm.

We could try to have every agent independently learn a single-agent algorithm, and in fact, this has been done several times, as in [32], where single-agent Q-learning is successfully applied to a decentralised multi-agent setting. However, most single-agent algorithms need a stationary environment to guarantee convergence. Suppose the learning is distributed in a multi-agent setting. In that case, this is equivalent to the environment being non-stationary, since the other agents' response to an action might have changed due to their changing policy. Thus many single-agent RL algorithms will have invalid convergence properties [6]. It is argued by [3] that Q-learning approaches suffer badly from this non-stationarity, and as a result, DQN will not be stable. Policy gradient methods, on the other hand, suffer from very high variance in multi-agent settings. For this reason, multi-agent algorithms have been developed to mitigate this problem.

2-3 Observational robustness

Observational robustness is robustness against uncertainties in the observations of the agent. This means that the observations by the agents are not equal to the true state. In such cases, the underlying MDP is considered a Partially Observable Markov Decision Process (POMDP). To accommodate for this in the mathematical description, the tuples of both the MDP and the stochastic game should be augmented with a set of observations $\mathcal{O}_1, \ldots, \mathcal{O}_N$ for each agent.

The uncertainties can be caused by various sources, such as sensor noise, bad lighting conditions or the sim-to-real gap [33]. Therefore it is desirable to design controllers and policies that can maintain their performance even in the presence of these uncertainties. This is what we commonly refer to as observational robustness.

2-4 Event-Driven Interactions

This section discusses the concept of certain control actions only happening on some triggering event. First, we shall discuss the more generally applied event-triggered control in Subsection 2-4-1, after which we shall explain how to apply that concept to our research in Section 2-4.

2-4-1 Introduction to Event-Triggered Control

Event-Triggered Control (ETC) is the use of a control action only when necessary, usually when some triggering threshold is crossed. The control values are often interpreted in a sample-and-hold fashion between the signals of control actions, meaning that the communicated value is considered constant between communications. Traditionally, ETC is applied to control feedback systems, as explained in [23]. A simple schematic of an ETC structure can be seen in Figure 2-3[2].

Determining a proper triggering condition is crucial for correctly implementing ETC. We want the triggering condition to be strict enough to guarantee performance, but at the same time, not so strict that the amount of signals is the same as when using periodic control. When choosing a triggering condition, it is essential to avoid *Zeno behaviour*, where an infinite amount of triggers occur in a finite amount of time [23]. Most triggering conditions are designed using the *Lyapunov function* [23], which is an indication of the distance between the actual state and the desired state.

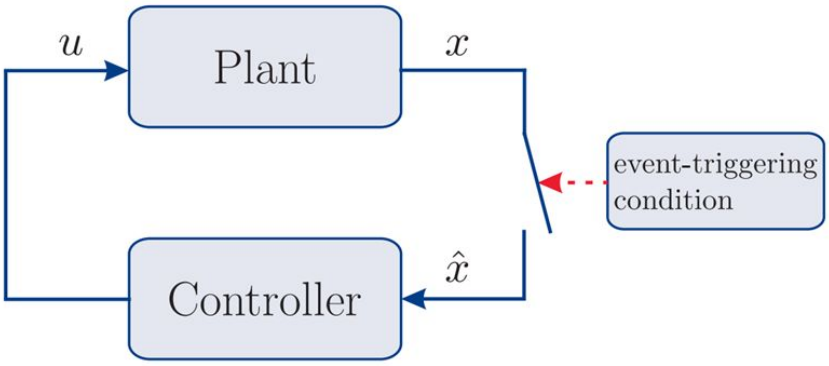


Figure 2-3: Simple schematic of an ETC structure. Figure from [2].

2-4-2 Event-Driven Interactions

Event-Driven Interactions (EDI) can resemble ETC, but with a focus on the communication aspect rather than control actions. Agents often need to exchange information in multi-agent systems to accomplish their tasks effectively. It is easily seen that the number of communications rises very fast when more agents are involved. This means that the communication network will need significant resources in terms of bandwidth and energy. Therefore, it makes sense to want to reduce the number of communications. For most applications, it will not be necessary for all agents to know where every other agent is at all time steps.

Unlike conventional MARL systems where information is shared periodically or at every time-step, using EDI means that data is transmitted to other agents only when necessary. Although it shares similarities with ETC, it deviates from the traditional state-feedback control problem, and no Lyapunov function exists to use as a base for the triggering condition. Instead, [4] introduces robustness surrogates based on the Q-value as an alternative, which we will delve into further in Chapter 4. An example of a communication scheme for state communication in an Multi-Agent Reinforcement Learning (MARL) system can be seen in Figure 2-4.

When the system skips communicating state updates to other agents, it will introduce uncertainties and inaccuracies in the knowledge used by agents to make their decisions. Therefore the policy shall have to be observationally robust against these uncertainties.

This leads to the trade-off of observational robustness versus communications, i.e. the more observationally robust a policy is, the more communications it can miss before the performance significantly suffers.

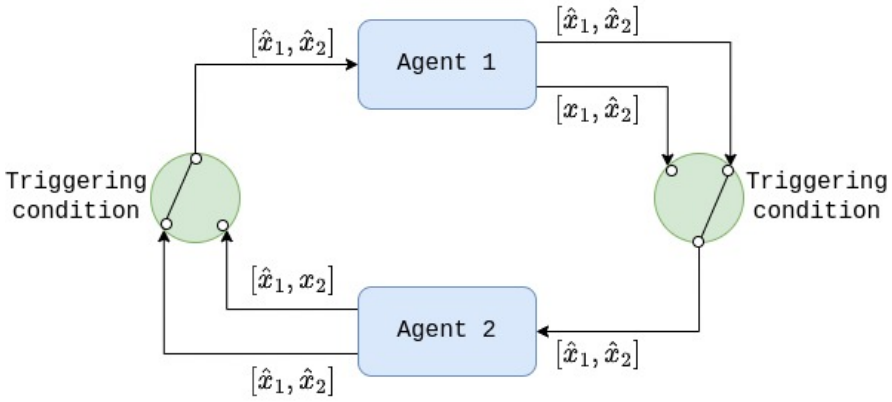


Figure 2-4: Simple schematic of EDI structure for state communication.

Chapter 3

Environment and Algorithm

This chapter will discuss the chosen base algorithm for the Multi-Agent Reinforcement Learning (MARL) system, which was chosen to be Multi-Agent Deep Deterministic Policy Gradient (MADDPG) from [3], as stated in Chapter 1. Multiple open-source implementations of this algorithm exist, such as [34], by the same authors as the paper on the algorithm [3]. However, we chose to go with Phil Tabor's implementation in [35], since the author included an explanatory YouTube video of his implementation in the documentation, making it easier to understand and adapt his code to suit our needs. This is also a more recent implementation, making it more compatible with up-to-date Python libraries.

We will first explain the specific implementation of the algorithm and the changes we made to it in Section 3-1. Secondly, we will discuss the used environments in Section 3-2.

3-1 Multi-Agent Deep Deterministic Policy Gradient Algorithm

The MADDPG algorithm as given in [3] can be seen in Algorithm 1. We shall now briefly explain the algorithm, and the chosen implementation by Phil Tabor [35]. We also discuss some convergence issues this algorithm is prone to in Subsection 3-1-1.

According to [3], this algorithm uses centralised training with decentralised execution, as can be seen in Figure 3-1[3], and has some distinct properties. First, the agents' learned policies only use local information and observations; in other words, each agent only has access to its own observations at execution time. Next, no assumptions are made about the differentiability of the environment or the structure of communication between the agents. This leads to the possibility of using the algorithm for cooperative, competitive or mixed scenarios. These properties make the MADDPG algorithm suitable for our purposes.

MADDPG is meant as a multi-agent variant of the Deep Deterministic Policy Gradient (DDPG) algorithm from [16]. The principle motivation behind the validity of the multi-agent variant is to include the actions taken by all agents in a centralised action-value function $Q_n^{\pi}(\mathbf{x}, u_1, \ldots, u_N)$,

which removes the problem of non-stationarity, since this will remain the same irrespective of the changing policies.

Both DDPG and MADDPG are meant for deterministic policies only, since they build on the work done in [15]. They make use of target networks and replay buffers to help with convergence. This means that each agent is described by four neural networks, an actor $\mu^{(n)}$, a target actor $\mu'^{(n)}$, a critic $Q^{\mu(n)}$ and a target critic $Q^{\mu'(n)}$. The target networks are updated slowly at each learning step, as in line 15 of Algorithm 1, where $\tau \ll 1$. Another property of DDPG algorithms is that they are suitable for both continuous and discrete state and action spaces.

Each agent keeps an estimate of other agents' policies, which is learned by maximising the log probability of that other agent's actions using an entropy regulariser. Furthermore, to prevent overfitting policies to other agents' policies, the paper discusses the possibility of policy ensembles. Under this approach, each agent trains a collection of sub-policies, one of which is randomly selected to execute at each episode. The chosen implementation by Phil Tabor [35] does not use these ensemble policies, and we will not use them or discuss them further.

According to [3], MADDPG performs better in multi-agent settings than its single-agent counterpart, and the authors believe the algorithm to be generally applicable to all multi-agent systems. The MADDPG algorithm presented in [3] can be seen in algorithm Algorithm 1. In this algorithm, the use of the centralised action-value function can be seen.

In our case, the actor loss to be maximised per agent n is given by: $Q^{\mu(n)}(\mathbf{x}_j, \mu^{(1)}(o_j^{(n)}), \ldots, \mu^{(N)}(o_j^{(N)}))$

Algorithm 1 Multi-Agent Deep Deterministic Policy Gradient for N Agents [3]

1: **for** episode $= 1$ to M **do**
2: Initialize a random process $\mathcal{N}$ for action exploration
3: Receive initial state $\mathbf{x}$
4: **for** $t = 1$ to max-episode-length **do**
5: For each agent n, select action $u^{(n)} = \boldsymbol{\mu}_{\theta^{(n)}}(o^{(n)}) + \mathcal{N}_t$ w.r.t the current policy and exploration
6: Execute actions $u = (u^{(1)}, \ldots, u^{(N)})$ and observe reward r and new state $\mathbf{x}'$
7: Store $(\mathbf{x}, u, r, \mathbf{x}')$ in replay buffer $\mathcal{D}$
8: $\mathbf{x} \leftarrow \mathbf{x}'$
9: **for** agent $n = 1$ to N **do**
10: Sample a random minibatch of S samples $(\mathbf{x}_j, u_j, r_j, \mathbf{x'}_j)$ from $\mathcal{D}$
11: Set $y_j = r_j^{(n)} + \gamma Q^{\boldsymbol{\mu'}(n)}(\mathbf{x'}_j, u'^{(1)}, \ldots, u'^{(N)})|_{u'^{(k)}=\boldsymbol{\mu'}^{(k)}(o_j^{(k)})}$
12: Update critic by minimizing the loss $\mathcal{L}\left(\theta^{(i)}\right) = \frac{1}{S}\sum_j \left(y_j - Q^{\boldsymbol{\mu}(n)}\left(\mathbf{x}_j, u_j^{(1)}, \ldots, u_j^{(N)}\right)\right)^2$
13: Update actor using the sampled policy gradient:
$\nabla_{\theta^{(n)}} J \approx \frac{1}{S}\sum_j \nabla_{\theta^{(n)}} \boldsymbol{\mu}^{(n)}\left(o_j^{(n)}\right) \nabla_{u^{(n)}} Q^{\boldsymbol{\mu}(i)}\left(\mathbf{x}_j, u_j^{(1)}, \ldots, u^{(n)}, \ldots, u_j^{(N)}\right)\Big|_{u^{(n)}=\boldsymbol{\mu}^{(n)}\left(o_j^{(n)}\right)}$
14: **end for**
15: Update target network parameters for each agent i: $\theta'^{(n)} \leftarrow \tau\theta^{(n)} + (1-\tau)\theta'^{(n)}$
16: **end for**
17: **end for**

3-1-1 Convergence Issues

Both the single-agent variant of DDPG and MADDPG suffer from convergence issues. In Phil Tabor's implementation, for example, the algorithm does converge for the showcased environment,

but not for all other readily available environments, which will be discussed in Section 3-2. One paper, [36], discusses these problems in environments with sparse rewards and mentions some solutions. Their key point is that when the reward is sparse, and is found late in training due to that sparsity, the policy can get stuck in a deadlock.

This leads to the obvious solution of making the rewards less sparse and continuously giving the agents penalties and rewards based on distances rather than specific events.

Another solution has to do with exploration noise. The algorithm, as given in Algorithm 1, uses additive noise that disturbs the action given by the policy. [36] suggests using ϵ-greedy exploration instead. This means that a random action will be chosen instead of the policy-determined action with a probability of ϵ. We implemented the ϵ-greedy algorithm where ϵ would slowly decrease throughout training.

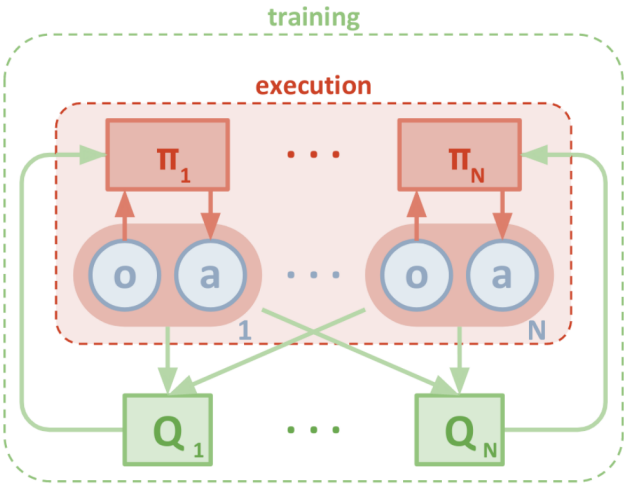


Figure 3-1: Overview of the multi-agent actor-critic architecture of [3]. Here, o is an observation, a is an action, π is a policy, and Q is a Q-function. In MADDPG, π and Q are neural networks. Figure from [3].

3-2 Environment

This section will discuss the environment used for our research. Some general information is given about the workings and properties of the environment, and we will go into more detail on the reward function in Subsection 3-2-1. Finally, we shall discuss the progression of environments as we slowly move towards more realistic simulations in Subsection 3-2-2.

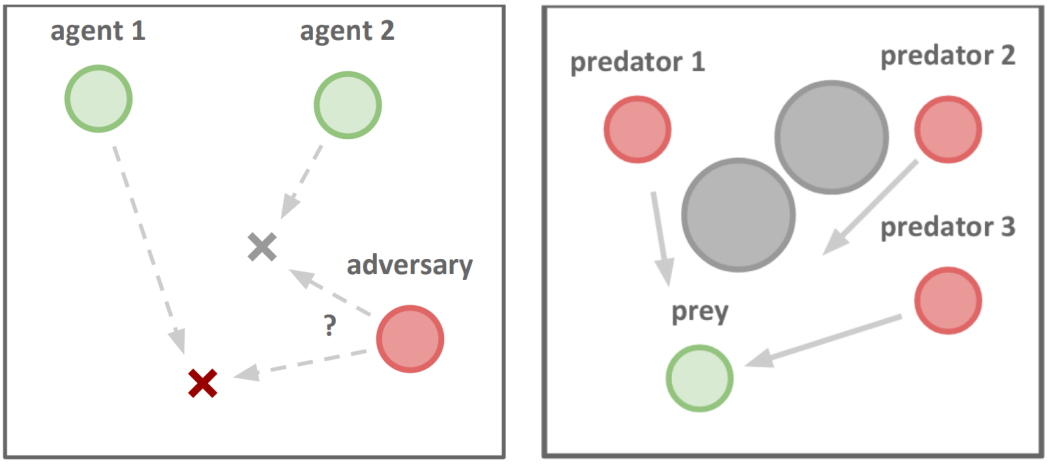
The set of environments used by [3] to evaluate the MADDPG algorithm, is available on GitHub [37]. These environments were eventually moved over to be maintained by Pettingzoo [38]. This set, called Multi-Particle Environments (MPE), consists of 9 different environments. We shall not discuss all of them here, but will refer the interested reader to the documentation on GitHub [37].

In Phil Tabor's implementation, he uses the simple-adversary environment, also known as the physical-deception environment. In the simple-adversary environment, there is one adversary, a certain amount of "good" agents, usually two, and the same number of landmarks. The good

agents are supposed to cover the goal landmark and are rewarded on the distance of the closest agent to this goal. The adversary is also rewarded for being near or covering the goal but does not know which landmark is the goal. The strategy for the agents to learn is to cover all landmarks so the adversary does not know which one is the goal. An exemplary screenshot of the simple-adversary environment can be seen in Figure 3-2a [3]

This environment is not very suitable for our purposes. The agents only need to communicate at the beginning of the episode to divide the landmarks between themselves and will function perfectly without communicating afterwards. Therefore we shall choose a different environment. Since the environment used in [4] is particle-tag, and this work is supposed to be an extension of that paper, choosing particle-tag as the environment is a very natural decision. MPE's equivalent of particle-tag is called simple-tag.

The simple-tag environment is fairly simple. Several adversaries are chasing one good agent. The good agent gets a negative reward when being in contact with an adversary, and the adversaries get a positive reward when being in contact with the good agent. It is also possible to set landmarks as obstacles and to constrain the good agent from leaving the map. An exemplary screenshot of the simple-tag environment can be found in Figure 3-2b [3].



(a) Simple-adversary environment. **(b)** simple-tag environment.

Figure 3-2: Figures of the simple-adversary and simple-tag environments. Figures from [3].

3-2-1 Reward function

The original reward function for the simple-tag environment provided by MPE consisted of the adversaries getting a positive reward if any of them tags the agent, and the agent getting a negative reward if it was tagged. However, these rewards proved too sparse for the algorithm to converge. Sparse rewards are known to cause problems for DDPG algorithms according to [36]. Therefore, we added more shape to the reward function by giving the adversaries a penalty based on their distance from the agent and the agent a positive reward based on its distance from the nearest adversary. Using these rewards, the algorithm would converge. However, it seemed that the agents were not sufficiently encouraged to communicate even with the simple-tag environment and using these rewards. This is evident from Figure 3-3, a plot of the Event-Driven Interactions (EDI) method implemented and tested for the simple-tag environment with their original rewards. It

can be seen that although the communications quickly drop to 0, the adversary return only drops from 146.7 to 140.3, a drop of merely 4.4%.

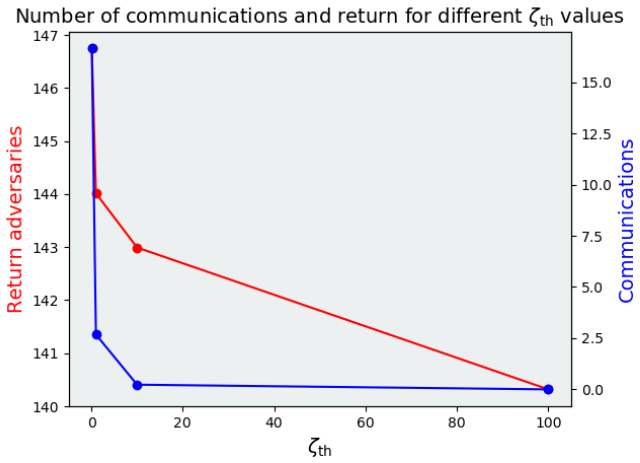


Figure 3-3: Plot of communications and adversary returns for different values of ζ using the simple-tag environment with its original reward.

Therefore, we designed a reward function that is not sparse, but would still require the cooperating agents to communicate. We adapted the reward function for the simple-tag environment to suit these requirements. First, the agent will get a reward based on its distance to the closest adversary, and a penalty if it is tagged by an adversary. The reward for the adversaries is slightly less straightforward. We opted to use only two adversaries for computation speed and ease of implementation. One of the adversaries is supposed to chase the agent, though it does not matter which one; the adversaries will get a penalty for the distance between the adversary closest to the agent and the agent. The other adversary, the one not chasing the agent, is supposed to mirror its partner's movements; if the first adversary's coordinates are $[-1, 0.5]$, the second adversary should be at $[1, -0.5]$. The adversaries will receive a penalty in the form of the norm of the sum of their coordinate vectors. This resulted in the final reward function given in Algorithm 2. This reward function will encourage the adversaries to communicate, since they must closely coordinate their movements to achieve the mirroring. See the recordings on GitHub [29] for an example of a policy trained for this reward.

Furthermore, since the adversaries are cooperative agents, all their rewards are summed together, which is a shared reward.

Algorithm 2 Reward Function

Inputs:
Set of agents $\mathcal{A}$, of size $N_a = 1$
Set of adversaries $\mathcal{C}$, of size N_c
Total set of robots $\mathcal{N}$, of size $N = N_a + N_c$
Agent coordinates $(x^{(a)}, y^{(a)})$ for $a \in \mathcal{A}$
Adversary coordinates $(x^{(c)}, y^{(c)})$ for $c \in \mathcal{C}$
Diameter of robots d

for $n \in \mathcal{N}$ **do**
  Initialize:
  $r = 0$
  if $n \in \mathcal{A}$ **then**
    for $c \in \mathcal{C}$ **do**
      $\Delta_x^{(c)} = ||(x^{(a)}, y^{(a)}) - (x^{(c)}, y^{(c)})||$
      if agent is tagged by the adversary, i.e. $\Delta_x^{(c)} \leq d$ **then**
        $r \ -= \ 10$
      end if
    end for
    $r \ += 0.2 \cdot \min_c \Delta_x^{(c)}$

  else if $n \in \mathcal{C}$ **then**
    for $c \in \mathcal{C}$ **do**
      $\Delta_x^{(c)} = ||(x^{(a)}, y^{(a)}) - (x^{(c)}, y^{(c)})||$
      if agent is tagged by the adversary, i.e. $\Delta_x^{(c)} \leq d$ **then**
        $r \ += \ 5$
      end if
      if $n \neq c$ **then**
        $r \ -= ||(x^{(n)}, y^{(n)}) + (x^{(c)}, y^{(c)})||$
      end if
    end for
    $r \ -= \min_c \Delta_x^{(c)}$
  end if
end for

3-2-2 Environment Progression

For our experiments, we used three different environments, all versions of the one described above but with different levels of reality. All environments are set to be $2\text{m} \times 2\text{m}$, with $x, y \in [-1, 1]\text{m}$. Eventually, we also want to perform tests on real robots, but for this research, this was not feasible due to time constraints and the availability of the laboratory. This environmental progression is desirable, because the more realistic a simulation becomes, the more it will increase computation time. So if we can do the bulk of the training in a highly simplified environment, and then only have to do a few rounds of training in the more realistic environments, this will greatly reduce the overall training time.

On GitHub [29] are recordings of renderings of executions of all the environments.

Environment A This environment is described above and uses the kinematics and dynamics as defined by the MPE environment, a screenshot of which can be seen in Figure 3-4. The action

space per agent is pretty simple; it is a vector of length 5. In the case of a discrete action and state space, this vector will be all zeros except for one entry, which will be 1. This will indicate whether the agent moves up, left, right, down or stays where it is. However, we are working with a continuous action and state space. This means that all entries in the vector will be $\in [0, 1]$. The environment translates this into a 2D force vector F. This force is then integrated into an updated position via these dynamics:

$$a = \frac{F}{m} \tag{3-1}$$

$$v_{k+1} = v_k \cdot (1 - d) + a \cdot \Delta_t \tag{3-2}$$

$$x_{k+1} = x_k + v_{k+1} \cdot \Delta_t \tag{3-3}$$

Where m is the robot mass, a the acceleration v the speed, $d = 0.25$ a damping factor, $\Delta_t = 0.25s$ the time-step and x the position. These equations are slightly simplified; the environment does a few additional things, such as adhering to a maximum speed and adding some small noise.

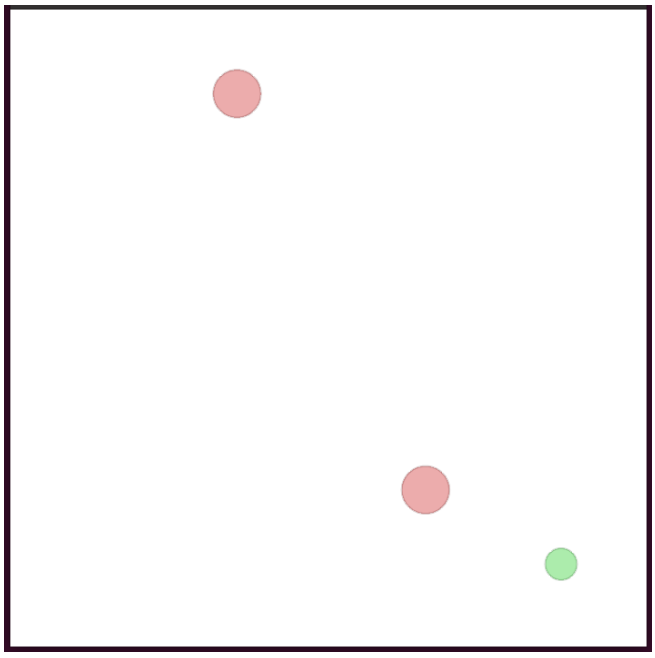


Figure 3-4: Screenshot of the MPE simple-tag environment, with two red adversaries, one chasing the green agent, the other mirroring its partner's movements.

Environment B This environment uses the reward function given above and generates waypoints through the dynamics of the MPE environment. However, this environment uses the kinematics of a *differential drive robot*, such as the Elisa-3 [39] or E-Puck [40] robot by GCtronic. A diagram of some relevant features, defined coordinate systems and variables can be found in Figure 3-5. A discrete-time state-space representation of the used robot kinematics is found in Eq. 3-5. In these equations, v is the longitudinal velocity of the robot. There is no lateral velocity, since the robot is a non-holonomic system. $\dot{\phi}$ is the yaw rate of the robot.

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \phi \end{bmatrix} \tag{3-4}$$

$$\mathbf{x}_{k+1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{x}_k + \begin{bmatrix} \cos\phi \cdot \frac{\Delta_t}{h} & 0 \\ \sin\phi \cdot \frac{\Delta_t}{h} & 0 \\ 0 & \frac{\Delta_t}{h} \end{bmatrix} \begin{bmatrix} v \\ \dot{\phi} \end{bmatrix} \tag{3-5}$$

The agents are then steered to their waypoint by a low-level controller, which can be a simple Proportional-Integral-Differential (PID) or Model Predictive Control (MPC) controller. Since designing the low-level controller would be outside the scope of this research, both the differential drive kinematics and the controller were taken off of GitHub [41]. Unfortunately, the controllers given there did not seem sufficient for our purposes, though we did use the provided robot model as a base for our own. We tried implementing a basic controller that would first turn the robot in place until it faced its waypoint, then move forward at full speed. This was working surprisingly well, so we decided to stick to this, though, for future research, it might be nice to implement a proper MPC controller. The simple controller can be found in Algorithm 3; please also consult Figure 3-5 for variable definitions.

For the use of this controller, we used a control horizon of $h = 4$, meaning that for every time-step of the waypoint generation by the MPE environment (with $\Delta_t = 0.25\text{s}$), the controller input and state of the robot is updated four times.

Environment C This environment uses the same strategy of generating waypoints, and the same low-level controller. Only the robot model and update are now provided by the simulator Webots, instead of our own robot model. In order for this to work, we needed to translate the longitudinal speed v and yaw-rate $\dot{\phi}$ to the motor speed of the right and left wheel, ω_r and ω_l respectively, for which we used Eq. 3-6. In this equation, r is the wheel radius, and b is the distance from the wheel to the midline of the robot; these variables can also be seen in Figure 3-5. For the Elisa-3 [39] robot, these parameters are $r = 0.0042\ \text{m}$ and $b = 0.02\ \text{m}$. These motor speeds we can set in Webots. Screenshots of the Webots simulation can be seen in Figure 3-6. This environment also uses a control horizon of $h = 4$.

$$\begin{bmatrix} \omega_r \\ \omega_l \end{bmatrix} = \begin{bmatrix} \frac{1}{r} & \frac{b}{r} \\ \frac{1}{r} & -\frac{b}{r} \end{bmatrix} \begin{bmatrix} v \\ \dot{\phi} \end{bmatrix} \tag{3-6}$$

Algorithm 3 Simple Controller

Inputs:
Set of robots $\mathcal{N}$, of size N
Robot coordinates $(x^{(n)}, y^{(n)})$ for $n \in \mathcal{N}$
Set of goal waypoints $(\bar{x}^{(n)}, \bar{y}^{(n)})$ for $n \in \mathcal{N}$
for $n \in \mathcal{N}$ **do**
  $\bar{\phi} = \arctan \frac{\bar{y}^{(n)} - y^{(n)}}{\bar{x}^{(n)} - x^{(n)}}$ (and choose correct quadrant)
  $\Delta_\phi = \bar{\phi} - \phi$ (and wrap to $[-\pi,\ \pi]$)
  $d = ||(\bar{x}^{(n)}, \bar{y}^{(n)}) - (x^{(n)}, y^{(n)})||$
  if $d < 0.005$ **then**
    $v = 0,\ \dot{\phi} = 0$
  else if $0.005 \leq d < 0.05$ **then**
    if $|\Delta_\phi| < 0.01$ **then**
      $v = 0.2,\ \dot{\phi} = 0$
    else
      $v = 0,\ \dot{\phi} = \text{sign}(\Delta_\phi) \cdot 0.5$
    end if
  else if $|\Delta_\phi| < 0.001$ **then**
    $v = 0.585,\ \dot{\phi} = 0$
  else if $0.001 \leq |\Delta_\phi| < \frac{1}{4}\pi$ **then**
    $v = 0.45,\ \dot{\phi} = \text{sign}(\Delta_\phi) \cdot 2$
  else if $\frac{1}{4}\pi \leq |\Delta_\phi| < \frac{1}{2}\pi$ **then**
    $v = 0.3,\ \dot{\phi} = \text{sign}(\Delta_\phi) \cdot 3$
  else
    $v = 0,\ \dot{\phi} = \text{sign}(\Delta_\phi) \cdot 4$
  end if
end for
Outputs:
$(v^{(n)}, \dot{\phi}^{(n)})$ for $n \in \mathcal{N}$

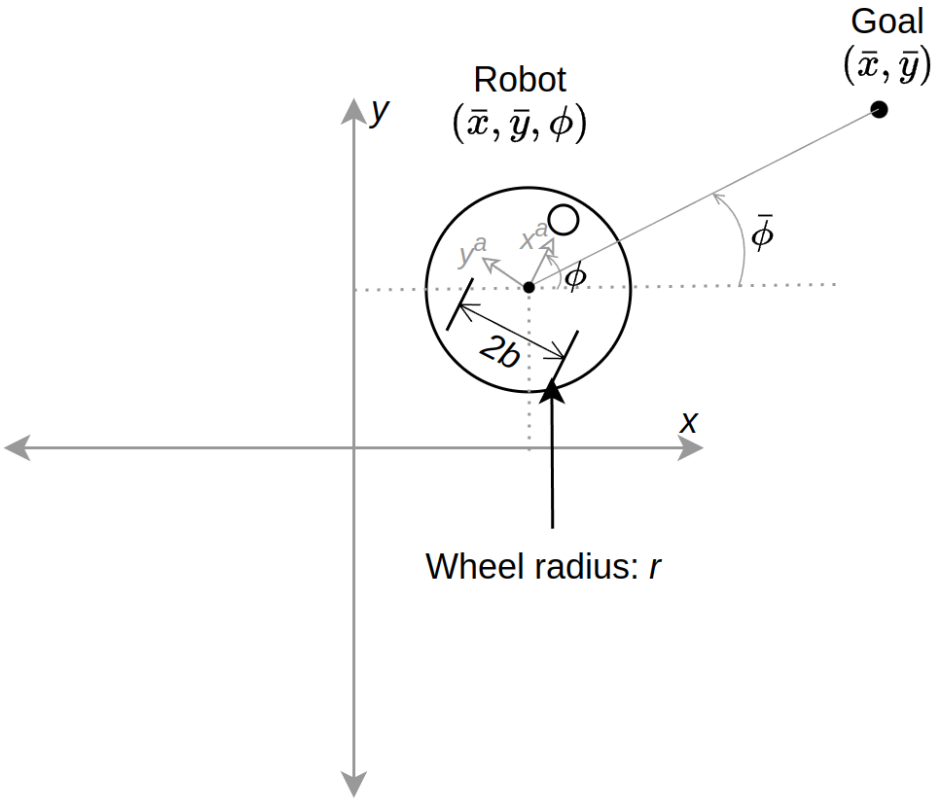


Figure 3-5: Graphic top-view representation of a robot and some important variables, such as the robot coordinates and goal coordinates, the definition of the coordinate system and the relevant robot parameters.

Figure 3-6: Screenshot of the Webots environment, with two red adversaries, one chasing the green agent, the other mirroring its partner's movements.

Chapter 4

Extension of the Robustness Surrogates to the MADDPG algorithm

This chapter will propose an extension and slight generalisation of the work done in [4], which is already an extension of [22]. These papers propose an equivalent to Lyapunov functions in classical Event-Triggered Control (ETC), to make it applicable for communications in multi-agent systems, which we call Event-Driven Interactions (EDI).

First, we shall give and explain the original algorithm in Section 4-1, after which we shall discuss our adaptation in Section 4-2, which we shall verify in Section 4-3.

4-1 Original Algorithm and Shortcomings

The original algorithm from [4] proposes the use of *robustness surrogates* as an equivalent to Lyapunov functions in ETC for communicating in Multi-Agent Reinforcement Learning (MARL) systems. These robustness surrogates, denoted by Γ_ζ, with ζ a sensitivity parameter, are defined as in Eq. 4-1 from [4]. In simple terms, $\Gamma_\zeta(\mathbf{x})$ defines a set of states in which executing the same input as the optimal input in $\mathbf{x}$ does not cause a degradation in Q-value more than ζ. The algorithm calculating these robustness surrogates can be seen in Algorithm 4, and describes how the robustness surrogates are found by exploring cubes around the state $\mathbf{x}$ and looking at how far you can go before the Q-values decrease more than ζ. Note the notation used in Algorithm 4 and other algorithms in this chapter of $M(\mathbf{x}) = [\mu^{(1)}(x^{(1)}), \cdots, \mu^{(N)}(x^{(N)})]$. A visualisation of the use of these robustness surrogates can be seen in Figure 4-1 from [4].

$$\begin{aligned}\Gamma_\zeta(\mathbf{x}) := \max \{d \mid \forall \mathbf{x}' : \|\mathbf{x}' - \mathbf{x}\|_\infty \leq d \Rightarrow \\ \Rightarrow Q^*(\mathbf{x}', \Pi^*(\mathbf{x})) \geq V^*(\mathbf{x}') - \zeta\}\end{aligned} \tag{4-1}$$

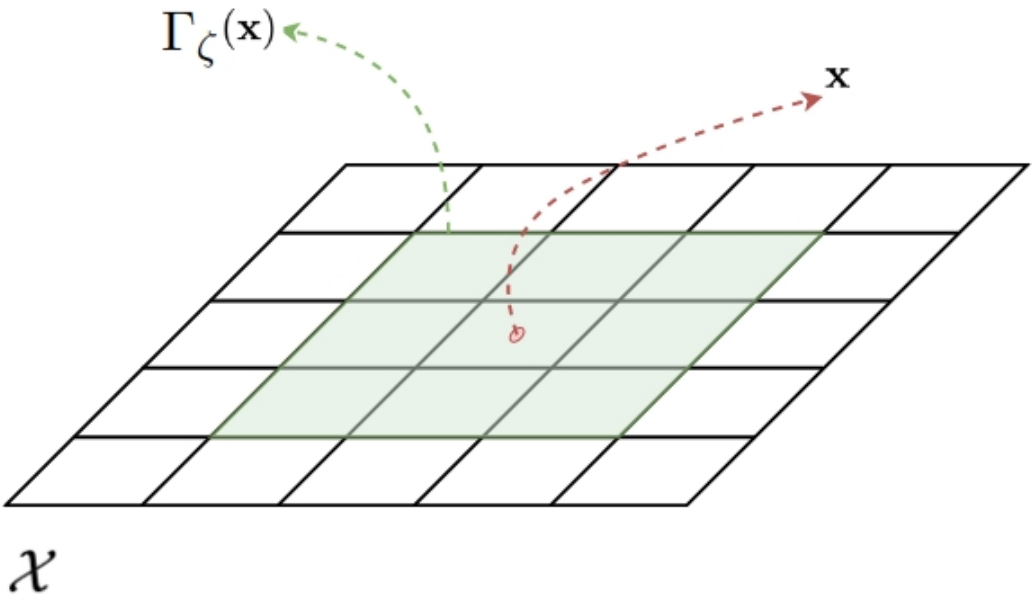


Figure 4-1: Visualisation of the use of the robustness surrogates, from [4].

Algorithm 4 Computation of Robustness Indicator [4]

Initialize:
  $\mathbf{x}$
  $d = 1$
  $\text{done} = False$
while not done **do**
  Compute set $\mathcal{X}^d := \{\mathbf{x}' : \|\mathbf{x} - \mathbf{x}'\| = d\}$
  if $\exists \mathbf{x}' \in \mathcal{X}^d : Q^*(\mathbf{x}', M^*(\mathbf{x})) \leq V^*(\mathbf{x}') - \zeta$ **then**
    $\text{done} = True$
  else
    $d++$
  end if
end while
$\Gamma_\zeta(\mathbf{x}) = d - 1$

Now, according to [4], if a collaborative multi-agent Markov Decision Process (MDP) follows the algorithm given in Algorithm 5 [4], using the robustness surrogate from Eq. 4-1, the expected return will be bounded by Eq. 4-2, the proof of which can be seen in [4]. Please note that in [4], a small error was made: they define the limit of $\sum_{k=0}^{\infty} \gamma^k = \frac{\gamma}{1-\gamma}$, when it is actually $\sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma}$, we have maintained the correct limit in this report and our calculations. Eq. 4-2 indicates that the loss of performance by using the robustness surrogates as a communication trigger will be at the most $\zeta \frac{1}{1-\gamma}$.

$$\mathbb{E}_{\mathbf{x}_0}\left[\sum_{t=0}^{\infty} \gamma^t \mathbf{R}\left(\mathbf{x}_t, \Pi^*\left(\hat{\mathbf{x}}_t\right), \mathbf{x}_{t+1}\right)\right] \geq V^*(\mathbf{x}_0) - \zeta \frac{1}{1-\gamma} \tag{4-2}$$

Since the robustness surrogates Γ_ζ can be quite difficult to compute analytically, [4] estimates the

Algorithm 5 Self-triggered State Sharing [4]

Initialize:
N agents at $\mathbf{x_0}$
Last-known state vector $\hat{\mathbf{x}}_0 = \mathbf{x}_0,\ n \in N$
$t = 0$
while $t < t_{\max}$ **do**
  for $n \in N$ **do**
    if $\left\| \mathbf{x}_t^{(n)} - \hat{\mathbf{x}}_{t-1}^{(n)} \right\|_\infty > \Gamma_\zeta \left(\hat{\mathbf{x}}_{t-1} \right)$ **then**
      $\hat{\mathbf{x}}_t^{(n)} \leftarrow \mathbf{x}_t^{(n)}$
      Send updated $\hat{\mathbf{x}}_t^{(n)}$ to all N_{-n}
    end if
    Execute action $\hat{u}^{*(n)} = \pi^{*(n)}(\hat{\mathbf{x}})$
  end for
  $t++$
end while

robustness surrogate, using a support vector regression model in combination with the scenario approach [42] to make estimates of the robustness surrogates for each state. Using these robustness surrogates is relatively straightforward, and can be seen in Algorithm 5. For every agent, it compares its true state with its last communicated state, and if the infinity norm of their difference is larger than the robustness surrogate, it will send an update.

The resulting algorithm has several shortcomings and is not directly applicable to the Multi-Agent Deep Deterministic Policy Gradient (MADDPG) algorithm. The main problem is that this algorithm works exclusively for discrete state spaces, while MADDPG uses a continuous state space. This problem is not present in Algorithm 5, but does make Algorithm 4 inapplicable to our situation. This is due to the $d++$ construction, where the algorithm looks in an expanding cube around the state where the Q-function starts to deviate too much. This brings us to one of the other shortcomings of this algorithm, namely that it looks in a square-shaped area around the state and if there is one direction in which the difference in Q-function crosses the threshold, this will limit the robustness indicator Γ_ζ. However, It is possible that for different directions, the robustness indicator has different values; this algorithm does not accommodate that. Another shortcoming is that this algorithm only applies to algorithms that can compute the Q-values explicitly, which means that the entire family of pure policy-based (see Figure 2-2) algorithms can not use it.

4-2 Algorithm Adaptations

We mean to mitigate some of the shortcomings of the original algorithm given in [4], which is explained in the previous section, but not all. It is necessary to adapt the algorithm to continuous state spaces for our purposes. Furthermore, we would like to address the issue of the square-shaped area of exploration. In order to do this, we adapted Algorithm 4 to Algorithm 6 and Algorithm 5 to Algorithm 7.

Algorithm 6 takes a possible sequence of state transitions as input, which can be produced by running episodes with a pre-trained actor and critic. For every combination of states in that sequence, it calculates ζ, the difference in Q-values when making decisions using the outdated

Algorithm 6 New Computation of Robustness Indicator

Inputs:
known sequence of transitions $(\mathbf{x}^0, \mathbf{x}^1, \ldots, \mathbf{x}^I)$
number of agents N
number of cooperating agents N_c
Initialize:
$\text{dataset} = \varnothing$

for $i = 0$ to $I - 1$ **do**
  for $j = i + 1$ to I **do**
    for n in N_c **do**
      $\zeta^{(n)} = Q'^{(n)}(\mathbf{x}_j, M^{*\prime}(\mathbf{x}_j)) - Q'^{(n)}(\mathbf{x}_j, M^{*\prime}(\mathbf{x}_i))$
    end for
    $\zeta = \text{sum}\, \zeta^{(n)}$
    for n in N_c **do**
      add the IO entry $([x_i^{(n)},\ x_j^{(n)}],\ \zeta)$ to dataset
    end for
  end for
end for

Outputs:
dataset

Algorithm 7 New Self-triggered State Sharing

Inputs:
Sensitivity threshold ζ_{th}
Trained neural network NN
Initialize:
N agents at $\mathbf{x_0}$
Last-known state vector $\hat{\mathbf{x}}_0 = \mathbf{x}_0,\ n \in N$
$t = 0$
while $t < t_{\max}$ **do**
  for $n \in N$ **do**
    $\zeta = \text{NN}(\mathbf{x}_t^{(n)}, \hat{\mathbf{x}}_{t-1}^{(n)})$
    if $\zeta > \zeta_{\text{th}}$ **then**
      $\hat{\mathbf{x}}_t^{(n)} \leftarrow \mathbf{x}_t^{(n)}$
      Send updated $\hat{\mathbf{x}}_t^{(n)}$ to all N_{-n}
    end if
    Execute action $\hat{u}^{*(n)} = \pi^{*(n)}(\hat{\mathbf{x}})$
  end for
  $t++$
end while

state. This is done for all cooperating agents, and the final ζ is the sum of these differences since we are interested in the total loss of the cooperating agents. We forgo the extra step of defining robustness surrogate Γ_ζ based on vector norms and use ζ directly. ζ can now be interpreted as the smallest ζ_{th} for which the two states are still in each other's $\Gamma_{\zeta_{\text{th}}}$, and thus, if $\zeta_{\text{th}} < \zeta$, the states are not in each other's $\Gamma_{\zeta_{\text{th}}}$ sets. The output of this algorithm is n (one for each cooperating agent) entries into a dataset, which consists of an input vector of the updated and outdated observations of that agent, and the output ζ.

This produced dataset can then be used to train an additional neural network, which takes as input two agent observations and outputs ζ, the loss in value when not communicating. During execution, we will now choose to communicate based on a determined ζ threshold ζ_{th}. And the cooperating agents can sample their updated and outdated states in the neural network, which will return ζ; that agent will then communicate if $\zeta > \zeta_{\text{th}}$. In between communications, the agents will operate on a sample-and-hold strategy. During execution, the agents do not have access to the Q-functions or each other's policies, which is the reason for not being able to use the Q-values during execution directly. We want to use both the updated and outdated states as inputs, to be able to differentiate ζ in the direction of the agent's movements. A flow diagram of the communication between two agents can be seen in Figure 4-2, as an extension of Figure 2-4. A more detailed diagram of the decision process can be seen for agent 1 in Figure 4-3.

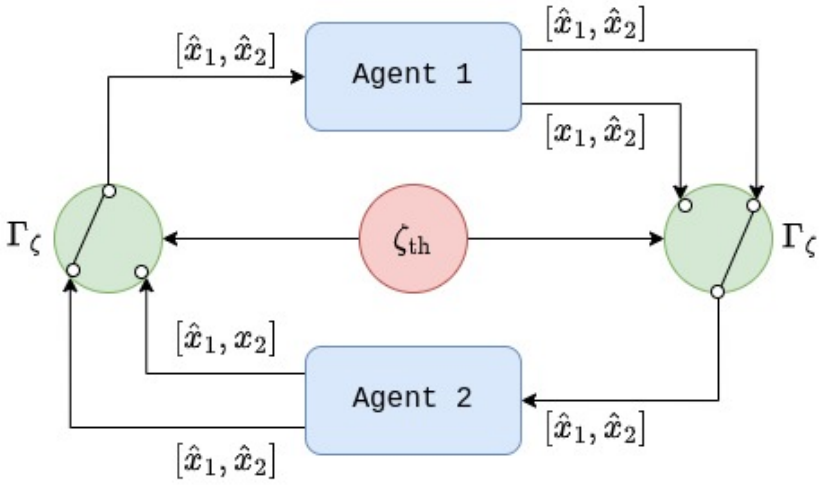


Figure 4-2: Diagram of state communication between two agents using robustness surrogates.

Theorem 3. *If we assume the neural network to be sufficiently trained, we can ensure the loss of return per time-step to be smaller than a chosen* ζ_{th}*, and thus the total loss* $\leq \zeta_{\text{th}} \frac{1}{1-\gamma}$.

Proof. If we view the neural network as a lookup table, it will output for two states the loss in Q-function for making decisions in one of the states with the information of the other state. This is the smallest ζ for which the states are still in each other's Γ_ζ set of states. Therefore, if we pick a ζ_{th} which is smaller than that ζ, this means the two states are not in each other's $\Gamma_{\zeta_{\text{th}}}$. By construction, Algorithm 7 ensures that $\hat{\mathbf{x}} \in \Gamma_{\zeta_{\text{th}}}(\mathbf{x})$. Which, together with the expression for the optimal Q-values, means:

$$Q^*\left(\mathbf{x}_t, M^*\left(\hat{\mathbf{x}}_t\right)\right) = \mathbb{E}_{\mathbf{x}_t}\left[\hat{r}_t + \gamma V^*\left(\mathbf{x}_{t+1}\right)\right] \geq V^*\left(\mathbf{x}_t\right) - \zeta_{\text{th}} \tag{4-3}$$

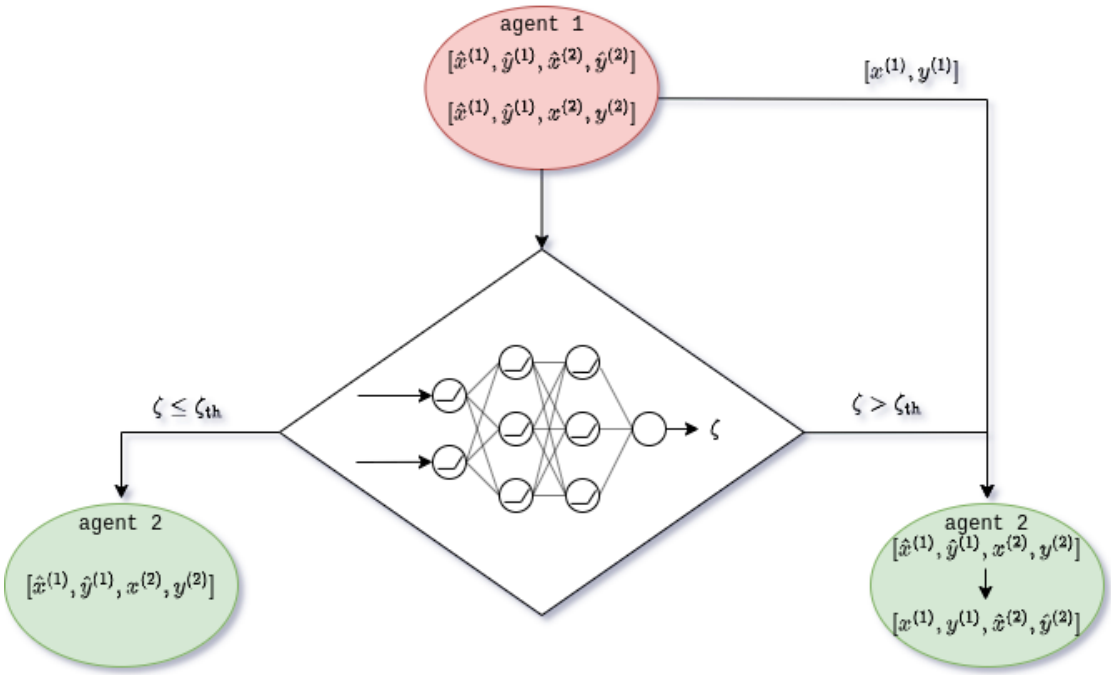


Figure 4-3: Decision process for agent 1 whether to communicate its updated state to agent 2. If $\zeta > \zeta_{\text{th}}$, then agent 2's knowledge of agent 1's state is updated.

We can now follow the same proof as given in [4], which we shall repeat here:

> "*Now let* $\hat{V}(\mathbf{x}_0) := \mathbb{E}_{\mathbf{x}_0}\left[\sum_{t=0}^{\infty} \gamma^t \hat{r}_t\right]$ *be the value of the policy obtained from executing the actions* $M^*(\hat{\mathbf{x}}_t)$*. Then:*

$$\begin{aligned}
&\mathbb{E}_{\mathbf{x}_0}\left[\sum_{t=0}^{\infty} \gamma^t \hat{r}_t\right] = \mathbb{E}_{\mathbf{x}_0}\left[\hat{r}_0 + \gamma V^{\hat{M}}(\mathbf{x}_1)\right] = \\
&= \mathbb{E}_{\mathbf{x}_0}\left[\hat{r}_0 + \gamma \hat{V}(\mathbf{x}_1) + \gamma V^*(\mathbf{x}_1) - \gamma V^*(\mathbf{x}_1)\right] = \\
&= \mathbb{E}_{\mathbf{x}_0}\left[\hat{r}_0 + \gamma V^*(\mathbf{x}_1)\right] + \gamma \mathbb{E}_{\mathbf{x}_0}\left[\hat{V}(\mathbf{x}_1) - V^*(\mathbf{x}_1)\right]
\end{aligned} \tag{4-4}$$

Then, substituting Eq. 4-3 in Eq. 4-4:

$$\mathbb{E}_{\mathbf{x}_0}\left[\sum_{t=0}^{\infty} \gamma^t \hat{r}_t\right] \geq V^*(\mathbf{x}_0) - \zeta_{\text{th}} + \gamma \mathbb{E}_{\mathbf{x}_0}\left[\hat{V}(\mathbf{x}_1) - V^*(\mathbf{x}_1)\right] \tag{4-5}$$

Now, observe we can apply the same principle as in Eq. 4-4 for the last term in Eq. 4-5,

$$\begin{aligned}
&\hat{V}(\mathbf{x}_1) - V^*(\mathbf{x}_1) = \mathbb{E}_{\mathbf{x}_1}\left[\hat{r}_1 + \gamma\hat{V}(\mathbf{x}_2)\right] - V^*(\mathbf{x}_1) = \\
&= Q^*(\mathbf{x}_1, M^*(\hat{\mathbf{x}}_1)) + \gamma\mathbb{E}_{\mathbf{x}_1}\left[\hat{V}(\mathbf{x}_2) - V^*(\mathbf{x}_2)\right] - V^*(\mathbf{x}_1) \geq \\
&\geq V^*(\mathbf{x}_1) - \zeta_{\text{th}} - V^*(\mathbf{x}_1) + \gamma\mathbb{E}_{\mathbf{x}_1}\left[\hat{V}(\mathbf{x}_2) - V^*(\mathbf{x}_2)\right] = \\
&= -\zeta_{\text{th}} + \gamma\mathbb{E}_{\mathbf{x}_1}\left[\hat{V}(\mathbf{x}_2) - V^*(\mathbf{x}_2)\right]
\end{aligned} \tag{4-6}$$

Substituting Eq. 4-6 in Eq. 4-5:

$$\mathbb{E}_{\mathbf{x}_0}\left[\sum_{t=0}^{\infty}\gamma^t\hat{r}_t\right] \geq V^*(\mathbf{x}_0) - \zeta_{\text{th}} - \gamma\zeta_{\text{th}} + \gamma^2\mathbb{E}_{\mathbf{x}_0}\left[\mathbb{E}_{\mathbf{x}_1}\left[\hat{V}(\mathbf{x}_2) - V^*(\mathbf{x}_2)\right]\right] \tag{4-7}$$

Now it is clear that, applying Eq. 4-6 recursively:

$$\begin{aligned}
&\mathbb{E}_{\mathbf{x}_0}\left[\sum_{t=0}^{\infty}\gamma^t\hat{r}_t\right] \geq V^*(\mathbf{x}_0) - \zeta_{\text{th}} \\
&\quad - \gamma\zeta_{\text{th}} + \gamma^2\mathbb{E}_{\mathbf{x}_0}\left[\mathbb{E}_{\mathbf{x}_1}\left[\hat{V}(\mathbf{x}_2) - V^*(\mathbf{x}_2)\right]\right] \geq V^*(\mathbf{x}_0) - \zeta_{\text{th}}\sum_{k=0}^{\infty}\gamma^k
\end{aligned} \tag{4-8}$$

Substituting $\sum_{k=0}^{\infty}\gamma^k = \frac{1}{1-\gamma}$ in Eq. 4-8:"

$$\mathbb{E}_{\mathbf{x}_0}\left[\sum_{t=0}^{\infty}\gamma^t\hat{r}_t\right] \geq V^*(\mathbf{x}_0) - \zeta_{\text{th}}\frac{1}{1-\gamma} \tag{4-9}$$

Now we can conclude that the total loss of return for an episode by using our EDI method is bounded by $\zeta_{\text{th}}\frac{1}{1-\gamma}$. □

4-3 Experimental Validation of Implementation

Three tests were performed to check if the algorithm was implemented correctly and doing its job. The conclusion from these experiments can be found in Subsection 4-3-1.

Looking through sequence The first test is to take a sequence of observations, and see if the ζ output by the neural network increases for observations further apart. The communication protocol can also be tested this way by picking a ζ_{th} and checking if the agent is triggered to communicate from some point in the sequence.

This test was performed by running 1000 episodes. From every episode, a random state from the first 20 states of the sequence is selected, and a random cooperating agent is selected. Then, for that state and that agent, the neural network's output is checked for all subsequent states. The given ζ was recorded, and we graphed the average ζ for states versus the number of steps apart.

Hypothesis 4. *For a correct implementation, on average, ζ will increase for states further apart.*

The results of this test can be seen in Figure 4-4. This graph shows that ζ indeed seems to be increasing, indicating Hypothesis 4 is correct. However, ζ does not increase monotonically over the entire graph; even if we smooth the graph, this is not the case. A reasonable assumption is that the fluctuation is caused by taking a too-small sample size. Episodes can be very different due to the random starting locations of the agents. It is interesting to note that ζ seems to stay more or less constant between 10 and 40 episode steps; after that, it shoots up.

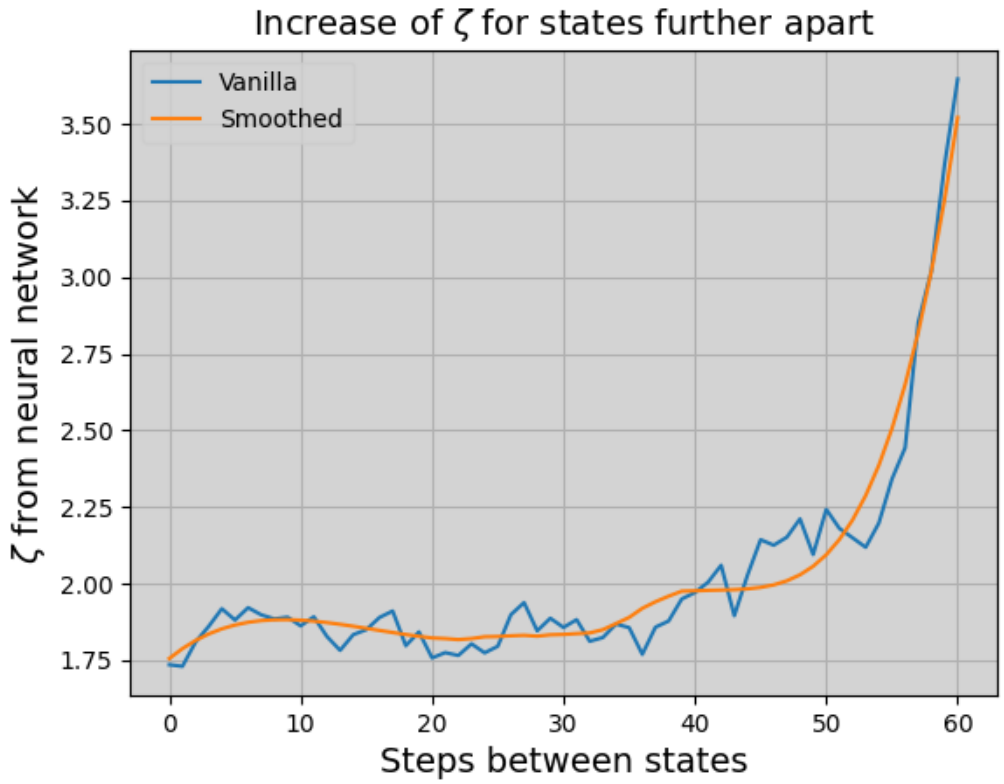


Figure 4-4: Visualisation of the increase of ζ for a larger difference in states.

Check limit The second test is to run episodes without reduced communications and with reduced communications for various values of ζ_{th} and check whether the loss in performance is indeed $\leq \zeta_{\text{th}} \frac{\gamma}{1-\gamma}$. For every ζ_{th} we run 1000 episodes.

Hypothesis 5. *For a correct implementation, the loss of adversary return should stay within the theoretical limit.*

The results of this test can be seen in Table 4-1. It seems that all losses of return are within the bound given by $\zeta_{\text{th}} \frac{1}{1-\gamma}$, as desired. It is also worth mentioning that the losses are very close to monotonically increasing with ζ. When this is not so, the decrease is small enough to be negligible.

ζ_{th}	$\zeta_{\text{th}} \frac{1}{1-\gamma}$	Loss of return
0.1	10.0	7.9
0.2	20.0	9.6
0.3	30.0	19.3
0.4	40.0	18.9
0.6	60.0	24.3
0.8	80.0	29.0
1.0	100.0	33.2
1.5	150.0	40.3
2.0	200.0	45.8
2.5	250.0	51.7
3.0	300.0	52.3
3.1	310.0	54.2
3.2	320.0	53.2
3.3	330.0	53.0
3.4	340.0	55.1
3.5	350.0	54.2
4.0	400.0	55.1

Table 4-1: Table to indicate the limit in the loss of return, and the actual loss of return for different values of ζ.

Performance and Communications The third test is to run episodes with reduced communication for multiple values of ζ_{th} and record the performance and number of communications. We then compared this with the baseline of performance for periodic communication. Both the baseline and every value of ζ_{th} we test for 1000 episodes.

Hypothesis 6. *For a correct implementation, for increasing ζ, both the performance and number of communications should go down.*

We can see the results of this test in Figure 4-5. We can see easily that both the adversary return and the number of communications per episode decrease for increasing ζ, as predicted in Hypothesis 6.

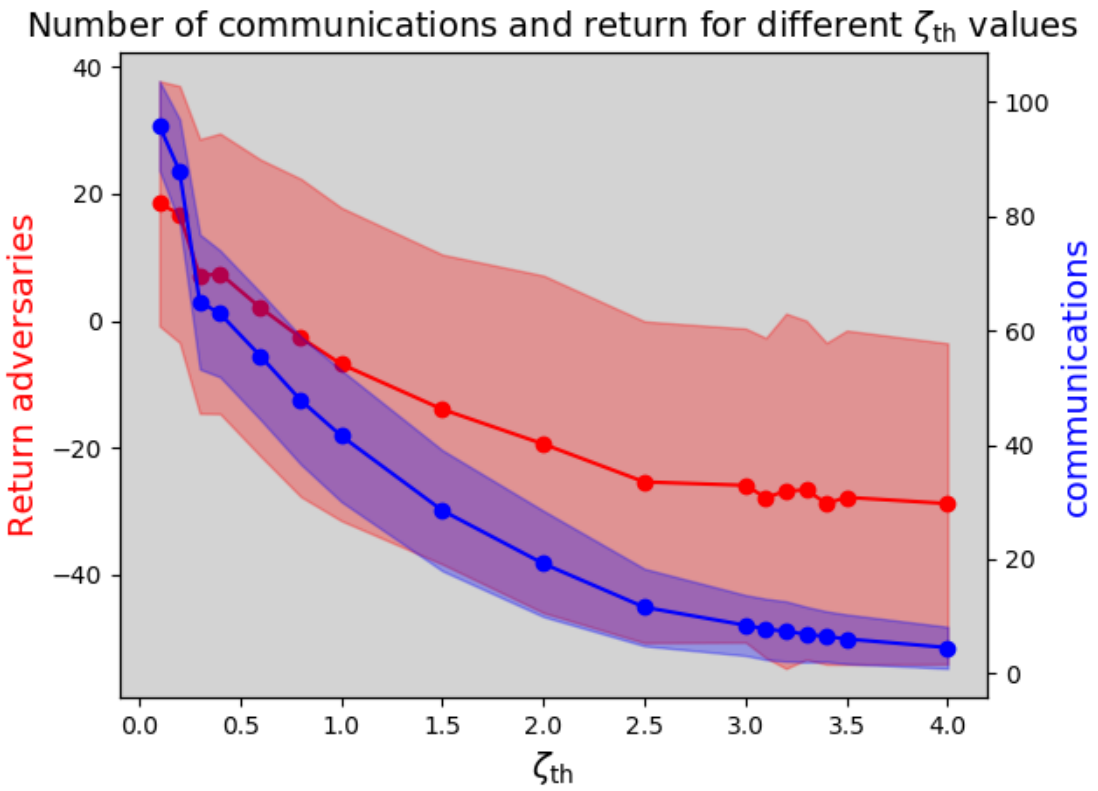


Figure 4-5: Graph displaying the change in adversary return and number of communications per episode for varying ζ.

4-3-1 Validation Conclusion

From the three tests performed above, we can conclude that the EDI method is correctly implemented. The only point of doubt is that Figure 4-4 is not monotonically increasing.

Chapter 5

Lexicographic Optimisation

This chapter will discuss our implementation of the observational robustness algorithm from [5]. Although the original algorithm, which we will discuss in Section 5-1, did not need many changes to be applicable to our situation, still some design choices required to be made, on which we shall elaborate in Section 5-2. Finally, in Section 5-3, we will validate the implementation by testing its performance over 1000 episodes.

5-1 Original Algorithm

The idea behind Lexicographic Reinforcement Learning (LRL) from [27] is to optimise the policy for multiple hierarchically ordered objectives. The idea is to find the set of policies that are at most some tolerance away from the optimal set of policies optimised to the first objective. Then, within that set of policies, it will find the set of policies that optimise the second objective within some bound of its optimal, and so forth for the following objectives.

This LRL method is applied by [5] to obtain sufficient observational robustness in an Observationally Disturbed Markov Decision Process (DOMDP) setting, which includes a noise kernel T, making $T(y|x)$ the probability of measuring y while the true state is x. They call this *Lexicographically Robust Reinforcement Learning (LRRL)*

They formalise observational robustness through the use of *robustness regret* $\rho(\pi, T)$, which is defined as the difference between the expected value of an initial state in a Markov Decision Process (MDP) using policy π with and without the noise kernel. We can now say that a policy π is κ robust against T if $\rho(\pi, T) \leq \kappa$. A policy is maximally robust if $\kappa = 0$.

One of the more significant results of [5] is the so-called *inclusion theorem*. This theorem states that the set of maximally robust policies Π_0, fully contains the set of policies that have a noise disadvantage, $D^\pi(x, T) = V^\pi(x) - \mathbb{E}[Q^\pi(x, u)]$, of zero. This set is denoted by Π_D. In turn, this set fully contains the set of fixed point policies Π_T, which action distribution is unaffected by T. Π_T itself fully contains the set of constant policies $\bar{\Pi}$, which are policies that have the same action distribution for any state:

$$\bar{\Pi} \subseteq \Pi_T \subseteq \Pi_D \subseteq \Pi_0 \tag{5-1}$$

Intuitively, this inclusion relation makes sense. Constant policies are unaffected by the noise kernel, which is why they are included in Π_T. It also makes sense that Π_T is maximally robust, since the noise kernel will not impact the expected value.

There is a trade-off between robustness and optimality. A constant policy will certainly be robust but far from optimal. Likewise, the optimal policy will most likely not be maximally robust. This is where the lexicographic optimisation comes in. It ensures a policy that is maximally some tolerance ε away from the optimum, while optimising robustness within that tolerance. The natural approach would first be to run the learning algorithm that optimises for performance, then run the learning algorithm that optimises for robustness while constraining it to be less than ε away from the first optimum found in the first pass. Unfortunately, this is not a very efficient way to do this, since the complete learning algorithm needs to be run through twice, which is why both [27] and [5] employ *Lagrangian relaxation* techniques [28]. A short explanation follows of how [5] has implemented this.

Let's take $K_1(\theta)$ to be the performance objective function and $K_2(\theta)$ to be the robustness objective function. We can construct a weighted combined objective function
$\hat{K}(\theta) = (\beta_1 + \lambda\beta_2) \cdot K_1(\theta) + \beta_2 \cdot K_2(\theta)$. The gradient ascent update is similar as described in subsection Subsection 2-1-1, and can be seen in (5-2) along with the update for the Lagrange multiplier λ.

$$\begin{aligned} \theta &= \theta + \nabla_\theta \hat{K}(\theta) \\ \lambda &= \max(0, \lambda + \eta(\hat{k}_1 - \varepsilon - K_1(\theta))) \end{aligned} \tag{5-2}$$

In these equations, β_1, β_2 and η are learning rates, $\hat{k}_1$ is an estimate of the optimum and $K_1(\theta)$ is a sampled objective value. We can see that if the term $(\hat{k}_1 - \varepsilon - K_1(\theta))$ is smaller than 0, meaning that the difference between the sampled objective and the estimated optimum is smaller than ε. Thus within the acceptable tolerance, λ will move toward zero (or stay at zero if already there). This means that the combined objective $\hat{K}(\theta)$ will prioritise the $K_2(\theta)$ objective more than if $\lambda > 0$. If the difference between the sampled objective and estimated optimum is greater than ε, λ will increase, putting the focus of the combined objective towards the performance objective $K_1(\theta)$. This is how to synchronously optimise the two objectives, while still being able to prioritise in a lexicographic manner. How these functions fall into the overall optimization algorithm, can be seen in Algorithm 8, which comes directly from [5]. Figure 5-1 shows a graphical representation of the Lagrangian optimisation to obtain a policy that is both maximally robust and at most ε away from the set of optimal policies [5].

The performance objective $K_1(\theta)$ is determined simply by the latest return. The optimum estimate is determined by taking the mean of the return of the latest episodes. In the implementation by Daniel Jarne Ornia in [43], he used the return of the last 25 episodes. Determining the robustness objective $K_2(\theta)$ is slightly less straightforward. In [43], actions are compared that are optimal for the true state, and optimal for the state disturbed by the noise kernel. It takes the Mean Squared Error (MSE) between these two sets of action probabilities as $K_2(\theta)$. As for noise, [43] compares a uniform noise kernel, a Gaussian noise kernel and adversarial noise.

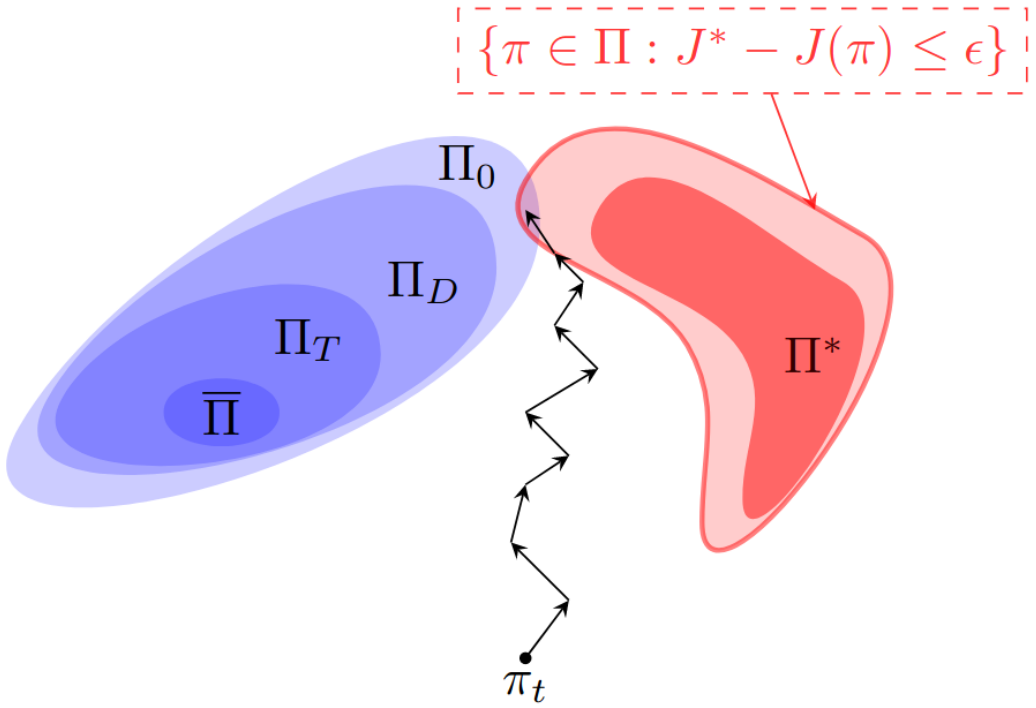


Figure 5-1: Graphical representation of the LRRL algorithm. We can see a visualisation of the inclusion theorem, and the policy converging to a maximally robust policy, while still being within the tolerance from the optimal policy. Figure from [5].

In [5], this algorithm is implemented for single-agent Proximal Policy Optimisation (PPO) and single-agent Advantage Actor-Critic (A2C) in the paper. On the GitHub repository [43]. However, it is also implemented for Deep Deterministic Policy Gradient (DDPG). This implementation of LRRL for the DDPG algorithm would be a good starting point for our extension to the Multi-Agent Deep Deterministic Policy Gradient (MADDPG) algorithm.

For proof of convergence, we will refer to [27], which proves that all policy-based LRL algorithms inherit the convergence guarantees of the single-objective algorithm. This proof is also adopted by [5], and we shall do the same and consider our MADDPG-LRRL algorithm to converge similarly as a vanilla MADDPG algorithm.

5-2 Algorithm Adaptations

The implementation of LRRL as described in [5] can be found in the repository in [43]. It was possible to almost directly copy the implementation, though there were still some design choices to be made.

If we want to incorporate the LRRL algorithm into the MADDPG algorithm, we will replace line 13 in Algorithm 1, which calculates the actor loss, with Algorithm 9. Besides this designed algorithm, some choices still need to be made. First of all, what noise kernel to use. We could test both

Algorithm 8 Lexicographically robust policy gradient [5]

Inputs:
MDP
$\tilde{T}$
ε
Initialize:
θ
critic
λ
$\{\beta_1, \beta_2, \eta\}$
$t = 0$
$x_t \sim \mu_0$
while $t < \max\_\text{iterations}$ **do**
  Perform $u_t \sim \pi_\theta(x_t)$
  Observe $r_t,\ x_{t+1}$
  if $\hat{K}_1(\theta)$ not converged **then**
    $\hat{k}_1 \leftarrow \hat{K}_1(\theta)$
  end if
  Update critic
  Update θ and λ using (5-2)
end while
Outputs:
θ

a uniform and a Gaussian noise kernel for our purposes. The adversarial noise also tested by [5] seems irrelevant to our research.

Another choice was at what point to start, including the robustness loss in the objective function. When including the robustness loss too early, before the main reward is found, this can lead to a local minimum that is driven by the robustness loss and will not converge. However, since we use non-sparse rewards, this problem should be mitigated slightly. Therefore we can start including the robustness loss at about 10% of training.

5-3 Experimenal Validation of Implementation

Two tests were executed to validate our implementation of the LRRL algorithm. We trained two different versions of observationally robust policies. One using a uniform noise kernel (LRRL$_{T_1}$) and one using a Gaussian noise kernel (LRRL$_{T_2}$).

The conclusion resulting from these experiments can be found in Subsection 5-3-1.

Constant policy production The initial test excludes $K_1(\theta)$ from the objective and focuses the policy's training solely on achieving robustness. This should result in a maximally robust policy. As the most straightforward path to maximising robustness is adopting a constant policy $\bar{\Pi}$, the algorithm will likely converge to this outcome due to its simplicity. This test is run for 1000 episodes. This test is more of a sanity check.

Algorithm 9 New actor loss when using LRRL

Inputs:
$\beta_1, \beta_2, \eta, \varepsilon, \lambda$
Noise kernel $T()$
Determine actor loss $K_1(\theta)$ according to MADDPG (Algorithm 1)
Set $\hat{k}_1$ as the mean of the last 25 values of $K_1(\theta)$
Determine robustness loss:
$\tilde{x} = T(x)$
$K_2(\theta) = 0.5 \cdot \text{mean}((\mu(x) - \mu\tilde{x})^2)$
$\hat{K}(\theta) = (\beta_1 + \lambda\beta_2) \cdot K_1(\theta) + \beta_2 \cdot K_2(\theta)$
Update actor to minimise $\hat{K}(\theta)$
Update weights:
$\lambda = \max(0, \lambda + \eta(\hat{k}_1 - \varepsilon - K_1(\theta)))$
Outputs:
λ
Updated actor

Hypothesis 7. *For a correct implementation, if the objective function consists solely of the robustness regret, the training will produce a constant policy.*

We validated our hypothesis by visual inspection of simulated episodes with the trained policies, where we observed that the policy seemed to be constant, the agents always moving in the same direction or remaining still, independent of the system state.

Performance under noise The second test is to run episodes for all implementations with different state disturbances and record the average performance over 1000 episodes. The other noise kernels are no noise ($\varnothing$), uniform noise (T_1) and Gaussian noise (T_2).

Hypothesis 8. *For a correct implementation, we expect the policies to perform similarly when no noise kernel is disturbing the state, but the lexicographically robust policy performs better under state disturbances.*

	Policy		
Noise	Vanilla	LRRL$_{T_1}$	LRRL$_{T_2}$
$\varnothing$	24.2 ± 17.7	28.1 ± 18.1	32.3 ± 18.0
T_1	1.6 ± 15.6	6.8 ± 15.3	3.2 ± 15.3
T_2	-63.2 ± 18.1	-47.8 ± 16.2	-27.8 ± 18.8

Table 5-1: Adversary return comparison of Vanilla policy and LRRL policy.

We can see the results of this test in Table 5-1. It can be seen that without noise, the LRRL policies perform slightly better, though in the same order of magnitude. When noise is added, the LRRL policies still perform better, as expected. The most significant difference can be seen in LRRL$_{T_2}$ with noise kernel T_2.

5-3-1 Validation Conclusion

The results indicate that the LRRL policy is correctly implemented. However, there is potential for further optimisation of hyperparameters, although such fine-tuning falls beyond the scope of this research. Since it Table 5-1 the most significant gain was had by LRRL$_{T_2}$, we shall choose this policy for the rest of our report as LRRL policy.

Chapter 6

Software Architecture

This chapter shall cover the most critical aspects of the software. First, we will discuss the general flow and architecture of training and execution in Section 6-1. Next, a simplified class diagram shall be presented in Section 6-2. The architecture of the neural networks will be given in Section 6-3, and finally, the most critical hyperparameters are shown in Section 6-4. The full code can be found in our GitHub repository [29].

6-1 Flow and Architecture

We will explain the architecture and flow of the entire algorithm with the help of the diagram given in Figure 6-1. The policies will decide actions based on observations, which are used to update the Multi-Particle Environments (MPE) environment. In environment A (as described in Subsection 3-2-2), the output of the MPE environment, such as the positions and the rewards, is used directly, as indicated by the dotted line. In environments B and C, the positions determined by the MPE environment are merely used as waypoints by the controller, which determines the robot's control action. This is either the linear and angular velocity for the Python robot model, or the wheel rotational speeds for the Webots implementation. Then the robot position is updated, and these positions are used by most other blocks in Figure 6-1. Suppose Event-Driven Interactions (EDI) mode is activated. In that case, these positions will be adjusted to incorporate the last communications by the agents, and then the resulting observations will be the following input for the policy block. If EDI mode is not activated, and periodic communication is employed, the EDI block is replaced by the dotted line and the positions are used directly.

It can also be seen that the Q-functions are only available during training, and not during execution. We need Q-values to train the neural networks for the policies and the robustness surrogate.

6-2 Class Diagram

The code used for this project contains a lot of classes. Therefore we constructed a simplified class diagram containing the relevant classes and only the most essential functions of every class.

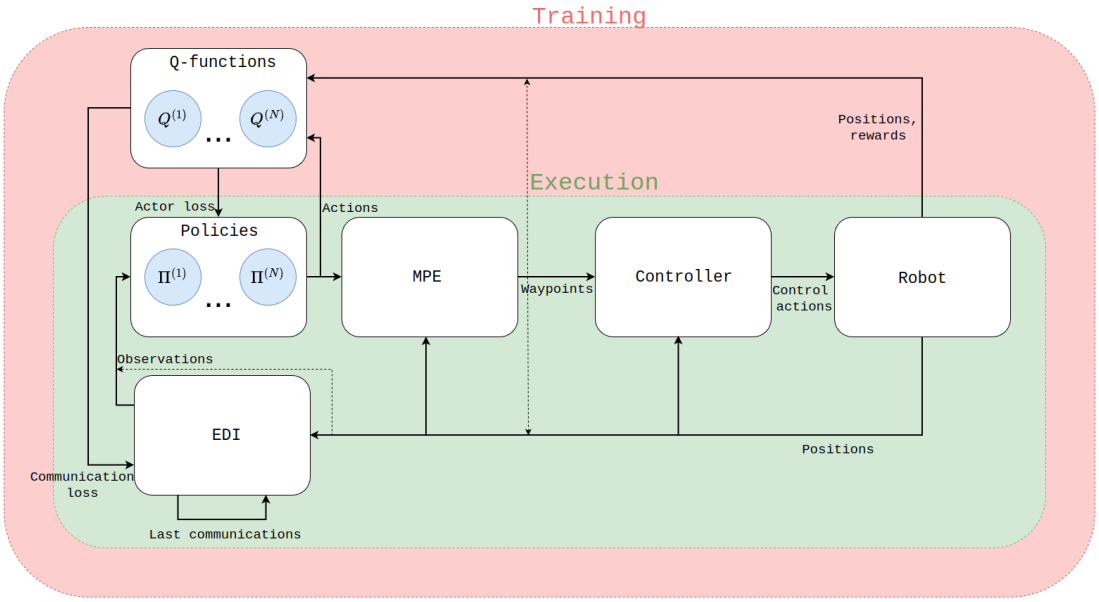


Figure 6-1: Diagram of the flow and architecture of the full algorithm.

This class diagram can be found in Figure 6-2. The `Train` class is the top-level class that sets up all the other classes. This class is called in the `main.py` script. For more details on the classes, please consult the `REAMDE` and the code comments provided in the GitHub repository[29].

6-3 Neural Networks

For this project, three different neural network architectures are used; one for the critic networks, one for the actor networks and one for the robustness surrogates. Since we took the Multi-Agent Deep Deterministic Policy Gradient (MADDPG) code from [35], the network architecture is the same as on this repository.

Critic Network The critic network has three linear layers. The first layer has the size of the observation space and action space of all agents as input. The output size is 64. The second layer has an input and output size of 64. The third layer has an input size of 64, and an output size of 1, which will be the Q-value. The first and second layers have a relu activation function. This network uses an Adam optimizer.

Actor Network The actor network is fairly similar, with again three linear layers. The input dimension is the size of the observation space of the agent the network belongs to. The first layer has an output size of 64. The second layer has an output size and input size of 64, and the third layer has an input size of 64 and an output size of the action space, which is 5. The first two layers use a relu activation function, and the third uses a softmax activation function. It needs a softmax activation function because the entries in the action vector should be $\in [0, 1]$. This network uses an Adam optimizer.

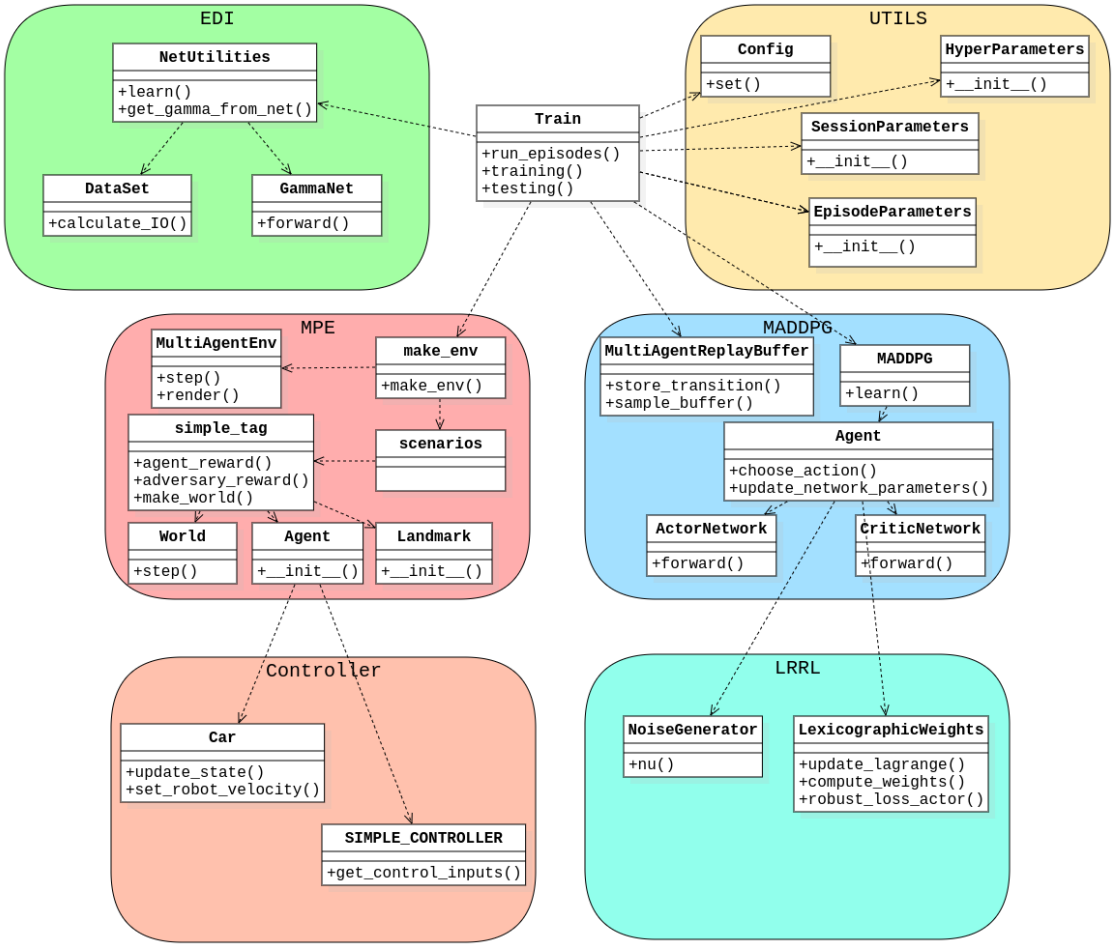


Figure 6-2: Simplified class diagram of the software.

Robustness Surrogates Network We made the network for the robustness surrogates similar to the actor and critic networks, since the inputs are similar and the computational complexity should be in the same order of magnitude. So this network consists of four linear layers, most with an input and output size of 64, except for the first layer, which has an input size of 2 observations of one agent, and the last layer, which has an output size of 1. The first three layers have relu activation functions, the last one does not have an activation function. This network also uses an Adam optimizer.

6-4 Hyperparameters

The important parameters used for this research can be seen in Table 6-1.

Symbol	Short explanation	Value
γ	Discount factor	0.99
τ	Target network learning rate	0.01
ϵ	Greediness	$\max\left((1 - \text{episode}/\text{maximum episodes}) \cdot 0.9,\ 0.27\right)$
β_1	Lexicographic learning rate	1
β_2	Lexicographic learning rate	2
η	Lexicographic learning rate	0.002
ε	Lexicographic tolerance	0.5
r	Wheel radius	0.0042m
b	Distance wheel to midline	0.02m
Δ_t	Time-step	0.25s
h	Control horizon	4

Table 6-1: Table of important parameters used for this research.

Chapter 7

Experimental Results

This chapter will describe the performed experiments, we organised the experiments by environment, described in Subsection 3-2-2. We shall first describe the experiments performed in Section 7-1, after which we shall give the results of these experiments in Section 7-2.

7-1 Experiments

We performed three kinds of experiments:

Sim-to-real gap The first is to see if the sim-to-real gap is mitigated using the Lexicographically Robust Reinforcement Learning (LRRL) policy when switching to an environment with more realistic kinematics or dynamics. This experiment is only performed for the B and C environments, since environment A is the starting point for which we have no networks pre-trained on less realistic environments. If we recall Hypothesis 2, we expect the LRRL policy to perform better when no additional training is done. This experiment consists of testing the vanilla (Algorithm 1) and LRRL (Algorithm 9) algorithms that are trained on the previous environment, with Event-Driven Interactions (EDI) algorithm (Algorithm 6 and Algorithm 7) activated, for 1000 episodes and noting the average adversary return.

ζ for steps between states The second is to take sequences of states and determine the ζ value returned by the robustness surrogate neural network for random pairs of states within that sequence. Since ζ is the *smallest* ζ_{th} for which the two states are still in each other's Γ_ζ, the lower the ζ is, the longer the policy can go without communicating. For this experiment, 1000 pairs of states are tested.

Communications and adversary return The third experiment is to run episodes with EDI mode activated and compare the number of communications and the adversary return for different values of ζ_{th}. If we recall Hypothesis 1, we expect both the return and the number of communications to

decrease for higher values of ζ_{th}, and we expect the LRRL policy to be able to achieve the same return with fewer communications.

7-2 Results

We will give the results of the experiments described above per environment, starting with environment A in Subsection 7-2-1, then environment B in Subsection 7-2-2 and ending with environment C in Subsection 7-2-3. Finally, we give some results comparing the environments in Subsection 7-2-4.

7-2-1 Experiments in Environment A

We shall now give the results of the experiments done in environment A.

ζ for steps between states The results of this experiment can be seen in Figure 7-1. We can see that the line for the LRRL policy lies lower than that for the vanilla policy, as we expected. As mentioned in Section 4-3, the lines are not monotonically increasing, which goes against expectations.

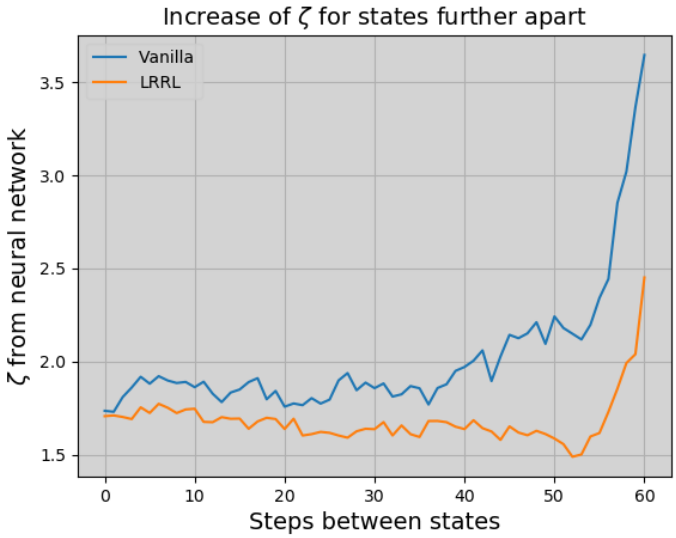


Figure 7-1: Experimental results for the increase of ζ for states further apart for environment A.

Communications and adversary return The results of this experiment are visualised in two graphs, Figure 7-2 and Figure 7-3. Figure 7-2 is a graph of the number of communications and adversary return plotted for an increasing ζ_{th}. The baseline of both policies, with EDI mode deactivated, is noted in Table 7-1.

	Adversary return	Number of communications
Vanilla	24.2	122
LRRL	32.3	122

Table 7-1: Baseline values, with EDI mode deactivated for environment A.

We can see in Figure 7-2 that, indeed, both the communications and adversary return go down for increasing ζ_{th}. It is also noticeable that while the adversary return for the LRRL policy is only slightly lower than that of the vanilla policy, it needs significantly fewer communications, especially for lower ζ_{th}.

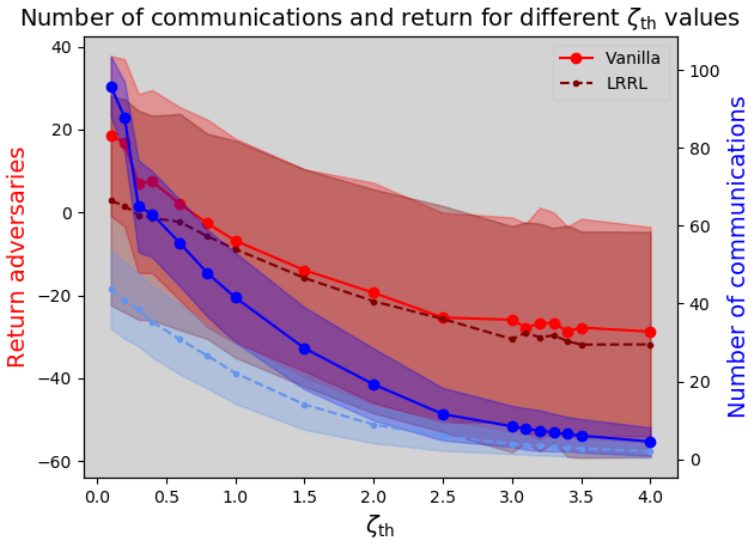


Figure 7-2: Experimental results for the adversary return and number of communications for environment A.

The second graph, Figure 7-3, shows the number of communications plotted versus the adversary return for both the vanilla and LRRL policy. We can see that the line for the LRRL policy lies above the line for the vanilla policy. What is also noticeable is that the LRRL line is much shorter than the vanilla one, indicating that the highest number of communications for the LRRL policy is a lot lower than the highest number of communications for the vanilla policy.

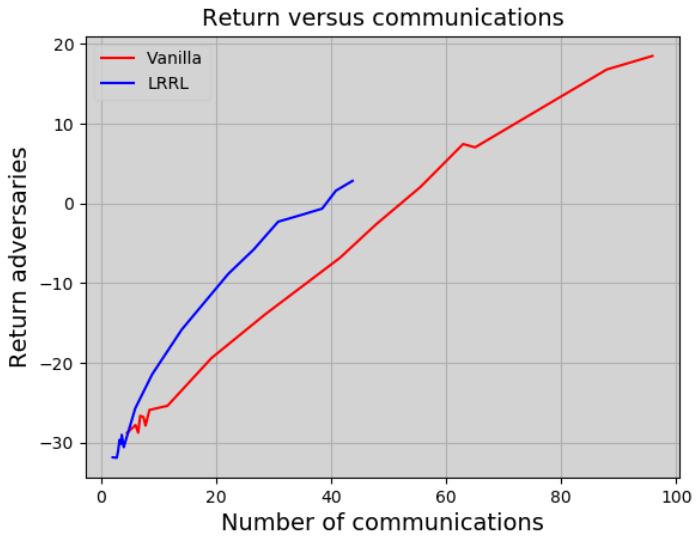


Figure 7-3: Adversary return versus the number of communications for environment A.

7-2-2 Experiments in Environment B

Here we will give the results of the experiments done in environment B. For all experiments, with the exception of the sim-to-real gap experiment, some additional training was done first in this environment.

Sim-to-real gap The results of this experiment are noted in Table 7-2. It can be seen that the adversary return for the LRRL policy is slightly higher, though the difference seems negligible.

Policy	Adversary return
Vanilla	-5.9
LRRL	-5.4

Table 7-2: Experimental results for the sim-to-real gap experiment for environment B.

ζ **for steps between states** The results of this experiment can be seen in Figure 7-4. Also, for this environment, the line for the LRRL policy lies lower than the line for the vanilla policy, as expected, though the difference here is much more pronounced.

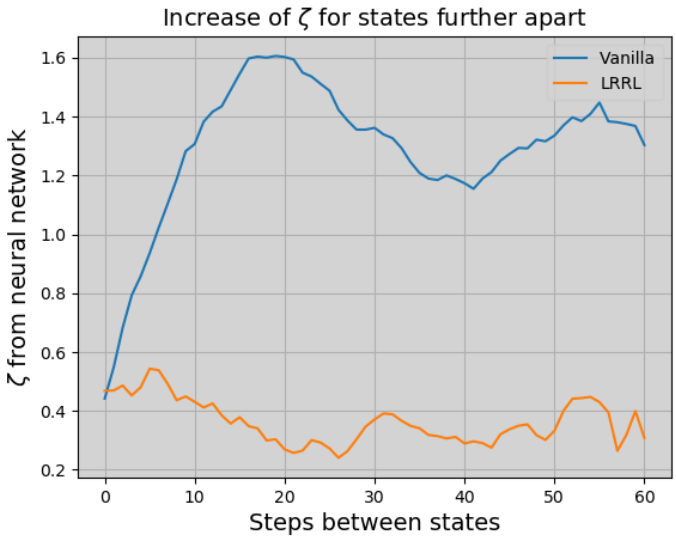


Figure 7-4: Experimental results for the increase of ζ for states further apart for environment B.

Communications and adversary return The two resulting graphs for this experiment can be seen in Figure 7-4 and Figure 7-5. Figure 7-5 is a graph of the number of communications and adversary return plotted for an increasing ζ_{th}. The baseline of both policies, with EDI mode deactivated, is noted in Table 7-3. When comparing Table 7-2 to Table 7-3, it is noticeable that the baseline returns for this environment are lower, after additional training, than the returns when using the policies only trained in environment A.

	Adversary return	Number of communications
Vanilla	-20.1	122
LRRL	-24.4	122

Table 7-3: Baseline values, with EDI mode deactivated for environment B.

For this graph, again, all lines go down for increasing ζ_{th}, only for this environment, the gain obtained by the LRRL policy is more pronounced. The adversary return line for the LRRL policy is now above the one for the vanilla policy. The difference in Figure 7-6 is more pronounced here than in environment A. It can also be seen that the standard deviation for the number of communications drops down to zero after $\zeta_{\text{th}} = 3$, when the mean is also zero, indicating there is no communication at all. Another interesting thing is that the overall return is a lot lower than it was in environment A.

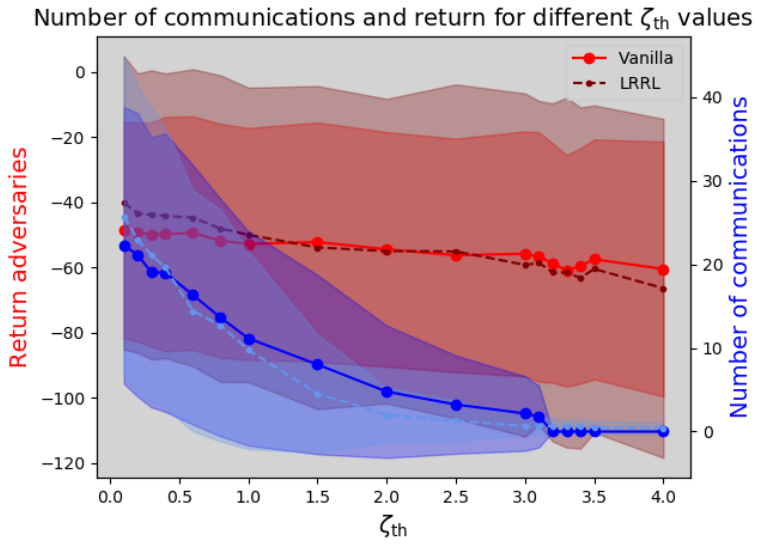


Figure 7-5: Experimental results for the adversary return and number of communications for environment B.

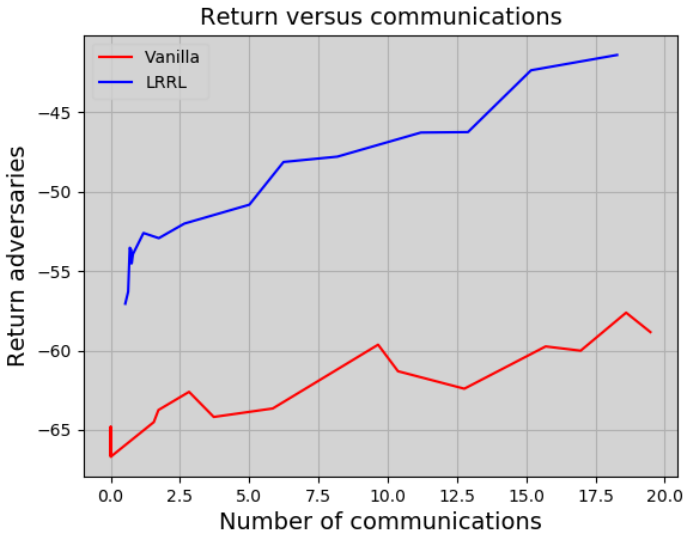


Figure 7-6: Adversary return versus the number of communications for environment B.

7-2-3 Experiments in Environment C

This subsection will give the results of the experiments done in environment C. For this environment, no additional training was done.

Sim-to-real gap The results of this experiment are noted in Table 7-4. We tested the policies trained in environment A and environment B. Interestingly, the policies trained in environment

B, which should be more similar to environment C, performed worse than the policies trained in environment A. The LRRL policy performed worse for both cases.

Policy	Adversary return policy env A	Adversary return policy env B
Vanilla	21.8	−7.9
LRRL	17.0	−39.8

Table 7-4: Experimental results for the sim-to-real gap experiment for environment C.

ζ for steps between states The results for this experiment can be seen in Figure 7-7. Once again, the line for the LRRL policy can be found lying lower than the line for the vanilla policy, though they are more similar than in the results for environment B.

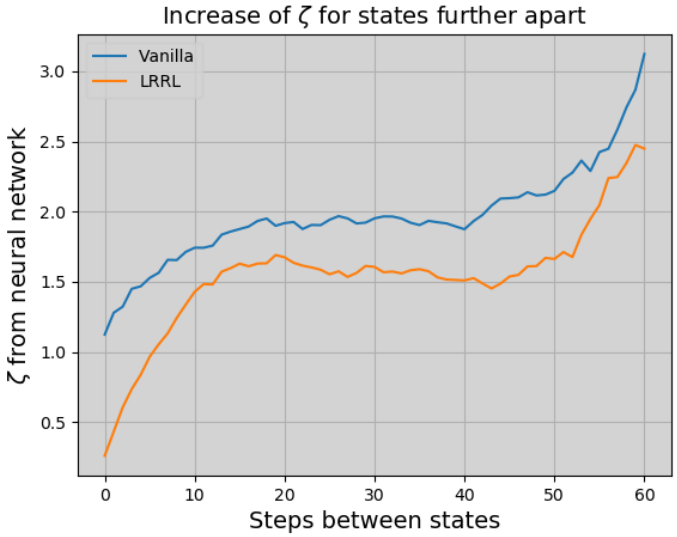


Figure 7-7: Experimental results for the increase of ζ for states further apart for environment C.

Communications and adversary return The two resulting graphs for this experiment can be seen in Figure 7-8 and Figure 7-9. The baseline for both policies with full communication can be seen in Table 7-5.

	Adversary return	Number of communications
Vanilla	21.9	122
LRRL	17.0	122

Table 7-5: Baseline values, with EDI mode deactivated for environment C.

In Figure 7-8, we can see again the downward slopes of the return and communication lines for both policies. The performance of the LRRL policy is similar, with less communications. In Figure 7-9, we can see that the line for the LRRL policy lies slightly above the line for the vanilla policy,

meaning the LRRL policy can achieve higher returns with the same number of communications. In general, the return is higher than in environment B.

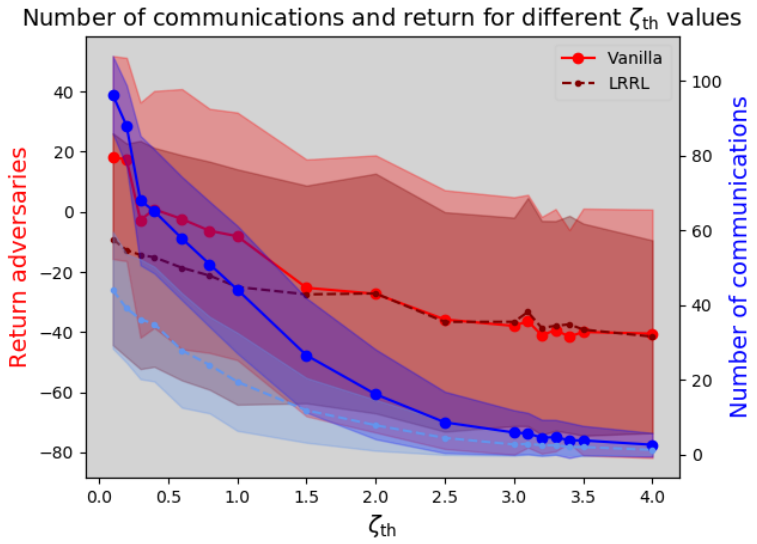


Figure 7-8: Experimental results for the adversary return and number of communications for environment C.

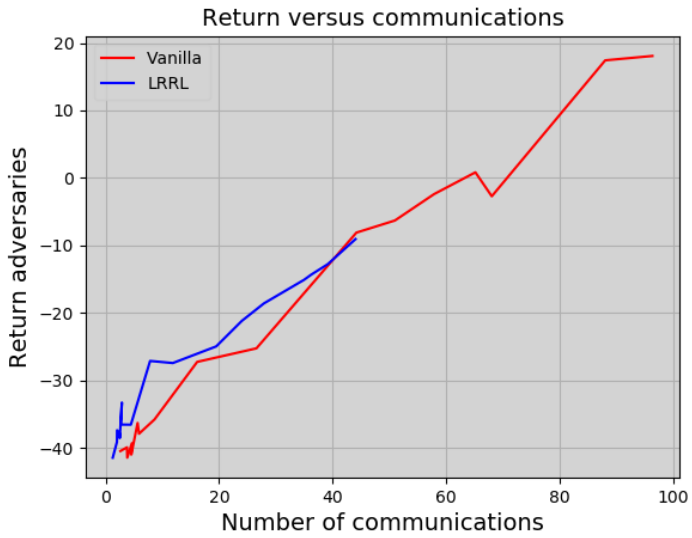


Figure 7-9: Adversary return versus the number of communications for environment C.

7-2-4 Comparison of return increase per environment

For some last comparisons, we constructed a graph showing the increase in adversary return obtained using the LRRL algorithm versus a number of communications. We can see these

results in Figure 7-10. The lines for all three environments seem similar, indicating no significant differences between the environments in how much gain is to be had by using a LRRL policy. The gain is increasing for more communications.

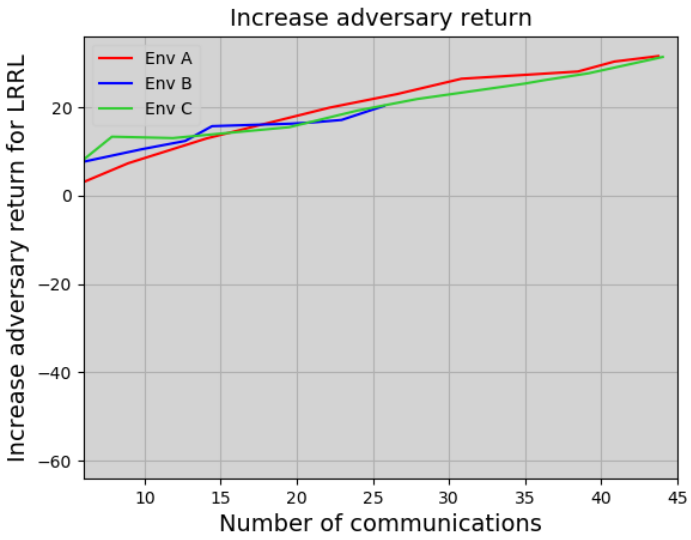


Figure 7-10: Increase in adversary return obtained by LRRL versus number of communications.

Chapter 8

Conclusion

This chapter will first give a qualitative discussion (Section 8-1) on the results given in Chapter 7. This discussion will lead to the main conclusions from this research in Section 8-2. Finally, we shall talk about some possible future research in Section 8-3.

8-1 Discussion

First of all, from Table 7-2 and Table 7-3, it is clear that the additional training in environment B did not achieve any increase in performance. This seems counter-intuitive since additional training in a slightly different environment should better equip the policy to perform in that environment. It could be that this training cycle did not converge or some hyperparameters were not set correctly. Since this is difficult to determine without running more lengthy experiments, we will not put much value on the results from environment B, and will not discuss them further. Since we did not do any additional training in environment C, due to the time it takes to train in Webots, this environment does not experience this issue.

Now let's talk about the sim-to-real gap. In environments B and C, the vanilla policy trained in environment A performed better than the Lexicographically Robust Reinforcement Learning (LRRL) policy trained in environment A. The difference in environment B is small enough to be negligible but is more significant in environment C. This is contrary to Hypothesis 2.

Concerning Hypothesis 1, there seems to be more evidence supporting its validity. For all three environments, the results for the ζ ***for steps between states*** experiments, are indicative that the LRRL policy has lower ζ values for sets of states, which indicates there can be more difference between states before the ζ will cross the ζ_{th}. The results for the ***Communications and adversary return*** experiments all indicate that the LRRL policies can achieve the same performance as the vanilla policies, while using fewer communications. This is in accordance with Hypothesis 1.

When looking at Figure 7-10, we can see that for all environments, there is an increase larger than zero in the return when using the LRRL policy, which is another indication of the validity of Hypothesis 1. This gain is more pronounced for a higher number of communications.

8-2 Conclusion

We presented an extension of the Event-Driven Interactions (EDI) method from [4] to accommodate continuous state and action spaces and proved its validity. We also extended the LRRL method from [5] to the multi-agent case and implemented both for the Multi-Agent Deep Deterministic Policy Gradient (MADDPG) algorithm from [3] on a modified simple-tag (also called particle-tag or predator-prey) domain.

We conducted tests on these methods individually and in combination, across three distinct environments that progressively increased in realism. We had expected the LRRL policy to help mitigate the sim-to-real gap, as per Hypothesis 2, but have shown that this is not the case. Hypothesis 1 appears to be hold true, and the increase in observational robustness obtained by LRRL allows the agents to attain equivalent returns while communicating less.

This means that the combination of these two methods can be used in multi-robot systems to help the demands on the bandwidth of communication channels, and thus help with the scalability of systems.

8-3 Future Work

There are many additional steps to take to improve (Subsection 8-3-1) this research or extend it (Subsection 8-3-2). This research can be seen as a proof of concept to the validity of using the EDI and LRRL methods combined, but there is still a lot to be gained by exploring further.

8-3-1 Improvements to this research

For the implementation as is, there are already some improvements or additional experiments to perform:

- It would be good to look at the additional training in the B and C environments for improvements. We think we could see some improved results when there is some proper additional training.
- We would also like to upgrade our simple lower-level controller of Algorithm 3 to a proper Model Predictive Control (MPC) controller. This will help with the performance in the B and C environments.
- Furthermore, we think that there should be more hyperparameter tuning. Minimal tuning was done for this research since the full training using new parameters takes a long time, thus we considered it to be out of scope for this research. We hope that with better hyperparameter tuning, the LRRL policy might obtain more robustness, which will make the EDI method even more effective.

Once these minor improvements are made, all experiments can be repeated. Additionally, we think it would be useful to perform tests with more than two communicating agents, to assess the scalability of the method. Finally, we would like to test on real Elisa-3 robots.

8-3-2 Extensions

There are also some extensions to make the method more generally applicable. The robustness surrogates still need to know the Q-values for training, making the method unsuitable for purely policy-based methods. A possible direction to take this research is to not look at the Q-values but at the number of time steps between communications.

Afterwards, extending the method to more Multi-Agent Reinforcement Learning (MARL) algorithms will be possible.

Bibliography

[1] H. Dong, Z. Ding, and S. Zhang, eds., *Deep Reinforcement Learning: Fundamentals, Research and Applications*. Singapore: Springer Singapore, 2020.

[2] W. P. M. H. Heemels, M. C. F. Donkers, and A. R. Teel, "Periodic Event-Triggered Control for Linear Systems," *IEEE Transactions on Automatic Control*, vol. 58, pp. 847–861, Apr. 2013.

[3] R. Lowe, Y. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments," *Advances in neural information processing systems*, vol. 30, 2017.

[4] D. J. Ornia and M. Mazo Jr, "Robust Event-Driven Interactions in Cooperative Multi-Agent Learning," in *Lecture Notes in Computer Science*, vol. 13465, pp. 281–297, 2022. arXiv:2204.03361 [cs, eess].

[5] D. J. Ornia, L. Romao, L. Hammond, M. Mazo Jr., and A. Abate, "Observational Robustness and Invariances in Reinforcement Learning via Lexicographic Objectives," Sept. 2022. arXiv preprint, arXiv:2209.15320 [cs].

[6] L. Busoniu, R. Babuska, and B. De Schutter, "A Comprehensive Survey of Multiagent Reinforcement Learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 38, pp. 156–172, Mar. 2008.

[7] S. Han, S. Su, S. He, S. Han, H. Yang, and F. Miao, "What is the Solution for State-Adversarial Multi-Agent Reinforcement Learning?," Dec. 2022. axXiv preprint, arXiv:2212.02705 [cs].

[8] J. Hu and M. P. Wellman, "Multiagent Reinforcement Learning: Theoretical Framework and an Algorithm," *ICML*, vol. 98, no. 242-250, 1998.

[9] L. Panait and S. Luke, "Cooperative Multi-Agent Learning: The State of the Art," *Autonomous Agents and Multi-Agent Systems*, vol. 11, pp. 387–434, Nov. 2005.

[10] C. Verdier and M. Mazo, Jr., "Formal Controller Synthesis via Genetic Programming," *IFAC-PapersOnLine*, vol. 50, pp. 7205–7210, July 2017.

[11] C. F. Verdier and M. Mazo Jr, "Formal Synthesis of Analytic Controllers for Sampled-Data Systems via Genetic Programming," Dec. 2018. arXiv preprint, arXiv:1812.02711 [cs].

[12] C. F. Verdier and M. Mazo Jr, "Formal controller synthesis for hybrid systems using genetic programming," Sept. 2020. arXiv preprint, arXiv:2003.14322 [cs, eess].

[13] C. F. Verdier, N. Kochdumper, M. Althoff, and M. Mazo, "Formal synthesis of closed-form sampled-data controllers for nonlinear continuous-time systems under STL specifications," *Automatica*, vol. 139, p. 110184, May 2022.

[14] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy Gradient Methods for Reinforcement Learning with Function Approximation," *Advances in neural information processing systems*, vol. 12, 2000.

[15] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic Policy Gradient Algorithms," *International conference on machine learning*, pp. 387–395, 2014.

[16] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," July 2019. arXiv preprint, arXiv:1509.02971 [cs, stat].

[17] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.

[18] J. Foerster and I. A. Assael, "Learning to Communicate with Deep Multi-Agent Reinforcement Learning," *Advances in Neural Information Processing Systems*, vol. 29, 2016.

[19] A. Das, T. Gervet, J. Romoff, D. Batra, D. Parikh, M. Rabbat, and J. Pineau, "TarMAC: Targeted Multi-Agent Communication," *International Conference on Machine Learning*, pp. 1538–1546, 2019.

[20] T. Chen, K. Zhang, G. B. Giannakis, and T. Başar, "Communication-Efficient Policy Gradient Methods for Distributed Reinforcement Learning," *Transactions on Control of Network Systems*, vol. 9, pp. 917–929, Apr. 2021. arXiv:1812.03239 [cs, stat].

[21] J. R. Kok and N. Vlassis, "Sparse cooperative Q-learning," *Twenty-first international conference on Machine learning - ICML '04*, p. 61, 2004.

[22] D. J. Ornia and M. Mazo Jr, "Event-Based Communication in Distributed Q-Learning," *IEEE 61st Conference on Decision and Control (CDC)*, no. 61, pp. 2379–2386, 2022. arXiv:2109.01417 [cs].

[23] W. Heemels, K. Johansson, and P. Tabuada, "An introduction to event-triggered and self-triggered control," *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pp. 3270–3285, Dec. 2012.

[24] A. Mandlekar, Y. Zhu, A. Garg, L. Fei-Fei, and S. Savarese, "Adversarially Robust Policy Learning: Active construction of physically-plausible perturbations," *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3932–3939, Sept. 2017.

[25] M. Everett, B. Lutjens, and J. P. How, "Certifiable Robustness to Adversarial State Uncertainty in Deep Reinforcement Learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, pp. 4184–4198, Sept. 2022.

[26] L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta, "Robust Adversarial Reinforcement Learning," *International Conference on Machine Learning*, pp. 2817–2826, 2017.

[27] J. Skalse, L. Hammond, C. Griffin, and A. Abate, "Lexicographic Multi-Objective Reinforcement Learning," *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*, pp. 3430–3436, July 2022.

[28] D. P. Bertsekas, *Nonlinear Programming*. Belmont, Massachusetts: Athena Scientific, 2 ed., 1997.

[29] J. de Gooijer, "Efficient Communication in Robust Multi-agent Reinforcement Learning," *https://github.com/J-deGooijer/Efficient-Communication-in-Robust-Multi-agent-Reinforcement-Learning*. *Last update: 10.07.2023*.

[30] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," Aug. 2017. arXiv preprint, arXiv:1707.06347 [cs].

[31] V. R. Konda and J. N. Tsitsiklis, "Actor-Critic Algorithms," *Advances in neural information processing systems*, vol. 12, 1999.

[32] J. Hale, S. Sen, and M. Sekaran, "Learning to Coordinate without Sharing Information," *AAAI*, vol. 94, pp. 426–431, 1994.

[33] Y. Jiang, T. Zhang, D. Ho, Y. Bai, C. K. Liu, S. Levine, and J. Tan, "SimGAN: Hybrid Simulator Identification for Domain Adaptation via Adversarial Reinforcement Learning," *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2884–2890, May 2021. arXiv:2101.06005 [cs].

[34] R. Lowe, C. Hesse, H. Shekhar, C. Berner, Y. Wu, and W. Wang, "Multi-Agent Deep Deterministic Policy Gradient (MADDPG)," *https://github.com/openai/maddpg*. *Accessed: 02.04.2023*.

[35] P. Tabor, "Multi-Agent-Deep-Deterministic-Policy-Gradients," *https://github.com/philtabor/Multi-Agent-Deep-Deterministic-Policy-Gradients*. *Accessed: 02.04.2023*.

[36] G. Matheron, N. Perrin, and O. Sigaud, "The problem with DDPG: understanding failures in deterministic environments with sparse rewards," 2019. arXiv preprint, arXiv1911.11679[cs].

[37] R. Lowe, "Multi-Agent Particle Environments," *https://github.com/openai/multiagent-particle-envs*. *Accessed: 02.04.2023*.

[38] F. Foundation, "Multi Particle Environment," *https://pettingzoo.farama.org/environments/mpe/*. *Accessed: 01.05.2023*.

[39] GCtronics, "Elisa-3," *https://www.gctronic.com/doc/index.php/Elisa-3*. *Accessed: 30.03.2023*.

[40] GCtronics, "E-Puck," *https://www.gctronic.com/doc/index.php?title=E-Puck*. *Accessed: 11.07.2023*.

[41] Zainkhan-afk, "Differential-Drive-Robot-Naviation," *https://github.com/zainkhan-afk/Differential-Drive-Robot-Navigation*. *Accessed: 04.07.2023*.

[42] M. C. Campi, S. Garatti, and M. Prandini, "The scenario approach for systems and control design," *Annual Reviews in Control*, vol. 33, pp. 149–157, Dec. 2009.

[43] D. Jarne Ornia, "LRRL," *https://github.com/danieljarne/LRRL*. *Accessed: 04.04.2023*.

Glossary

List of Acronyms

RL	Reinforcement Learning
MARL	Multi-Agent Reinforcement Learning
LRL	Lexicographic Reinforcement Learning
LRRL	Lexicographically Robust Reinforcement Learning
A2C	Advantage Actor-Critic
PPO	Proximal Policy Optimisation
MDP	Markov Decision Process
POMDP	Partially Observable Markov Decision Process
DOMDP	Observationally Disturbed Markov Decision Process
ETC	Event-Triggered Control
EDI	Event-Driven Interactions
AC	Actor-Critic
PG	Policy Gradient
DPG	Deterministic Policy Gradient
DDPG	Deep Deterministic Policy Gradient
MADDPG	Multi-Agent Deep Deterministic Policy Gradient
DQN	Deep Q-Network
MPE	Multi-Particle Environments
MSE	Mean Squared Error
PID	Proportional-Integral-Differential
MPC	Model Predictive Control

List of Symbols

β_1	Learning rate
β_2	Learning rate
ϵ	Exploration rate
η	Learning rate
γ	Discount factor
Γ_ζ	Robustness surrogate
κ	Robustness indicator
λ	Lagrange multiplier
$\mathcal{O}_1, \ldots, \mathcal{O}_N$	Observations
μ	Deterministic agent policy
∇	Gradient operator
π	Stochastic agent policy
$\rho(\pi, T)$	Robustness regret
θ	Policy or network parameters
ε	Tolerance
ζ	Sensitivity parameter of robustness surrogate
$\hat{g}$	Gradient estimator
$\hat{K}(\theta)$	Combined objective in LRL
$\hat{k}_1$	Estimate of optimum of K_1
$\mathcal{U}$	Set of actions
$\mathcal{X}$	Finite set of states
$\varnothing$	Empty set
G_t	Return
K	Objective function
$K_1(\theta)$	Primary objective function (performance)
$K_2(\theta)$	Secondary objective function (robustness)
N	Number of agents
P	Transition relation
$Q^\pi(x, u)$	Q-function
R	Reward function
r	Reward
T	Noise kernel
u	Action
$V^\pi(x)$	Value function of state x using policy π
x	State
x'	Updated state
$(.)^*$	Indication of optimality

$(.)^\top$	Vector or matrix transpose
$(.)'$	Indicate next or updated state, or target network
$\hat{}$	Indication of estimated or outdated value

Index

Actor-critic, 7

Competitive stochastic game, 7
Cooperative stochastic game, 7

Differential drive robot, 17

Event-driven interactions, 9
Event-triggered control, 8

Inclusion theorem, 31

Lagrangian relaxation, 32
Lexicographically Robust Reinforcement Learning, 31
Lexicograpic reinforcement learning, 31
Lyapunov function, 9

Markov Decision Process, 5
Mixed stochastic game, 8
Multi-agent deep deterministic policy gradient, 11
Multi-Agent Reinforcement Learning, 7

Observational robustness, 8

Policy, 6
Policy gradient, 6

Q-function, 6

Reinforcement learning, 5
Return, 6
Robustness regret, 31
Robustness surrogates, 21

Stochastic game, 7

Value function, 6

Zeno behaviour, 9