Multi-Agent Actor-Critic Reinforcement Learning for Cooperative Tasks

Yagiz Efe Bayiz
Multi-Agent Actor-Critic Reinforcement Learning for Cooperative Tasks

MASTER OF SCIENCE THESIS

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

Yagiz Efe Bayiz

July 29, 2014

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology
For single-agent problems, Reinforcement Learning (RL) algorithms proved to be useful learning optimal control laws for nonlinear dynamic systems without relying on a mathematical model of the system to be controlled. With their ability to work on continuous action and state spaces, actor-critic RL algorithms are especially advantageous in that manner. So far, actor-critic methods have been applied to several single-agent control problems often with impressive results.

A Multi-Agent System (MAS) distributes computational resources and capabilities across a network of interconnected agents. The main advantage of using such an approach is to distribute a globally complex problem to simpler sub-problems, which is a more natural way to address source allocation and team planning. Application of MAS to domains, such as robotics, distributed control and telecommunications, gained popularity in last two decades. From the control point of view, cooperative MAS have a special importance since agents in control problems frequently seek to achieve a joint goal.

So far, a significant amount of research has been dedicated to Multi-Agent Reinforcement Learning (MARL) for both cooperative and non-cooperative tasks. Yet, the actor-critic methods in MARL context have not been examined in detailed. The aim of this project is to implement actor-critic RL methods to cooperative MAS to combine the advantages of these two approaches and apply the resulting methods to a real-life control problem as a proof of concept. To achieve such task Model Learning Actor-Critic (MLAC) algorithm is extended to two of the Independent Learners (IL) based methods: optimistic learners and lenient learners. The resulting algorithms are tested on 2-link manipulator problem. The results indicate that, the initial learning speed of the proposed multi-agent MLAC algorithms is similar or faster than the centralized MLAC at the start of learning experiments, and the end performance is acceptable compared to the centralized MLAC.
# Table of Contents

Acknowledgements v

1 Introduction 1
   1-1 Motivation ........................................... 1
   1-2 Research Objectives ................................ 2
   1-3 Outline ............................................. 3

2 Preliminaries 5
   2-1 Reinforcement Learning Framework ...................... 5
      2-1-1 RL in Markov Decision Process Framework .......... 6
      2-1-2 Returns, Value Functions and Optimality .......... 6
      2-1-3 Temporal Difference Concept ..................... 8
      2-1-4 Eligibility Traces ................................ 9
   2-2 Actor-Critic Methods .................................. 10
      2-2-1 Policy Gradient .................................. 11
      2-2-2 Actor-Critic Structure ............................ 12
   2-3 Summary ............................................. 13

3 Cooperative Multi-Agent Actor-Critic Learning 15
   3-1 Multi-Agent Systems (MAS) and cooperative MARL .......... 15
   3-2 Centralized Learning ................................ 16
   3-3 Multi-Agent Markov Decision Process .................. 18
   3-4 Independent Learners ................................ 19
   3-5 Optimistic Learners ................................ 21
   3-6 Lenient Learners .................................... 23
   3-7 Summary ............................................. 25
# 4 Results and Comparison

4-1 Learning Problem .................................................. 27
4-2 Function Approximation-Radial Basis Functions ..................... 31
4-3 Results ............................................................... 31
   4-3-1 Centralized MLAC ............................................. 32
   4-3-2 Optimistic Multi-Agent MLAC ................................ 33
   4-3-3 Lenient Multi-Agent MLAC ................................... 35
4-4 Comparison and Discussion of the Results .......................... 36
4-5 Summary ............................................................. 39

# 5 Conclusion

5-1 Future Work .......................................................... 42

## Bibliography

## Glossary

List of Acronyms .......................................................... 49
Acknowledgements

I would like to thank my supervisor prof.dr. R. Babuška his many helpful comments, suggestions and patience during the preparation this thesis.

I am also grateful for my girlfriend Tuğçe Vural who stuck with me through thick and thin and endlessly supported me.

I cannot end the acknowledgments part without expressing my gratitudes to my family and friends for their encouragement and guidance.

Delft, University of Technology

July 29, 2014

Yagiz Efe Bayiz
“In philosophy, or religion, or ethics, or politics, two and two might make five, but when one was designing a gun or an aeroplane they had to make four.”

— George Orwell, 1984
Chapter 1

Introduction

1-1 Motivation

In the field of robotics, as well as many others, unknown or complex environments can be a major challenge. Furthermore, robots may need to perform tasks that cannot always be predefined, such that some sort of adaptation or learning is required. A well-known and popular technique is called Reinforcement Learning (Reinforcement Learning (RL)), which is a machine learning technique that falls right in between the supervised learning [1] and unsupervised learning [2] in terms of the level of supervision needed to learn. RL techniques show close similarity on how animals learn considering the assessment of rewards, and have been successfully applied to several applications [3, 4, 5, 6]. In RL, the agent receives a scalar reward at each time step, and the goal is to find an optimal policy that maximizes the total accumulated rewards also known as the return. While following a given policy and collecting the rewards, the agent can estimate the return through a value function that allows the agent to indirectly update its policy towards the optimal one.

Over the course of time, a number of RL algorithms, have been developed and they are categorized in three groups: critic-only, actor-only and actor-critic methods [7]. In this classification, the actor stands for the policy followed and the critic represents the value function which criticizes the policy. This study focuses on the actor-critic class of the RL in which the presence of a parametrized policy allows the use of continuous action space, unlike frequently used critic-only methods while providing a decent evaluation of the policy followed through the critic. Actor-critic algorithms are proved to be an efficient, fast way of learning continuous policies for various applications [8, 9, 10, 11].
Multi-Agent Systems (MAS) can be defined as a group of agents interacting with each other in a common environment in which they try to act depending on their individual interests and goals [12]. In that sense, instead of having a centralized action taker, the control is distributed among the autonomous agents. Single-agent RL methods have been researched for a long time with well-understood fundamentals, and the RL techniques offers a simple solution to wide range of learning problems. Due to these properties, the RL framework is extended to handle multiple agents, and this framework is denoted as Multi-Agent Reinforcement Learning (MARL) .

In control and robotics applications, cooperative MARL, where agents try to achieve a common goal, attract attention due to the existence of numerous cooperative problems in these fields. Considering the nature of the problem, learning in a cooperative MAS can also be performed by a centralized RL technique. Yet, the size of the space to be searched for an optimal policy exponentially grows with the number of agents in the system when a centralized RL technique is preferred over a MARL method [13]. Hence, learning through a MARL approach can be faster, more robust and computationally more effective than centralized RL methods if sufficient coordination between the agents is provided. So far, several MARL algorithms are developed for cooperative systems and successfully applied to various problems [13, 14, 15, 16].

Although both the actor-critic RL and the MARL are being researched extensively, MARL methods that use actor-critic learning techniques have not been investigated thoroughly yet, especially for control applications. For that purpose, in this study, it is intended to combine these two major domains of RL to investigate the potential advantages of such an approach.

1-2 Research Objectives

The goal of this project is to incorporate the benefits of actor-critic methods into the MARL framework for the cooperative MAS. Imposing actor-critic to MARL, it is aimed to obtain smooth, continuous policies for the agents with similar or better performance than critic-only MARL or centralized RL. In this context, the performance is measured by 2 criteria: the initial convergence rate and the optimality of the converged policy at the end of the experiments.

The objectives to be accomplished through this research are as follows:

- The first objective is to develop a cooperative Multi-Agent Actor-Critic (MAAC) algorithm with suitable policy and critic updates following the Independent Learners (IL) approach and its variants. The IL approach will be used as foundations of the primal MAAC algorithm due to its simplicity.
• The second objective is to test and verify the developed algorithm for IL through simulations and experimentation, and to compare the performance of the algorithm to its modifications based on variants of IL. The performance will also be compared with the centralized Model Learning Actor-Critic (MLAC) algorithm [11, 17].

1-3 Outline

This study consist of five chapters. In Chapter 2, some preliminary knowledge on RL and actor-critic methods are explained. Chapter 3 describes the MAS concept, and the centralized learning algorithm (MLAC) that is used throughout the thesis. In addition these, this section consist of the proposed MAAC algorithms that are based on MLAC and the IL approaches.

Chapter 4 exhibits the learning problem that is used to test the algorithms, the results of the experiments, and the comparison and discussion of results. Finally, Chapter 5 concludes this study with drawn conclusions and comments for future studies.
Chapter 2

Preliminaries

This chapter presents preliminary knowledge about Reinforcement Learning (RL) and actor-critic RL. RL is a learning framework inspired by the way animals learn to deal with new situations. In RL the learning agent is not told what to do in a specific situation; it has to discover this on its own via trial-and-error. It therefore receives a numerical reward for every state transition, the better the transition the higher the reward, and tries to maximize the return over the long run. The trial-and-error nature together with the maximization of the return over the long run are important properties of RL. In Section 2-1 an introduction to RL containing the important aspects necessary for this thesis is given, after which in Section 2-2 the basics of the actor-critic techniques are presented.

2-1 Reinforcement Learning Framework

RL [3, 18] is a machine learning method that can be applied to solve optimization and control problems. It involves a distinct environment and an agent, that makes decisions by interacting with the environment. RL gets its name from reinforcements, also known as rewards, given to the agent while learning; and it adapts the agent’s policy as the agent collects rewards.

In this section an overview of RL will be briefly introduced. As an introductory section, this overview will only contain the information necessary for this thesis, so it will not be exhaustive.
2-1-1 RL in Markov Decision Process Framework

RL problems can be conveniently described as Markov Decision Process (MDP) with discrete time steps $k$ [3]. For the sake of generality, stochastic setting is used in this paper although the problem that will be encountered is highly deterministic as well as the algorithms used to solve the problem. Further information about MDP is extensively covered in [19].

An MDP is defined as a 4 tuple $\langle X, U, f, \rho \rangle$, where $X$ denotes the state space of the process, $U$ is the action space of the controller, $f$ is the state transition probability density function which maps the state changes with the control actions and, finally, $\rho$ indicates the reward function. State transition function $f : X \times U \times X \rightarrow [0, \infty)$ gives the probability density over $X$ for the next state $x_{k+1}$ after executing action $u_k$ in state $x_k$. Also, at the time instant $k$, the agent receives the reward $r_{k+1}$ according to the reward function $\rho(x_k, u_k) : X \times U \rightarrow \mathbb{R}$ where this reward function is assumed to be bounded. This function is explicitly represented as follows.

$$r_{k+1} = \rho(x_k, u_k)$$

Action $u$ is chosen according to policy $\pi : X \times U \rightarrow [0, \infty)$. A MDP has the Markov property, which means that transitions and rewards only depend on the current state-action pair and not on past state-action pairs nor on information excluded from $x$.

2-1-2 Returns, Value Functions and Optimality

In RL, the agent seeks an optimal policy that maximizes a function of cumulative aggregation of rewards collected. This cumulative function is called return, and unlike rewards, the return is a measure of long-time performance of the controller. If the RL task ends in a terminal
state, the problem becomes episodic, and as a result, the return will assume a finite horizon. On the other hand, the RL problem may not be defined over a terminal state. In that case, it is continuing and the return will have an infinite horizon. Most of the time, return is selected as either the discounted sum of rewards or the average of collected rewards for both finite and infinite horizon problems [20]. The differences between these two return modes are extensively covered in [21] including the cases with parametrically approximated value functions. In this section, only the discounted return case will be discussed as it is used in the entire thesis.

The discounted return setting [18], [20] is defined as the discounted sum of rewards starting from \( x_k \in X \) with the discount rate \( \gamma \). This function is denoted as:

\[
J^\pi = \mathbb{E}^\pi \left\{ \sum_{i=0}^{K} \gamma^i r_{i+1} \mid x_0 = x \right\}
\]

(2-1)

where \( \gamma \in [0, 1) \), \( K \) denotes the final time step at which the task terminates, and \( J^\pi_k \) is the return at time instant \( k \) under the policy \( \pi \). \( \mathbb{E}^\pi \{ \ldots \} \) indicates the expected value of its argument provided that the agent follows the policy \( \pi \). While episodic tasks have a finite \( K \), for continuing tasks (infinite horizon reward setting) \( K \to \infty \).

Throughout the learning process, the agent needs to evaluate the performance of a given policy. Thus, the return \( J^\pi_k \) should be estimated. As a result, value functions are introduced and there are two definitions exist: state-action value functions and state value functions. The state value function \( V^\pi(x) \) gives the expected return of following policy \( \pi \) depending on state \( x \).

\[
V^\pi(x) = \mathbb{E}^\pi \{ J^\pi \mid x_0 = x \} = \mathbb{E}^\pi \left\{ \sum_{i=0}^{K} \gamma^i r_{i+1} \mid x_0 = x \right\}
\]

(2-2)

On the contrary, the state-action value function \( Q^\pi(x, u) : \)

\[
Q^\pi(x, u) = \mathbb{E}^\pi \{ J^\pi \mid x_0 = x, u_0 = u \} = \mathbb{E}^\pi \left\{ \sum_{i=0}^{K} \gamma^i r_{i+1} \mid x_0 = x, u_0 = u \right\}
\]

(2-3)

is a function of \( u \), as well as \( x \). Therefore, in state-action value functions, first action after time instant \( k \) is now free to choose, instead of immediately following the policy \( \pi \) at that time. If this first action were to be chosen according to the policy \( \pi \), the equality \( V^\pi(x) = \mathbb{E}^\pi \{ Q^\pi(x, u) \} \) is obtained.

Both value functions can be organized in a recursive form called the Bellman equations [22].

\[
V^\pi(x) = \mathbb{E} \{ \rho(x, u, x') + \gamma V^\pi(x') \}
\]

(2-4a)

\[
Q^\pi(x, u) = \mathbb{E} \{ \rho(x, u, x') \} + \gamma Q^\pi(x', u')
\]

(2-4b)

where, \( x' \) is drawn from the probability distribution, \( f(x, u, \cdot) \) and \( u' \) is drawn from the policy, \( \pi(x', \cdot) \). As mentioned above, the agent tries to learn an optimal policy to maximize
the return defined in the problem. The optimality of these value functions is determined by Bellman optimality equations [22]. Any state value or state-action value function that satisfies corresponding Bellman optimality equations will yield to the optimality.

\[ V^*(x) = \max_u E \{ \rho(x, u, x') + \gamma V^*(x') \} \]  
\[ Q^*(x, u) = E \{ \rho(x, u, x') + \gamma \max_{u'} Q^*(x', u') \} \]

(2-5a)
(2-5b)

To find the (locally) optimal policy, \( \pi \) for RL problems, several model-based (mostly dynamic programming) and model-free algorithms have been developed. In real-world applications, typically model of the environment is not available for the agent, thus model-based approaches cannot be directly applied. At this point, Temporal Difference (TD) learning techniques provides a decent solution as they do not require a model of the environment to be available to the agent. Therefore, the next part of the section is devoted to introducing the TD concept.

### 2-1-3 Temporal Difference Concept

TD methods can be viewed as an iterative approximation of Bellman equations. With TD methods, the agent processes the immediate rewards it receives after each time step, thereby learning from each action [23]. The most basic TD update rule for the state value function is in the form for the discounted return:

\[ V(x_k) \leftarrow V(x_k) + \alpha_k \left[ r_{k+1} + \gamma V(x_{k+1}) - V(x_k) \right] \]

(2-6)

where \( \alpha_k \) is the learning rate for time instant \( k \) and the term in brackets in equation, \( \delta_k \) is called the TD-error. The learning rate determines how much TD-error dictates on the the new prediction of the value function. If the exact value functions are used, the value function will converge to optimal value function with proper conditions on the learning rate [3]. These conditions can be given as

\[ \sum_{k=1}^{\infty} \alpha_k = \infty \]  
\[ \sum_{k=1}^{\infty} \alpha_k^2 < \infty \]

(2-7a)
(2-7b)

Obviously, the learning rate \( \alpha_k \) should decay over time to satisfy the conditions above and constant learning rates will not yield to the convergence. Yet, there are vast amount of real applications that are not obeying these.

The simple update rule (2-6) ensures updating solely the previous state, so the reward that is learned gives credit to only that state. Nevertheless, each early state makes its contribution
to acquire this reward, therefore, states visited in the past should be rewarded too. Logically, more recent states should be rewarded more compared to states visited in the further past. To handle this credit assignment problem, eligibility traces [3] can be incorporated to TD structure. In the next part of the section, this approach will be discussed.

2-1-4 Eligibility Traces

A major improvement over TD methods introduced in previous section is the application of eligibility traces [23], [24]. This mechanism tries to cope with delayed rewards by giving a mark a to the visited state (or state-action pair) that traces the eligibility of the state (state-action pair) to be rewarded. As the time passes, this trace given to the state (state-action pair) decays to give credit to more recent states (state-action pairs). Whenever a reward is earned, these states (state-action pairs) visited will get an amount of credit depending on the corresponding value of their trace.

The eligibility trace for state $x$ is denoted as $e(x) \in \mathbb{R}^+$. The factor $\gamma \lambda$ is used to decay the eligibility of the trace where the $\lambda$ is the trace decay. Depending on the update of the eligibility trace of the visited state, there are two widely known options: **accumulating traces** and **replacing traces**. In accumulative case, after a state is visited, the corresponding trace is incremented by 1. The mathematical formulation for accumulative traces in discounted reward setting is as follows.

$$
e(x) \leftarrow \begin{cases} 
\gamma \lambda e(x) & \text{if } x \neq x_k \\
\gamma \lambda e(x) + 1 & \text{if } x = x_k
\end{cases} \quad (2-8)
$$

for all nonterminal states.

On the other hand, if the replacing traces are used, the trace of the visited state is directly replaced by 1.

$$
e(x) \leftarrow \begin{cases} 
\gamma \lambda e(x) & \text{if } x \neq x_k \\
1 & \text{if } x = x_k
\end{cases} \quad (2-9)
$$

The difference between accumulating and replacing traces is presented in Figure 2-2 as a simple graph. Note that, the method of accumulating traces is known to suffer from convergence problems, therefore replacing traces is the most commonly used method in practice [24].

The update rule given in (2-6) can be further extend by using the eligibility traces:

$$V(x) \leftarrow V(x) + \alpha_k \delta_v e(x) \quad \forall x \in X \quad (2-10)$$

The TD method that uses eligibility traces as in (2-10) is called TD($\lambda$), where TD(0) is the one step TD method mentioned in (2-6). TD(1) is also refereed as Monte Carlo (MC) learning [3].

Master of Science Thesis

Yagiz Efe Bayız
2-2 Actor-Critic Methods

So far exact representations of value functions or policies have been introduced in previous section. However, these functions should be stored for every possible state-action couple to ensure such exactness and this is mostly infeasible or even impossible for the cases with continuous state or action spaces. Therefore, function approximators are commonly used to approximate the value functions and/or the policy.

Over the course of time, a number of RL algorithms, has been developed and they are categorized in 3 groups depending on how these functions are allocated in the methods: critic-only, actor-only and actor-critic methods [7]. In this classification, actor stands for the policy followed and the critic represents the estimated value function which criticizes the policy.

Critic-Only Methods Critic-only methods, such as Q-Learning and SARSA, learn value functions and determine a policy exclusively based on the estimated value function. As mentioned above, if the action and/or state spaces are continuous, the value function is approximated by using function approximators. The most compelling property of the critic-only methods is that they result to low-variance estimate of the expected returns which speeds up the learning [20]. The actions are mostly selected by using greedy action selection strategy that is simply selecting the action which maximizes the value for that state. Therefore, an optimization procedure is required each time an action is selected. To avoid the computational complexities caused by the optimization, in critic-only methods, mostly discretized action spaces are used. Therefore, the advantages brought by using a continuous action space are not available to these methods.

Actor-Only Methods Actor-only methods on the other hand, use parametrized policies to directly estimate the gradient of the return, with respect to the parameters of the actor, and update these parameters in a direction of improvement without using value functions. In these methods, the policy is mostly parametrized using a function approximator with its own parameters. The commonly used approach to update the policy parameter is to use
so called *policy gradient* which is the the differentiated expected return with respect to the policy parameter. Although, actor-only methods work well in continuous action spaces, they suffer from high-variance of the estimated policy gradient\[25, 26\]. Learning a value function reduces the variance of the gradient estimate which is essential for rapid learning.

**Actor-Critic Methods** The actor-critic structure is first introduced by Barto et al. in 1983 [8] and have been thoroughly investigated since then. Actor-critic algorithms inherit the advantages of both critic-only and actor-only methods as they bridge the gap between these classes. The presence of the parametrized value function provides a decent evaluation of the policy followed while the having a parametrized policy allows the use of continuous action space.

The focus of this section is determined as the policy gradient actor-critic methods with temporal difference critic update. As in previous section, the stochastic MDPs in discrete time setting are considered as the framework of the methods mentioned in here.

**2-2-1 Policy Gradient**

The aim of the RL methods is to find a policy that maximizes the return, $J$. The most straightforward method to iteratively approach to optimal policy is to use gradient ascent method (gradient descent for maximization) which is a first order technique. Let $\pi_{\nu}$ is the approximated policy with parameter vector $\nu \in \mathbb{R}^p$. Then, the policy parameter is updated as follows.

$$
\nu \leftarrow \nu + \alpha_a \nabla_{\nu} J 
$$

where $\alpha_a \in [0, 1)$ denotes the learning rate of the actor and $\nabla_{\nu} J$ is the policy gradient at current time step which is $k$. One advantage of the policy gradient methods is that they provide decent convergence properties, and the conditions for the guaranteed convergence are well-established.

The core problem of the policy gradient approaches is to estimate the $\nabla_{\nu} J$. There are several techniques to estimate the policy gradient introduced such as finite difference gradient estimators [27] and likelihood ratio methods (e.g REINFORCE) [28, 29]. These methods do not rely on any value functions, thus, they are actor-only methods. As mentioned above, actor only methods suffer from high-variance of policy gradient estimation thus yield to slower learning.
2-2-2 Actor-Critic Structure

The actor-critic methods form a special class of the policy gradient approaches as they use an approximate value function and an approximate policy to solve the MDP. Function approximation is used to generalize the information gained from a small group of visited states to a much larger set of states, of which most of them never have been visited. In this section, the functions are approximated by parameterizing with respect to a parameter vector.

The critic part of the actor-critic algorithms consists of the approximated value function, either \( V_\phi(x) \) or \( Q_\phi(x, u) \) which are parametrized with respect to \( \phi \in \mathbb{R}^q \). Similarly, the policy is parametrized with respect to parameter vector \( \nu \in \mathbb{R}^p \) as \( \pi_\nu(x, u) \). Note that, \( \pi_\nu(x, u) \) is the stochastic policy which represents the probability density of choosing action \( u \) depending on state \( x \). The structure of the actor-critic update of discounted return case is presented in the following part.

The critic update can be performed with any known policy evaluation method such as TD or Least-Squares Temporal Difference (LSTD) approaches [30]. Temporal difference for the critic can be approximated as follows for the state value function \( V_\phi(x) \):

\[
\delta_V \leftarrow r_{k+1} + \gamma V_\phi(x_{k+1}) - V_\phi(x_k)
\]  

(2-12)

Then, the critic update in discounted reward setting can be performed with TD(0) update which is the most basic approach. In [3], the update rule for \( \phi \) is derived by using the gradient descent of the mean square between the \( V_\phi(x) \) and the actual value function \( V_\pi(x) \) with the TD(0) update.

\[
\phi \leftarrow \phi + \alpha \cdot \delta_V \cdot \nabla_\phi V_\phi(x_k)
\]

(2-13)
where \( \alpha_c \in [0, 1) \) is the learning rate for critic. To improve the learning performance, a common extension to TD(0) is to introduce eligibility traces with trace decay rate \( \lambda \). After the application of the eligibility traces, the update rule for TD(\( \lambda \)) evolves to:

\[
\begin{align*}
z & \leftarrow \lambda \gamma z + \nabla \phi V_{\phi}(x_k) \\
\phi & \leftarrow \phi + \alpha_c \delta V z
\end{align*}
\]

(2-14a)

(2-14b)

where, \( z \in \mathbb{R}^q \) denotes the eligibility traces. As in classical RL, the eligibility traces provide a better assignment of received rewards to states (or state-action pairs) visited. The parameter vector is not only updated under the influence of the last time step but it also includes the effects of the value function approximations of all the visited states where the contribution of the visited states decays with the factor of \( \gamma \lambda \). The complete set of update rules for the discounted return case with eligibility traces becomes:

\[
\begin{align*}
\delta V & \leftarrow r_{k+1} + \gamma V_{\phi}(x_{k+1}) - V_{\phi}(x_k) \\
z & \leftarrow \lambda \gamma z + \nabla \phi V_{\phi}(x_k) \\
\phi & \leftarrow \phi + \alpha_c \delta V z \\
v & \leftarrow v + \alpha_a \nabla v J
\end{align*}
\]

(2-15a)

(2-15b)

(2-15c)

(2-15d)

The critic update can be employed by vast amount of approaches other than TD(\( \gamma \)) and some of these methods can outperform TD(\( \gamma \)). However, the options are limited when it comes to the actor and all rely on the estimation of the policy gradient, \( \nabla \nu J \). The major challenge in actor-critic approaches is to find a suitable estimate of \( \nabla \nu J \) based on the critic. To relate the critic and the actor with each other, policy gradient theorem is introduced as a result of 2 simultaneous, independent studies [7, 25] which prove san unbiased estimate of the policy gradient can be determined from the critic. However, this theorem will not be discussed here since it is out of the scope of this thesis. Further information policy gradient theorem can be found in [7, 25, 20].

In this section, an introduction to general actor-critic structure has been made and no specific algorithms has been discussed. The Model Learning Actor-Critic (MLAC) algorithm that has been used as a base to the algorithms proposed in this thesis is presented in Section 3-2.

### 2-3 Summary

In this chapter, the fundamental knowledge on reinforcement learning and actor-critic algorithms have been introduced. This information has been either directly used to develop the algorithm of this work or required to understand the methodology discussed in this thesis. The chapter starts with the basic concepts of reinforcement learning such as the value functions, optimality, temporal difference and eligibility traces. In the second part of the chapter, actor-critic methods have been covered and the background on these methods are provided.
In this chapter, cooperative Multi-Agent Systems (MAS) is discussed and proposed extension of the actor-critic techniques to such systems is presented. A MAS is a collection of interacting agents that share a common environment which they perceive through sensors, and upon which they act through actuators [12]. MAS have been categorized regarding the level of cooperativeness of the agents in the same environment [31]. One of the major groups in MAS is the case of fully cooperative agents which collectively aim to achieve a common goal. These systems occur very frequently in practical applications. Especially in control problems, the controller tries to manipulate a given system to achieve a certain goal, so if a control problem has multiple agents, these agents are cooperative in most of the times. For that purpose, this project targets cooperative multi-agent systems exclusively.

Before going into the multi-agent algorithms, MAS concept will be addressed together with the cooperative Multi-Agent Reinforcement Learning (MARL). Then, centralized learning will be mentioned briefly and the Model Learning Actor-Critic (MLAC) algorithm will be introduced in its original form. Later on, the MARL approaches that have been used in proposed Multi-Agent Actor-Critic (MAAC) algorithms are explained, and these algorithms are presented.

3-1 MAS and cooperative MARL

A MAS can be defined as a collection of several independent entities sharing a common environment in which the agents try to behave according to their individual interests and goals
For some cases, MAS may arise as the most natural way of looking at the system, or may provide an alternative perspective on systems that are originally regarded as centralized. For instance, in robotic teams, the control authority is naturally distributed among the robots thus it is convenient to analyze such systems as MAS.

In MAS, the environment is usually complex, so a priori design for a good agent behavior is not practical. Therefore, learning is mostly required in such systems. In this context, Reinforcement Learning (RL) provides a good basis to multi-agent learning due to its simplicity and convergence properties. Thus, RL methods are extended to work in MAS framework that is called MARL. Potentially, MARL configuration can brought several advantages due to its decentralized structure since it enables us to distribute a globally complex problem into sub-problems that are easier to deal with. One major advantage is that the action space to be searched for each agent can be arranged such that only the related action spaces are included. This indicates a computational relaxation for agents due to the parallel computation.

In addition to computational advantages, this might yield to a faster learning as the size of the action space to be searched for each agent is reduced. Additionally, possibility to share experience between the agents, and the robustness properties of MARL makes it an active field of study.

Depending on the level of cooperativeness of the agents in the same environment, MARL methods can be grouped in three categories: cooperative tasks, competitive tasks and general-sum tasks. In this thesis, cooperative tasks will be taken into account as mentioned in the introduction of this chapter. This group of MARL deals with the tasks where all agents have a common goal and do not compete with each other in any sense.

Literature in cooperative MARL problems can be grouped as model-based and model-free approaches, similarly as in the single-agent counterpart. If the agents are aware of the common state transition function and the reward function, the learning method can be based on the predictions of agents about the next state and the reward that will be obtained. In that case, solution of Multi-Agent Markov Decision Process (MMDP) will be called model-based. In contrast to model based approaches, in model-free RL no information about the $f$ and $\rho$ is supplied to the agents. In this thesis model-free based methods are of interest, hence, learning techniques for this case will be discussed. Several approaches have been proposed to solve model-free MARL consisting the Independent Learners (IL) based methods [33], Distributed Value Functions (DVF) [34] and Coordination Graph Methods [35]. The scope of this study is determined as IL approach and two of its variants: optimistic learners and lenient learners.

### 3-2 Centralized Learning

A cooperative multi-agent learning problem can be considered as if it is a single-agent system with multiple inputs and outputs. This allows us to use single-agent RL techniques which
are well-researched and have good convergence properties. For this project, centralized RL is used as a basis for the multi-agent learning as well as a method to compare the performance of the proposed MAAC algorithms.

In this research, centralized learning is performed by MLAC [11, 17]. This algorithm offers good convergence properties with fast learning with a deterministic policy.

Model Learning Actor-Critic

MLAC is developed to employ an efficient policy update which increases the learning speed considerably compared to other actor-critic methods. As an improvement, this algorithm learns a process model in addition to the actor and critic. Different from other algorithms, actor update in this algorithm is done using a policy gradient which is calculated using a local gradient of the critic and a local gradient of the learned process model.

MLAC algorithm has the same structure that is introduced in 2-2, and it can be presented as follows. Let $V_\phi(x)$ is the approximation of value function where $x$ is the normalized state variable vector and $\pi_\nu(x)$ is the parametrized deterministic policy. The process model $\hat{f}_\zeta(x, u)$ is used to approximate the next state given the current state and the input and it is learned through the steepest ascent method. Then, the functions are updated with the following update rules:

$$
\delta V \leftarrow r_{k+1} + \gamma V_\phi(x_{k+1}) - V_\phi(x_k)
$$

$$
\phi \leftarrow \phi + \alpha_c \delta V z
$$

$$
z \leftarrow \lambda \gamma z + \nabla_\phi V_\phi(x_k)
$$

$$
\nu \leftarrow \nu + \alpha_a \nabla_x V_\phi(x_{k+1}) \cdot \nabla_u \hat{f}_\zeta(x_k, u_k) \cdot \nabla_v \pi_\nu(x_k)
$$

$$
\zeta \leftarrow \zeta + \alpha_p (x_{k+1} - \hat{f}_\zeta(x_k, u_k)) \cdot \nabla_\zeta \hat{f}_\zeta(x_k, u_k)
$$

where $z$ indicates the eligibility traces, and $\alpha_a, \alpha_c, \alpha_p$ are the learning rates for the actor, critic and the process model. The learning problem that is in scope in this research has a saturating input. Therefore, the process gradient $\nabla_{u^j} \hat{f}_\zeta(x_k, u_k) = 0$ whenever input $u^j$ is saturated where $u^j$ is the $j^{th}$ element in $u$. In other words, inputs stay constant outside their boundaries, the partial derivative with respect to this input should selected as 0 whenever the corresponding input saturates. The pseudo-code that summarizes the MLAC algorithm is given as Algorithm 1.

So far the centralized approaches have been discussed. The next part of this chapter is devoted to MARL methods. As in single-agent RL, MARL also needs a the framework to fit in. Thus, MMDP structure will be explained in the the next section.
**Algorithm 1** Model Learning Actor-Critic Algorithm

**Input:** $\gamma$, $\lambda$, and learning rates $\alpha_a$, $\alpha_c$ and $\alpha_p$

1: Initialize function approximators
2: $z_0 = 0$
3: Initialize $x_0 = 0$
4: Apply input $u_0$
5: $k \leftarrow 0$
6: loop
7: Measure $x_{k+1}$ and $r_{k+1}$
8: % Select action for the next state
9: $u_{k+1} \leftarrow \pi_\nu(x_{k+1})$
10: % Update critic
11: $\delta V \leftarrow r_{k+1} + \gamma V_\phi(x_{k+1}) - V_\phi(x_k)$
12: $\phi \leftarrow \phi + \alpha_c \delta V z$
13: $z \leftarrow \lambda \gamma z + \nabla V_\phi(x_k)$
14: % Update actor
15: $\nu \leftarrow \nu + \alpha_a \nabla_x V_\phi(x_{k+1}) \cdot \nabla_u \hat{f}_\zeta(x_k, u_k) \cdot \nabla \nu \pi_\nu(x_k)$
16: % Update process model
17: $\zeta \leftarrow \zeta + \alpha_p (x_{k+1} - \hat{f}_\zeta(x_k, u_k)) \cdot \nabla \zeta \hat{f}_\zeta(x_k, u_k)$
18: Choose exploration $\Delta u_{k+1} \sim \mathcal{N}(0, \sigma^2)$
19: Apply $u_{k+1} + \Delta u_{k+1}$
20: $k \leftarrow k + 1$
21: end loop

### 3-3 Multi-Agent Markov Decision Process

Boutilier has introduced MMDP which is a straightforward extension of Markov Decision Process (MDP)s to cooperative multi-agent systems [36]. In this framework, it is assumed that individual actions are applied to the process by the agents and a common reward function is defined for all agents. In discrete time with the deterministic setting MMDP can be defined as a tuple $(\mathcal{A}, \mathcal{X}, \{\mathcal{U}^1, \mathcal{U}^2, \ldots, \mathcal{U}^a\}, f, \rho)$, where $\mathcal{A}$ is the set of agents having a number of agents, $\mathcal{X}$ denotes the joint state space of all the agents, $\mathcal{U}^j$ is the action space of the agent $j$, $f : \mathcal{X} \times \mathcal{U}^1 \times \mathcal{U}^2 \times \cdots \times \mathcal{U}^a \rightarrow \mathcal{X}$ is the joint state transition function which maps the state changes with the control actions and, finally, $\rho : \mathcal{X} \times \mathcal{U}^1 \times \mathcal{U}^2 \times \cdots \times \mathcal{U}^a \rightarrow \mathbb{R}$ indicates the reward function shared among all agents. The individual action of the agent $j$ at time is denoted as as and $u^j$ whereas the joint action is simply $u$. Keeping the essence of MMDP the same, some modifications are also used while using this framework. On of the modified framework uses local reward functions associated to individual agents [37, 14]
It should be noted that, MMDP framework can be extended jointly observable Decentralized Markov Decision Process (Dec-MDP) or even to Decentralized Partially Observable Markov Decision Process (Dec-POMDP) [38]. However, for the purpose of this project, MMDP provides a sufficient base as the project involves control of fully observable agents.

3-4 Independent Learners

As discussed in Section 3-1, multi-agent approaches bring several advantages to a learning algorithm. The most straightforward method to extend a learning technique to multi-agent systems is to use IL approach [33]. In this case, the actions and rewards of other agents are not taken into account by each agent, i.e, they learn their policy independently. An independent value function for each agent depending on their own actions is stored and updated separately, disregarding any means of coordination with other agents.

In that sense, the agents are independent reinforcement learners and any of the single agent methods can be applied on these agents. However, as indicated by [39], from single agent point of view the environment is now no longer stationary or Markovian, since the agents are simply ignoring the presence of each other. In other words, an agent is not aware of actions of the other agents. Assume that the result of the agent’s individual action is changed, i.e its policy results to a different state than the previous one for a given state. In that case, the agent cannot distinguish if this change is a result of a teammate changing its policy, or due to the explorative actions of some agents. This means that, from the agent’s point of view, the process is not Markovian and not stationary anymore. Hence, due to violation of Markov and stationary assumptions, the convergence is not guaranteed for IL’s as it does for most single-agent learning algorithms [3, 7]. Yet, this approach has been successfully applied to many control problems in simulation or real-time environments including the distributed micro-manipulation of air-jet systems [15] and multi-agent control of walking robots [14].
Although the problem is non-Markovian and non-stationary as mentioned above, the single agent methods may still be applied to individual learners. For instance, Q-Learning has been extended to MAS with an IL approach by Tan in [31] as Decentralized Q-Learning. Additionally, this idea can be further applied to approximate RL, or even eligibility traces can be implemented [14]. An example to such update rule to policy can be given as:

$$\phi^j \leftarrow \phi^j + \alpha^j_c \delta^j z^j$$  \hspace{1cm} (3-1)

where $\phi^j$ is the parameter vector of the independent value function for the $j^{th}$ agent, $\alpha^j_c$ is the learning rate of the corresponding value function, $\delta^j$ is the temporal difference of the agent $j$, and $z^j$ indicates the eligibility traces for agent $j$.

The policies of the agents are also independent in IL since the agents treated as single agents. Thus, the actor can be implemented to IL methods should be straightforward as well. This indicates that the MLAC method can also be extended to multi-agent framework.

At this point, it should be noted that MLAC is not the optimal algorithm to extend to MAS. Since the V-functions are used in the critic, approximated value functions only depend on state variables, so they do not benefit from the direct reduction of the number of variables they depend on due to the independent actions instead of joint ones. In other words, the use of IL approach would be more computationally efficient for an algorithm that uses Q-values such as the algorithms described in [7, 40]. Additionally, the estimation of a process model makes MLAC harder to extend to IL method. However, the aim of this project is to prove that actor-critic techniques can be used in MAS framework in practical terms and the performance of the learning is not the primary goal. Additionally, the author’s previous experience on this algorithm makes it more appealing. Thus, despite the shortcomings of the MLAC algorithm, it will be used as the basis of the algorithms that will be used for multi-agent learning.

**Independent Multi-Agent MLAC**

The Independent Multi-Agent MLAC is the direct extension of MLAC to multi-agent problems with IL approach. While extending the MLAC algorithm, either global or local rewards can be used. Although the use of global rewards imposes an information flow between the agents, it also indicates that full state information should be used in independent value functions and policies. The reason for that is the global reward is generally a function of most state variable that might not be relevant for an individual agent to accomplish the joint task.

On the other hand, local rewards provide further flexibility to use only the state information that is relevant to the learning for each agent. That will reduce the state space of agents to a certain extent and speed up the learning. For instance, in a team of mobile robots, at a given time, the position and velocity of robots that are far away from the considered robot might...
not be interesting for it. In such cases, the learning agents can consider only the relevant state components and thus further decrease the size of the problem [41]. In this, project rewards are assigned to the agents locally and the global performance of the agents are measured by the cumulative of these local rewards. Then, direct extension of the MLAC algorithm to multi-agent systems has the following update structure for agent $j$:

$$
\delta^j \leftarrow r^j_{k+1} + \gamma^j V^j_{\phi}(x^j_{k+1}) - V^j_{\phi}(x^j_k)
$$

$$
\Phi^j \leftarrow \Phi^j + \alpha^j \delta^j V^j_{\phi}(x^j_k)
$$

$$
z^j \leftarrow \lambda^j \gamma^j z^j + \nabla_{\phi^j} V^j_{\phi}(x^j_k)
$$

$$
v^j \leftarrow v^j + \alpha^j \nabla_{x^j} V^j_{\phi}(x^j_{k+1}) \cdot \nabla_{u^j} \hat{f}^j(x^j_{k+1}, u^j_k) \cdot \nabla_{\nu^j} \pi^j_{\nu^j}(x^j_k)
$$

where superscript $j$ indicates the local parameters, functions and variables associated with agent $j$. An important modification here is the use of the local process model, $\hat{f}^j(x^j_k, u^j_k)$. The local transition function for agent $j$ is not stationary and changes with the policies of other agents, so the approximation of the function $\hat{f}^j(x^j_k, u^j_k)$ becomes infeasible. As a result, the partial derivative $\nabla_{u^j} \hat{f}^j(x^j_k, u^j_k)$ cannot be estimated by simple iterative update of $\hat{f}^j(x^j_k, u^j_k)$. Thus, a global approximated transition function, $\hat{f}^{\zeta}(x_k, u_k)$, is used instead of a local one, and agent specific $\nabla_{u^j} \hat{f}^j(x^j_k, u^j_k)$ is drawn from $\nabla_{u^j} \hat{f}^{\zeta}(x_k, u_k)$. The introduction of global process model also indicates that knowledge is transferred between the agents in an indirect manner. Note that, as in classic MLAC, the transition function, $\hat{f}^{\zeta}(x_k, u_k)$, is trained by using the recursive steepest ascent method.

$$
\zeta \leftarrow \zeta + \alpha_p (x_{k+1} - \hat{f}^{\zeta}(x_k, u_k)) \cdot \nabla_{\zeta} \hat{f}^{\zeta}(x_k, u_k)
$$

(3-2)

The experiments with this method indicate that it is not suitable for the task which is used to test the performance of the algorithms in this research as the agents constantly failed to converge to a policy. The reason for that is IL approach does not provide sufficient robustness to compensate the problems that is caused by the lack of coordination between the agents that are thoroughly examined in [42]. However, further improvements on IL, which are proposed to solve some of the problems, can be applied to this algorithm, and they offer a more robust learning. Two of these improvements will be discussed in the next sections.

### 3-5 Optimistic Learners

In IL approach, the learning problem for each agent is not stationary. Hence, each agent has to deal with a moving target learning problem: the best policy depends on the other agents’ policies. When no coordination exists between the other agents as in IL, convergence problem may occur. To handle the coordination problem, several methods can be found in literature.
As a modification of IL, Lauer and Riedmiller [43] proposed the optimistic learners targeting the shadowed equilibrium problem.

Shadowed equilibrium is a commonly encountered coordination problem in Independent Learners [44]. It is the convergence of independent agents to a sub-optimal equilibrium when unilateral deviation form that sub-optimal policy appears to be not preferable, even though, a simultaneous change of policies of some or all agents may yield to a potentially better policy.

Optimistic learners optimistically assume that all other agents will act to strictly maximize their reward. Thus, value functions are only updated if the Temporal Difference (TD) error is positive, thereby ignoring mistakes by other agents that lead to a lower expected return. This idea can be extended to systems with multiple states and function approximation [14].

The value function update for an approximate value function is presented as follows:

$$\delta_j^V \leftarrow r_j^k + \gamma_j V_j^k(x_j^{k+1}) - V_j^k(x_j^k)$$

$$\phi_j^j \leftarrow \begin{cases} 
\phi_j^j + \alpha_j \delta_j^V z_j^j & \text{if } \delta_j^V > 0 \\
\phi_j^j & \text{otherwise} 
\end{cases}$$ (3-3)

In [43], Lauer and Riedmiller give an additional condition that makes agents choose the same optimal equilibrium as well as updating value functions optimistically. This additional condition provides a convergence criterion for the Q-learning with IL just as single-agent Q-learning. Although there is no direct proof of convergence for multiple stage games or the algorithms with V-functions, the idea of optimistic learners will be embraced and combined with MLAC, and it will serve as a basis to Optimistic Multi-Agent MLAC.

Optimistic Multi-Agent MLAC

The idea in this algorithm is to apply optimistic learners approach to Independent Multi-Agent MLAC so that it will become more robust to lack of coordination between the agents. The update structure of the Optimistic Multi-Agent MLAC for agent $j$ is given in the following update rules.

$$\delta_j^V \leftarrow r_j^k + \gamma_j V_j^k(x_j^{k+1}) - V_j^k(x_j^k)$$

$$\phi_j^j \leftarrow \begin{cases} 
\phi_j^j + \alpha_j \delta_j^V z_j^j & \text{if } \delta_j^V > 0 \\
\phi_j^j & \text{otherwise} 
\end{cases}$$

$$z_j^j \leftarrow \lambda_j \gamma_j z_j^j + \nabla_{\phi_j} V_j^k(x_j^k)$$

$$\nu_j^j \leftarrow \nu_j^k + \alpha_j \nabla_{\nu_j} V_j^k(x_j^{k+1}) \cdot \nabla_{u_j} f_j^k(x_k, u_k) \cdot \nabla_{\nu_j} \pi_j^k(x_j^k)$$

The only modification is in the critic update as the value function parameter vector $\phi_j^j$ is updated optimistically. Note that, the eligibility traces are updated whether the TD error is positive or negative. Because, eligibility traces are associated with the history of the states.
visited no matter how the critic is updated. In other words, the complete history of the states is responsible for either improvement or decay of the current performance of the agents. Thus, the eligibility traces should be updated continuously. Similarly to the Independent Multi-Agent MLAC the global process model is used to draw $\nabla \hat{f}(x_k, u_k)$ for agent $j$ and this global process model is updated by the steepest ascent method.

$$\zeta \leftarrow \zeta + \alpha_p(x_{k+1} - \hat{f}(x_k, u_k)) \cdot \nabla \hat{f}(x_k, u_k)$$  \hspace{1cm} (3-4)

Algorithm 2 gives the pseudo-code for the optimistic Multi-Agent MLAC algorithm.

As the problem at hand becomes more stochastic, fluctuations of value functions occur due to stochastic state transitions and stochastic rewards. Such fluctuations, cannot be distinguished from fluctuations due to the changing policies of the other agents. Thus, using the optimistic assumption in stochastic problems leads to an overestimation of the total expected reward, and results to poor convergence properties contrary to the deterministic problems.

### 3-6 Lenient Learners

One approach to solve the overestimation problem of optimistic learners in stochastic domains is to adjust the degree of optimism of the learners with time. In that domain, lenient learners [45] idea is proposed for the stateless problems. Lenient learners start learning optimistically since being optimistic may be useful at early stages of learning to identify promising actions. However, as optimistic learners may lead to overestimation of values, at later stages of learning, achieving accurate estimation of values becomes more important. Hence, degree of optimism will drop for each agent after selecting an action. The lenient learners idea is further extended for multi-state problems with approximate value functions and successfully applied to robotic applications in [14]. The update rule for agent $j$ in that case is is given as follows:

$$\eta \sim U(0, 1)$$

$$\phi' \left\{ \begin{array}{ll}
\phi' + \alpha \delta^j \zeta^j & \text{if } \delta^j_Q > 0 \text{ or } \eta > \nu^j \\
\phi^j & \text{else}
\end{array} \right.$$  \hspace{1cm} (3-5)

where $\eta \in [0, 1]$ is a uniformly distributed random variable, and $\nu^j$ is the lenience defined for agent $j$. **Reminder:** explain lenience parameters. Then, the lenience can be updated as follows;

$$\nu^j \leftarrow e^{\kappa^j T^j_l} - 1$$

$$T^j_l \leftarrow \xi^j T^j_l$$

$\kappa^j$ being the lenience parameter and $T^j_l$ is the lenience temperature. This temperature is discounted with factor $\xi^j \in [0, 1)$ each time that a state is visited. Although, lenient learners
Algorithm 2 Optimistic Multi-Agent Model Learning Actor-Critic Algorithm

**Input:** $\gamma^j$, $\lambda^j$, and learning rates $\alpha^j_a$, $\alpha^j_c$ and $\alpha^j_p$, $\forall j$

1: Initialize function approximators
2: $z_j^0 = 0$, $\forall j$
3: Initialize $x_0 = 0$ and $x_j^0 = 0$, $\forall j$
4: Apply input $u_0$
5: $k \leftarrow 0$
6: loop
7: Measure $x_{k+1}$ and $r_{k+1}^j$, $\forall j$
8: for all $j \in A$
9: Draw $x_{k+1}^j$ from $x_k + 1$
10: % Select action of agent $j$ for the next state
11: $u_{k+1}^j \leftarrow \pi_j^\nu(x_{k+1}^j)$
12: % Update critic for agent $j$
13: $\delta_V^j \leftarrow r_{k+1}^j + \gamma^j V_\phi^j(x_{k+1}^j) - V_\phi^j(x_k^j)$
14: if $\delta_V^j < 0$ then
15: $\phi^j \leftarrow \phi^j + \alpha_c \delta_V^j z^j$
16: end if
17: $z^j \leftarrow \lambda^j \gamma^j z^j + \nabla_\phi^j V_\phi^j(x_k^j)$
18: % Update actor for agent $j$
19: Draw $\nabla_u \hat{f}_j^\nu(x_k, u_k)$ from $\nabla_u \hat{f}_\zeta(x_k, u_k)$
20: $\nu^j \leftarrow \nu^j + \alpha_a \nabla_x^j V_\phi^j(x_{k+1}^j) \cdot \nabla_u \hat{f}_j^\nu(x_k, u_k) \cdot \nabla_\nu^j \pi_\nu^j(x_k^j)$
21: end for
22: % Update global process model
23: $\zeta \leftarrow \zeta + \alpha_p(x_{k+1} - \hat{f}_\zeta(x_k, u_k)) \cdot \nabla_\zeta \hat{f}_\zeta(x_k, u_k)$
24: Choose exploration $\Delta u_{k+1} \sim \mathcal{N}(0, \sigma^2)$
25: Apply $u_{k+1} = u_{k+1} + \Delta u_{k+1}$
26: $k \leftarrow k + 1$
27: end loop

provide a solution to shadowed equilibrium problem while avoiding the overestimation of state-action values, parameter tuning can be hard in this method.

**Lenient Multi-Agent MLAC**

As in optimistic learners, lenient learners approach can be extended to Multi-Agent MLAC in a direct manner. Similarly, the Lenient Multi-Agent MLAC method has the following update...
rules:

\[
\delta_{j}\phi_{j} \leftarrow r_{j}^{k+1} + \gamma\phi_{j}(x_{k}^{j}) - \phi_{j}(x_{k}^{j}) + \gamma V_{\phi_{j}}(x_{k}^{j}) - V_{\phi_{j}}(x_{k}^{j}) + \gamma V_{\phi_{j}}(x_{k}^{j}) - \phi_{j}(x_{k}^{j})
\]

\[
\phi_{j} \left\{ \begin{array}{ll}
\phi_{j} + \alpha_{j}\delta_{j} \cdot z_{j} & \text{if } \delta_{j} > 0 \text{ or } \eta > \nu_{j} \\
\phi_{j} & \text{otherwise}
\end{array} \right.
\]

\[
z_{j} \leftarrow \lambda_{j}^{j} z_{j} + \nabla \phi_{j}(x_{k}^{j})
\]

\[
\nu_{j} \leftarrow \nu_{j} + \alpha_{j} \nabla_{x_{j}} \phi_{j}(x_{k}^{j}) \cdot \nabla u_{j} f_{j}(x_{k}, u_{j}) \cdot \nabla \nu_{j} \pi_{j}(x_{k})
\]

\[
l_{j} \leftarrow e^{\kappa_{j}T_{j}} - 1
\]

where \(\eta \in [0, 1]\) is a uniformly distributed random variable, \(l_{j}\) is the lenience defined for agent \(j\), \(\kappa_{j}\) is the the lenience parameter and \(T_{j}\) is the lenience temperature that is initialized from 1 and factored with \(\xi_{j} \in (0, 1)\). As in previous algorithms, global process model that is used in actor update is performed by:

\[
\zeta \leftarrow \zeta + \alpha_{p}(x_{k+1} - \hat{f}_{j}(x_{k}, u_{j})) \cdot \nabla \zeta \hat{f}_{j}(x_{k}, u_{j})
\]

(3-6)

Lenient Multi-Agent MLAC may possibly yield to better performance than Optimistic Multi-Agent MLAC since it brings a solution to the overestimation of the value functions that is encountered with that algorithm. This is especially important to MLAC algorithms since it uses the approximate value functions to update the policies. Thus, better estimates of the policies result to superior performance of the agents.

### 3-7 Summary

In this chapter, actor-critic methods are implemented to cooperative MARL framework. At first, concepts of multi-agent approaches have been discussed as well as the potential benefits. Then, the cooperative MAS is briefly introduced and the centralized RL approach to solve such problems is presented. That part of the chapter also includes the MLAC algorithm which has been used as a basis to MAAC methods proposed in this chapter.

After conceptual knowledge and centralized learning have been explained, MMDP framework for cooperative MARL has been introduced. Later on, three IL based approaches in MARL have been introduced: independent learners, optimistic learners and lenient learners. Each of these approaches is followed by an algorithm that is developed from the extension of MLAC algorithm to corresponding approach, and the proposed algorithms are called Independent Multi-Agent MLAC, Optimistic Multi-Agent MLAC and Lenient Multi-Agent MLAC respectively.
Chapter 4

Results and Comparison

The capabilities of Optimistic Multi-Agent Model Learning Actor-Critic (MLAC) and Lenient Multi-Agent MLAC algorithms have been demonstrated in this section. The 2-link manipulator set-up has been used to test the algorithms and the resulting performances are compared with each other and the centralized MLAC algorithm.

This chapter starts with the description of the learning problem that is used to test the algorithms. Later on, the normalized Radial Basis Functions (RBF) function approximators are briefly introduced since such approximators have been used throughout the thesis. Finally, the experiment results will be presented for each algorithm and these results are discussed.

4-1 Learning Problem

In this research, the proposed algorithms will be tested on a 2-link manipulator that includes two agents, one for each actuator. Both agents can observe the complete state yet are unaware of which action that other agent will take. The schematic of the system is given in Figure 4-1.

The full state of the robotic arm is then given by:

$$\theta = [\theta_1 \ \theta_2 \ \dot{\theta}_1 \ \dot{\theta}_2]^T$$

These links are rotating horizontally, i.e. there is no gravitational effects on the links. The agent’s actions are the torques applied to the links: $\tau_1$ and $\tau_2$ for agents 1 and 2 respectively. The joint action is defined as follows:

$$\boldsymbol{\tau} = [\tau_1 \ \tau_2]^T$$
The actions taken by the agents are set to be saturated with $\tau_{\text{max},j}$. Additionally, the angular velocities of the links are assumed to be bounded by $\dot{\theta}_{\text{max},j}$. These saturation values are given as follows:

$$
\begin{align*}
\tau_{\text{max},1} &= 2 \text{ N m} \\
\tau_{\text{max},2} &= 1 \text{ N m} \\
\dot{\theta}_{\text{max},1} &= 2\pi \text{ rad s}^{-1} \\
\dot{\theta}_{\text{max},2} &= 2\pi \text{ rad s}^{-1}
\end{align*}
$$

The equation of motion for this system can be given as:

$$
M(\theta_1, \theta_2) \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} + C(\theta) \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} = \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} \tag{4-1}
$$

where the matrices $M(\theta_1, \theta_2)$ and $C(\theta)$ are defined as follows.

$$
M(\theta_1, \theta_2) = \begin{bmatrix} P_1 + P_2 + 2P_3 \cos(\theta_2) & P_2 + P_3 \cos(\theta_2) \\ P_2 + P_3 \cos(\theta_2) & P_2 \end{bmatrix} \\
C(\theta) = \begin{bmatrix} b_1 - P_3 \dot{\theta}_2 \sin(\theta_2) & -P_3(\dot{\theta}_1 + \dot{\theta}_2) \sin(\theta_2) \\ P_3 \dot{\theta}_2 \sin(\theta_2) & b_2 \end{bmatrix}
$$

The meaning and values parameters that are used to simulate the system are presented in Table 4-1. From the parameters provided in this table, $P_1$, $P_2$ and $P_3$ are computed as follows.

$$
\begin{align*}
P_1 &= m_1 c_1^2 + m_2 l_1^2 + I_1 \\
P_2 &= m_2 c_2^2 + I_2 \\
P_3 &= m_2 l_1 c_2
\end{align*}
$$
### Table 4-1: Model parameters of 2-link robotic manipulator used in simulations

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length of link 1</td>
<td>$l_1$</td>
<td>0.4 m</td>
</tr>
<tr>
<td>Length of link 2</td>
<td>$l_2$</td>
<td>0.4 m</td>
</tr>
<tr>
<td>Mass of link 1</td>
<td>$m_1$</td>
<td>1.25 kg</td>
</tr>
<tr>
<td>Mass of link 2</td>
<td>$m_2$</td>
<td>0.8 kg</td>
</tr>
<tr>
<td>Center of mass of link 1</td>
<td>$c_1$</td>
<td>0.2 m</td>
</tr>
<tr>
<td>Center of mass of link 2</td>
<td>$c_2$</td>
<td>0.2 m</td>
</tr>
<tr>
<td>Inertia of link 1</td>
<td>$I_1$</td>
<td>0.0667 kgm$^2$</td>
</tr>
<tr>
<td>Inertia of link 2</td>
<td>$I_2$</td>
<td>0.0427 kgm$^2$</td>
</tr>
<tr>
<td>Damping in joint 1</td>
<td>$b_1$</td>
<td>0.08 kg s$^{-1}$</td>
</tr>
<tr>
<td>Damping in joint 2</td>
<td>$b_2$</td>
<td>0.02 kg s$^{-1}$</td>
</tr>
</tbody>
</table>

Throughout learning, the states variables and actions are normalized with corresponding maximum values so that normalized state variables and actions have values between $-1$ and $1$. The reason for that is to obtain uniformity in different function approximation methods and learning schemes. Thus, the value functions and policies are a function of normalized $x$ and $u$.

The goal of the problem is to stabilize the links at $[0 \ 0]$ position, i.e. to reach the state $x = [0 \ 0 \ 0 \ 0]^\top$ as fast as possible and stay there till the rest of the episode. To accomplish such task, a quadratic global reward structure is selected which is presented as follows:

$$r_{k+1} = -\theta^\top Q \theta$$

where the weighting matrix $Q$ equals to:

$$Q = \text{diag}(5, 5, 0.1, 0.1)$$

When it is needed the global reward is factorized into following local reward functions:

$$r_{k+1}^j = -[\theta_j, \dot{\theta}_j] Q^j \begin{bmatrix} \theta_j \\ \dot{\theta}_j \end{bmatrix}$$

where $Q^j = \text{diag}(5, 0.1)$. Note that, this update rule is not suitable for algorithms that are optimistic at the start of the learning. The reason for that is as the rewards collected are always negative, the temporal difference for all states will be negative since the initial values for current state and the next state will be selected equally. Therefore, if an agent acts optimistically at the very beginning of the learning, the value functions of that agent will not be updated at all and remains as the initial value. Thus, the learning cannot be performed.

To overcome this issue, the rewards for algorithms that are optimistic at the beginning (in
this case both of the tested algorithms) are biased with a positive number so that some of the rewards encountered become positive. For this application, this positive number is selected as 10. Then, the local rewards for Optimistic Multi-Agent MLAC and Lenient Multi-Agent MLAC algorithms are computed as:

\[ r_{k+1}^j = -[\theta_j, \dot{\theta}_j] Q_j^j \left[ \frac{\theta_j}{\dot{\theta}_j} \right] + 10 \]

At this point, it should be noted that the cumulative reward curves for these algorithms are plotted with the rewards collected without the existence of this corrective term in order to compare the results with centralized learning.

In this application the episodes are initialized from \( \theta_1 = 2 \text{ rad} \) and \( \theta_2 = 3 \text{ rad} \) which are arbitrarily selected. The sampling time is set to be 0.03 s, and the duration of each episode is determined to be to 6 s. Thus, each episode includes 200 time steps.

As indicated in Section 3-4, local states should be determined for the multi-agent algorithms. Dynamics of the system will be used to determine agent specific local states. The equation of motion of this system states that, complete state is in direct relation with the torque applied to the agent 1. Thus, the complete state will be used as the local state for agent 1 as follows.

\[ x^1 = x \quad (4-2) \]

On the other hand, the angular position of the agent 1 does not directly affects the policy of the agent 2 as it appears in equation of motion for this system. It is true that, the policy followed by agent 1, which is a direct function of the angular position of it, will influence the policy of agent 2. This indicates that, there is an indirect effect of \( \theta_1 \) to the policy of the agent 2. However, to simplify the learning problem for one agent, this influence will not be taken into account. The downside of such approach is that the multi-agent algorithms will be less robust and more susceptible to divergence problems due to lack of information from the incomplete states. However, the simplification brought by the reduction in dimension of the approximated functions is observed to be more useful in this application. For that purpose, the normalized angular position of the agent 1, \( x_1 \), is discarded from the local state of the agent 2.

\[ x^2 = [x_2 \ x_3 \ x_4] \quad (4-3) \]

In this thesis, the algorithms are tested on only simulation of the test set-up and real-time experiment did not performed. To simulate the state space mode of the system is integrated by using the 4th order Runge-Kutta method with a with time step 0.003 s.
4-2 Function Approximation—Radial Basis Functions

In this project normalized RBF has been selected to approximate the functions which have the following form. Let $h_w(x)$ is the function to be approximated.

$$h_w(x) = w^T \psi(x)$$  \hspace{1cm} (4-4)

where the $i^{th}$ element of the basis function vector is normalized as follows;

$$\psi^i(x) = \frac{\tilde{\psi}^i(x)}{\sum_k \tilde{\psi}^k(x)}$$  \hspace{1cm} (4-5)

Then $\tilde{\psi}^i(x)$ has the form of:

$$\tilde{\psi}^i(x) = \prod_k \exp \left( \frac{(x_k - c_{i,k})^2}{2b_{i,k}} \right)$$  \hspace{1cm} (4-6)

where $x_k$ is the $k^{th}$ element of state vector $x$, $c_{i,k}$ denotes the centers for each state variable and basis function, and $b_{i,k}$ determines the widths of the basis functions.

Since some of the state variables has a circular manifold, wrapping is necessary to accurately approximate some functions. To solve this problem, keeping other components the same, RBF that includes the first and last centers of the state variable that needed to be wrapped are compared, and the smaller of these two RBF is discarded. These two RBF are entitled to the same parameter value. In this application, wrapping is used while approximating the value functions and the policies for angular state variables. The process models used in algorithms are excluded from wrapping practice since the accuracy of model approximation is not crucial.

For all algorithms applied to 2-link robotic arm set-up, 7 equidistant RBF per state variable and input is used for actor, critic and the model of the system. The neighboring RBF are arranged to intersect at height of 0.8. Figure 4-2 presents the exponential components of RBF for a single variable.

4-3 Results

In this section, the performance of the learning algorithms are presented and compared with each other in terms of their convergence speed and the optimality of their final policy. The learning experiments are repeated 20 times to generate mean learning curve in terms of the cumulative reward per trial and the 95% confidence interval to corresponding learning curve. The function approximators are initialized with 0 parameters. The exploration is performed by zero-mean white noise for each agent and the standard deviation of this exploration gradually decays as the learning experiment continues. The initial and the final standard deviations for the exploration are given in the parameter tables for learning algorithms.
The learning parameters of these algorithms are roughly tuned by trial thus they will not yield to most efficient learning. As the MLAC is not the most suitable algorithm to extend to the multi-agent learning, these parameters are not the optimal ones for neither the initial nor the end performances of the corresponding algorithms. Yet they illustrate the general behavior of the learning algorithms and prove that actor-critic algorithms can be practically applied to multi-agent problems.

### 4-3-1 Centralized MLAC

In this test centralized MLAC is applied to the given learning problem. The learning parameters for this application is given in Table 4-2. The resulting learning curve is given in Figure 4-3.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>$\alpha_a$</th>
<th>$\alpha_c$</th>
<th>$\alpha_p$</th>
<th>$\gamma$</th>
<th>$\lambda$</th>
<th>$\sigma_{(start)}$</th>
<th>$\sigma_{(end)}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>0.2</td>
<td>0.45</td>
<td>0.70</td>
<td>0.97</td>
<td>0.5</td>
<td>[0.5 0.5]</td>
<td>[0.05 0.05]</td>
</tr>
</tbody>
</table>

Table 4-2: Reinforcement learning parameters for the centralized learning. $\sigma$ denotes the standard deviation for exploration in agent 1 and agent 2 as a vector.

Centralized MLAC takes about 250-300 trials to converge to a decent policy as can be seen from Figure 4-3. The confidence interval is rather large initially due to the high exploration rates and the dynamical influence of the agents to each other. However, at the later stages of the learning as the agents learn good policies the confidence interval gets smaller. The figure also indicates that the end learning performance tend to improve slightly at the end of the learning experiment. The mean learning curve gets closer to the -1000 at the final stage of the experiment which is approximately the cumulative reward that can be collected per trial.
4-3 Results

Figure 4-3: Results for the centralized MLAC algorithm. The mean and 95% confidence region are computed from 20 learning experiments.

Figure 4-4: Time response of the system with the feedback controller given in (4-7). The cumulative reward for this simulation is computed as $-981$. With an acceptable full state feedback controller in the form of $u = -Kx$. Figure 4-4 gives the time response of the system with such controller and it is observed that the cumulative reward collected with that controller is $-981$. The controller is formulated as follows.

$$\tau = \begin{bmatrix} 5 & 0 & 2 & 0 \\ 0 & 4 & 0 & 2 \end{bmatrix} \theta$$ (4-7)

4-3-2 Optimistic Multi-Agent MLAC

The Optimistic Multi-Agent MLAC algorithm was applied using the parameter settings in Table 4-3. This results in the learning curve for the 2-link robotic manipulator problem shown in Figure 4-5.
## Results and Comparison

<table>
<thead>
<tr>
<th>Parameter</th>
<th>$\alpha_a$</th>
<th>$\alpha_c$</th>
<th>$\alpha_p$</th>
<th>$\gamma$</th>
<th>$\lambda$</th>
<th>$\sigma$(start)</th>
<th>$\sigma$(end)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent 1</td>
<td>0.07</td>
<td>0.15</td>
<td>0.70</td>
<td>0.97</td>
<td>0.5</td>
<td>0.5</td>
<td>0.05</td>
</tr>
<tr>
<td>Agent 2</td>
<td>0.15</td>
<td>0.4</td>
<td>0.70</td>
<td>0.97</td>
<td>0.5</td>
<td>0.5</td>
<td>0.05</td>
</tr>
</tbody>
</table>

**Table 4-3:** Reinforcement learning parameters for the Optimistic Multi-Agent MLAC algorithm. $\alpha_p$ is a global parameter that is used to train global process model.

**Figure 4-5:** Results for the Optimistic Multi-Agent MLAC algorithm. The mean and 95% confidence region are computed from 20 learning experiments.

Figure 4-5 indicates that the learning is slightly faster than the centralized case. As it can be observed from the plot, the learning is performed in two stages. Agent 2, which is the more aggressive agent, locally converges to a decent policy at the first step. Then agent 1 with the low learning rates converges to a policy. The later stages of the learning (after 300 episodes), the agents start to diverge from the policy as the cumulative rewards decrease. The widening trend of the confidence interval at the final stages of the experiments implies that, the learning tend to get unstable at the later stages of the experiment. In other words, the agents diverged from better policies in many experiments.

Considering the nature of the optimistic learners, this phenomenon is expected due to the overestimation problem. Although, the learning problem is highly deterministic, the explorative actions introduce some stochasticity. The direct relation between the policy update and the overestimated value functions in the MLAC algorithm also adds on to this problem. Furthermore, as it is discussed in Section 4-1, the incomplete states used in agent 2 make the learning less robust since the Markov property is now violated by not only the changing transient functions for each agent, but also due to the hidden variables.

In the next section, the Lenient Multi-Agent MLAC is applied to the learning problem that overcomes the overestimation problem.
4-3-3 Lenient Multi-Agent MLAC

The Lenient Multi-Agent MLAC algorithm was applied using the parameter settings in Table 4-4. In this application, the reinforcement learning parameters are kept the same with the optimistic learners. The leniency parameters are selected to behave fully optimistic in first 100 trials and have a decaying level of optimism later on. Additionally, in this application leniency is updated in each episode contrary to the update rule stated in Reminder: don’t forget the section. For both agents, the probability of acting as an optimistic learner is given in Figure 4-6.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>$\alpha_a$</th>
<th>$\alpha_c$</th>
<th>$\alpha_p$</th>
<th>$\gamma$</th>
<th>$\lambda$</th>
<th>$\sigma_{(start)}$</th>
<th>$\sigma_{(end)}$</th>
<th>$\kappa$</th>
<th>$\xi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent 1</td>
<td>0.07</td>
<td>0.15</td>
<td>0.70</td>
<td>0.97</td>
<td>0.5</td>
<td>0.05</td>
<td>3.07</td>
<td>0.985</td>
<td></td>
</tr>
<tr>
<td>Agent 2</td>
<td>0.15</td>
<td>0.4</td>
<td>0.70</td>
<td>0.97</td>
<td>0.5</td>
<td>0.05</td>
<td>3.07</td>
<td>0.985</td>
<td></td>
</tr>
</tbody>
</table>

Table 4-4: Reinforcement learning parameters for the Lenient Multi-Agent MLAC algorithm. $\alpha_p$ is a global parameter that is used to train global process model. The leniency parameters, $\kappa$ and $\xi$, are used to update the leniency in each episode, not for each visited state.

The resulting learning curve for this application is presented in Figure 4-7 together with the 95% confidence interval. As expected, the performance in the early stages of the learning is quite similar with Optimistic Multi-Agent MLAC. This is not a surprise considering the fact that the learning parameters are the same, and through first 100 episodes the agents are fully optimistic. In fact, the similarity between the learning curves of the lenient and the optimistic agents continues up until the mid-learning period.

On the other hand, the end performance of the lenient learners is better than the optimistic ones. The agents do not diverge from the policy they learned at the end of the learning and the learning is more robust as the confidence interval is narrower than the optimistic agents case. As mentioned in [45], the agents benefit from the optimism at the start of the learning since they do not stuck with a bad policy where they simultaneously need to explore a better policy. However, at later stages of the of the learning experiment, it is more advantageous to
learn the real value functions instead of the overestimated ones. Thus, the lenient agents are superior to the optimistic learners as it can be seen from this application.

4-4 Comparison and Discussion of the Results

In this section the results will be compared and discussed. Figure 4-8 shows the mean learning curves of the centralized MLAC, Optimistic Multi-Agent MLAC and Lenient Multi-Agent MLAC in one plot, allowing for a comparison in performance. From this figure, it can be observed that, multi-agent algorithms is slightly faster than the centralized MLAC considering the early learning speed. The more aggressive agent (agent 2) in multi-agent algorithms clearly learns a decent policy in a fast manner which kick starts the learning speed for these algorithms. In that sense, decentralized nature of the proposed multi-agent methods benefit them at the start of the learning. However, considering the convergence speed to a solution where both agents achieve an acceptable policy, all 3 methods have performed similarly. Although the application with the lenient learners converges slightly faster than other algorithms, this result does not indicate a conclusive superiority in terms of the convergence speed.

As the end performances of the used methods compared, it can be observed that the centralized MLAC is better than its multi-agent extensions and the performance is still in increasing trend for it. This result is expected as the centralized learning methods do not yield to coordination problems, and the Markov and stationary assumptions hold for such methods. Although there is a slight difference between the cumulative rewards at the end, it yields to a huge performance drop in the learned policy. The reason for that is the use of quadratic
reward functions which will penalize the large deviations from the reference state much more than the small ones. The time response of the system after a generic final episode for each algorithm is given in Figures 4-9, 4-10 and 4-11. The differences between the cumulative rewards and the learned controller performance can be observed from these plots.

Figure 4-8 also indicates that the Optimistic Multi-Agent MLAC performs the worst at the end of the experiment. After around episode 300, the mean learning curve of this algorithm tends to decline. Additionally, the confidence interval of the algorithm gets broader and broader after some point. Considering these facts, it can be said that the agents are tend to diverge from the learned policy and the learning becomes fragile. As mentioned in Section 3-5, the main reason for that is the overestimated value functions resulted from the use of optimistic learners. Even the smallest overestimations of the value functions due to the slight stochastic behavior (in this case as a result of exploration) accumulate through the learning and the approximated value functions considerably deviate from the optimal ones at the late stage of the learning. Thus, the policies updated with such value functions yield to a divergent characteristic of the algorithm.

From Figure 4-8, it is clear that the late performance of the Lenient Multi-Agent MLAC algorithm is superior to the Optimistic Multi-Agent MLAC. The reason for that is the lenient agents do not deal with overestimation of the value functions as optimistic agents do. On the other hand, the mean learning curves reveal that lenient learners cannot improve their performance after some point as centralized method does. This phenomenon can be explained by two properties of the Lenient Multi-Agent MLAC algorithm: uncoordinated agents and the incomplete states in agents. The first reason can be further explained as the shadowed equilibrium problem that is mentioned in Section 3-5. As the agents are not act
Results and Comparison

Figure 4-9: Final time response for the centralized MLAC after one learning experiment. Left figure gives the time response of the agent 1 and figure on the right gives the time response of the agent 2. The cumulative reward for this episode is computed as -1268.

Figure 4-10: Final time response for the Optimistic Multi-Agent MLAC after one learning experiment. Left figure gives the time response of the agent 1 and figure on the right gives the time response of the agent 2. The cumulative reward for this episode is computed as -2191.

Figure 4-11: Final time response for the Lenient Multi-Agent MLAC after one learning experiment. Left figure gives the time response of the agent 1 and figure on the right gives the time response of the agent 2. The cumulative reward for this episode is computed as -1762.
optimistically at the end of the learning, they cannot abandon their already learned policy by simply changing their individual policy unless a simultaneous change in the both agents policies. In other words, the agents experience the shadowed equilibrium problem, but only after they learn an acceptable policy as a result of their optimistic behavior at the start of the learning. The second reason is due to the fact that the agent 2 does not aware of the goal the agent 1. Thus, the agent 2 cannot predict the action that agent 1 takes even if it adapts to the current policy of the agent 1. The increases the level of coordination problem that is faced by the agents. The incomplete states problem is also encountered by the Optimistic Multi-Agent MLAC. However, the effects of it in that context is insignificant compared to the overestimation problem.

4-5 Summary

In this chapter, centralized MLAC, Optimistic Multi-Agent MLAC and Lenient Multi-Agent MLAC algorithms have been applied to the 2-link manipulator problem to test the performance of the proposed algorithm. The chapter starts with the description of the learning problem. Then, the RBF function approximation method that has been used in the algorithms has been introduced briefly. Later on the simulation results for the centralized MLAC algorithm have been presented followed by the results of the Optimistic Multi-Agent MLAC and Lenient Multi-Agent MLAC algorithms. Finally, the chapter is concluded by the comparison and the discussion of the results.
In this work, it is found that actor-critic and Multi-Agent Reinforcement Learning (MARL) methods can be combined to solve cooperative Multi-Agent Systems (MAS) problems. For that purpose, Model Learning Actor-Critic (MLAC) algorithm is extended to work in multiple agent environments by using Independent Learners (IL) approach, and 3 algorithms based on this idea are proposed: Independent Multi-Agent MLAC, Optimistic Multi-Agent MLAC and Lenient Multi-Agent MLAC. These algorithms are tested on 2-link manipulator problem among with the centralized MLAC. While testing the algorithms, simulation experiments indicated that Independent Multi-Agent MLAC is not suitable for the given problem possibly due to coordination problems and lack of robustness to these problems. Therefore, only the results of centralized learning, Optimistic Multi-Agent MLAC and Lenient Multi-Agent MLAC are presented in Chapter 4.

As discussed in Chapter 4, proposed and tested Multi-Agent Actor-Critic (MAAC) methods are capable of performing in a similarly to centralized actor-critic learning, if not better. Although the optimal learning parameters and function approximators did not found, the results of the simulations implies that faster learning can be achieved by MAAC than the centralized actor-critic techniques.

Concerning the final performance of the learning experiments, it is observed that, the centralized learning tend to perform better than the proposed multi-agent algorithms. In fact, centralized MLAC was still improving its performance at the end of the learning experiments while Lenient Multi-Agent MLAC was settled to a lower cumulative reward, and Optimistic Multi-Agent MLAC was started to diverged from a learned policy. In centralized learning the the actions are selected from a joint action space with respect to a full state, and the transition function between the next state and the current state-action pair is stationary. Thus, it is
expected to have a better end performance with centralized MLAC algorithm. On the other hand, the divergence of the Optimistic Multi-Agent MLAC was due to the overestimation of value functions as explained in Chapter 4.

The primary goal of this research was to prove that a MAAC concept can be applied to learn policies for a control problem. Thus, the computational advantages brought by parallel computation property of MAS was not tried to be exploited, and did not measured. Instead, simpler coding and previous experience on MLAC considered more important throughout the study. When computational expenses did not taken in to account while coding, total simulation time of centralized MLAC algorithm observed to be approximately 1.5 times more than its multi-agent counterparts.

A persisting problem in reinforcement learning is the proper tuning of the learning parameters and function approximators. Choosing these was proved to more difficult in the multi-agent case as the number of parameters to be picked increases. Besides, depending on the problem and the algorithm, these parameter might influence the convergence more due to the interaction of the agents. Although it is not discussed in this work, implementing coordination between the agents might complicate parameter tuning even more considering the parameters added by the coordination mechanisms.

To sum up, this thesis work should serve as an exploratory research on MAAC for cooperative tasks. This study can be an initial point to theoretical research on such methods as well as the many practical applications.

5-1 Future Work

As mentioned couple of times the algorithms was tested in simulation environment. The next step to prove the proposed methods should be testing them in real-time experiments. In that way, the performance of the algorithms for practical applications can be measured and a more solid discussion on the applicability of the MAAC can be made. Additionally, the proposed algorithms might be tested on different test set-ups to observe the effects of the dynamics of the problem.

Another, immediate research topic based on this thesis is to find the optimal learning parameters for different function approximators used in the proposed algorithms. In this way, the capabilities of proposed algorithms can be found and a more clear comparasson can be made between the centralized learning and the proposed algorithms.

One of the major research questions that can be addressed in the future is to test the performance of the different actor-critic algorithms combined with IL approaches. For instance, algorithms proposed in [7, 40] can benefit from the computational advantages of IL approach.
more than other algorithms as they use Q values. Additionally, the implementation of IL and its variants to different algorithms can yield to different effects on the performance.

The IL approaches that is in scope of this thesis does not incorporate any coordination between the agents. Since the test-set up used has only two agents, the methods may not suffer from the lack of coordination as much as other applications that has more agents or more complex dynamics. To implement coordination between the agents, MARL techniques other than IL methods, such as Distributed Value Functions [34] and Coordination Graphs [35] can be used to extend actor-critic algorithms to multi-agent problems. Explicit coordination methods can also be used such as social roles and communication [46]. By introducing the coordination between the agents, the algorithms will be more robust and possibly perform better.
Conclusion
Bibliography


## Glossary

### List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RL</td>
<td>Reinforcement Learning</td>
</tr>
<tr>
<td>MDP</td>
<td>Markov Decision Process</td>
</tr>
<tr>
<td>TD</td>
<td>Temporal Difference</td>
</tr>
<tr>
<td>MC</td>
<td>Monte Carlo</td>
</tr>
<tr>
<td>LSTD</td>
<td>Least-Squares Temporal Difference</td>
</tr>
<tr>
<td>MAS</td>
<td>Multi-Agent Systems</td>
</tr>
<tr>
<td>MMDP</td>
<td>Multi-Agent Markov Decision Process</td>
</tr>
<tr>
<td>Dec-MDP</td>
<td>Decentralized Markov Decision Process</td>
</tr>
<tr>
<td>Dec-POMDP</td>
<td>Decentralized Partially Observable Markov Decision Process</td>
</tr>
<tr>
<td>MARL</td>
<td>Multi-Agent Reinforcement Learning</td>
</tr>
<tr>
<td>DVF</td>
<td>Distributed Value Functions</td>
</tr>
<tr>
<td>IL</td>
<td>Independent Learners</td>
</tr>
<tr>
<td>MAAC</td>
<td>Multi-Agent Actor-Critic</td>
</tr>
<tr>
<td>MLAC</td>
<td>Model Learning Actor-Critic</td>
</tr>
<tr>
<td>RBF</td>
<td>Radial Basis Functions</td>
</tr>
</tbody>
</table>