Applying Reinforcement Learning to develop Data Communication Protocols in Networked Control Systems

Q-learning with Function Approximation Techniques

Tiago Sousa Ferreira

Thesis to obtain the Master of Science Degree in

Electrical and Computers Engineering

Supervisors: Prof. Alexandre Rodrigues Mesquita
Prof. Luís Manuel Marques Custódio

Examination Committee
Chairperson: Prof. João Fernando Cardoso Silva Sequeira
Supervisor: Prof. Luís Manuel Marques Custódio
Member of the Committee: Prof. Manuel Fernando Cabido Peres Lopes

December 2018
Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.
I would first like to thank my thesis advisor, Prof. Alexandre Mesquita, of the Department of Electronics Engineering at the Federal University of Minas Gerais, for the patience, guidance, encouragement and advice he has given me throughout the making of this thesis. I have been extremely lucky to have an advisor who cared so much about my work, as the door to its office was always open whenever I needed help with my thesis. As a foreign student working on my master thesis in Belo Horizonte, I really appreciated the extra care and the hospitality that Prof. Mesquita has shown during my time in Brazil.

Secondly, I would also like to thank my thesis advisor, Prof. Luís Custódio, of the Institute for Systems and Robotics at Instituto Superior Técnico, for trusting in my capabilities and for having the patience and availability to help me whenever I asked. I am gratefully indebted to him for his very valuable comments. Prof. Custódio’s critical thinking and on-point remarks allowed me to highly improve the writing and the structure of this thesis.

I must express my gratitude to my mother, for her continued encouragement and support throughout the years. Without her help, I would not have been able to be where I am right now, and I will forever be in debt. I would also like to thank my family, the ones living in Portugal and in Brazil, whose help has been valuable in different ways, without them this would not have been possible.
Abstract

Wireless data communication changed the way information is shared worldwide, and has become increasingly more present in engineering applications. One of the technologies that uses wireless data networks, in sectors such as automation and robotics, are networked control systems (NCS), control systems where the control loop is closed through a wireless communication network.

A problem that compromises NCSs is the constant loss of information by the networks, which affects the behavior and stability of the systems to control. Therefore, it is of extreme importance to find means to avoid or correct that loss of information. One of the proposed approaches used is to create data transmission protocols, that coordinate the information transmission in data networks. The protocols proposed in this thesis are based on the redundant transmission property studied in [1]. In lossy networks the probability of successful communication can be significantly increased by transmitting multiple copies of a same message through independent channels, without increasing the transmission delays.

The goal of this thesis is to evaluate and to design better terms for the protocols that rule the communication in NCSs, with the objective of optimizing the estimation error - cost of communication tradeoff for different contexts. It is used Q-learning, a reinforcement learning technique. Q-learning in its classical tabular form is not efficient for large state space problems. Thus, in this thesis, it is approximated by two different function approximation methods, neural networks and linear parametrization, with the endgame of generating laws that optimize the data transmission rate in NCSs.

The results are presented for each of the Q-learning approximations applied to two different NCSs: $NCS_1$, where only one node transmits using three wireless communication channels, and $NCS_2$, that has two nodes sharing the wireless network with three communication channels, where one channel is used by both nodes for redundant transmission. Both function approximation techniques were able to obtain optimal protocols for $NCS_1$, with equal or better estimation error - cost of communication relation than trivial protocols proposed. The good results obtained for $NCS_1$ were a motivation for the application of the same techniques in $NCS_2$, that introduced a more complex problem. The results for the neural networks approximation were satisfactory, as optimal protocols were obtained that had a equal or better performance than the trivial protocols proposed. However, the results for the linear parametrization approximation did not fulfill the expectations.

Keywords: Networked Control Systems, Data Transmission Protocols, Reinforcement Learning, Neural Networks, Linear Parametrization
Resumo

A comunicação de dados em redes sem fios mudou a maneira como a informação é partilhada no mundo, e está cada vez mais presente em aplicações de engenharia. Uma das tecnologias que utiliza redes de dados sem fios, em sectores como a robótica e a automação, são os sistemas de controlo em rede (networked control systems, ou NCSs), sistemas de controlo em que a malha é fechada através de uma rede de comunicação sem fios.

Um problema que compromete os NCSs é a constante perda de informação das redes, o que afeta o comportamento e estabilidade dos sistemas a controlar. Assim, é de extrema importância encontrar meios que permitam evitar ou remediar essa perda de informação. Uma das abordagens propostas é criar protocolos de transmissão de dados, que coordenem a transmissão de informação na rede de dados. Os protocolos propostos nesta tese são baseados na propriedade da transmissão redundante estudada em [1]. Em redes com perdas de informação, a probabilidade de comunicar com sucesso pode aumentar significativamente ao transmitir múltiplas cópias da mesma mensagem usando canais independentes, sem aumentar os atrasos nas transmissões.

O objetivo desta tese é avaliar e construir melhores termos para os protocolos que controlam a comunicação em NCSs, com o objetivo de optimizar a relação erro de estimação - custo de comunicação. É utilizado Q-learning, uma técnica de reinforcement learning. Q-learning, na sua clássica forma tabular, não é eficiente para problemas com grande espaço de estados. Assim, nesta tese, será aproximado por duas técnicas de aproximação de funções, redes neurais e parametrização linear, com o objetivo de gerar leis que optimizem a taxa de transmissão de dados em NCSs.

Os resultados são apresentados para cada uma das aproximações de Q-learning aplicadas em dois NCSs diferentes: NCS$_1$, em que apenas um nó transmite utilizando três canais de comunicação, e NCS$_2$, em que dois nós partilham a rede sem fios, composta por três canais de comunicação, em que um é utilizado por ambos os nós para transmissões redundantes. Ambas as técnicas de aproximação de funções conseguiram obter protocolos ótimos para o NCS$_1$, com igual ou melhor relação erro de estimação - custo de comunicação que os protocolos triviais propostos. Os resultados obtidos para o NCS$_1$ serviram de motivação para a aplicação das mesmas técnicas no NCS$_2$, que introduziu um problema mais complexo. Os resultados para a aproximação com redes neurais foram satisfatórios, dado que foram obtidos protocolos ótimos que tiveram uma performance igual ou melhor que os protocolos triviais propostos. No entanto, os resultados para a aproximação com parametrização linear ficaram aquém das expectativas.

Palavras-Chave: Sistemas de Controlo em Rede, Protocolos de Transmissão de Dados, Aprendizagem por Reforço, Redes Neurais, Parametrização Linear
# Contents

## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>xi</td>
<td></td>
</tr>
</tbody>
</table>

## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>xiii</td>
<td></td>
</tr>
</tbody>
</table>

## 1 Introduction

1.1 Motivation .................................................. 3
1.2 Objectives .................................................. 8
   1.2.1 Main Objective ........................................ 8
   1.2.2 Specific Objectives ................................... 8
1.3 Contributions .............................................. 8
1.4 Thesis Outline .............................................. 9

## 2 Theoretical Background

2.1 Networked Control Systems ................................. 11
   2.1.1 Introduction .......................................... 11
   2.1.2 Categories and Components ............................ 11
   2.1.3 NCS Challenges ....................................... 13
   2.1.4 NCS Solutions ........................................ 13
   2.1.4.1 Communication Protocol Design .................... 13
2.2 Learning .................................................... 15
   2.2.1 Learning Paradigms .................................... 15
   2.2.2 Reinforcement Learning ................................ 17
      2.2.2.1 Introduction .................................... 17
      2.2.2.2 Exploration and Exploitation .................... 18
      2.2.2.3 Dynamic Programming ............................ 18
      2.2.2.4 Generalization .................................. 20
2.3 Neural Networks ............................................. 21
   2.3.1 Introduction .......................................... 21
   2.3.2 Network Architectures and Properties ................. 21
2.4 Stochastic Approximation .................................. 22
   2.4.1 Introduction .......................................... 22
   2.4.2 Stochastic Gradient Descent ......................... 23
   2.4.3 Linear Parametrization ............................... 24
      2.4.3.1 Features ........................................ 24
   2.4.4 Stochastic Newton-Raphson ............................ 24
      2.4.4.1 Zap Stochastic Newton-Raphson ................ 25
List of Tables

3.1 Properties of the environment for each NCS ........................................... 36
3.2 Parameters used in training and validation of the algorithms from section 3.4.1 .... 43
3.3 Parameters of the algorithms from section 3.4.2 ....................................... 44

4.1 State space matrices for NCS\(_1\) and NCS\(_2\) ......................................... 45
4.2 Parameters used in training of the algorithm 1 ............................................ 46
4.3 Comparison between the trivial protocols and the Q protocol generated by algorithm 1 \((\alpha = 0.8, \beta = 0.2)\) ......................................................... 47
4.4 Parameters used for algorithm 3 ................................................................. 48
4.5 Comparison between the trivial protocols and the Q protocol generated by algorithm 3 \((\alpha = 0.8, \beta = 0.2)\) ......................................................... 48
4.6 Parameters used in training of the algorithm 2 ............................................ 51
4.7 Comparison between the trivial protocols and the Q protocol generated by algorithm 2 \((\alpha = 0.6, \beta = 0.4)\) ......................................................... 52
4.8 Parameters used for algorithm 4 ................................................................. 53
4.9 Comparison between the trivial protocols and the Q protocol generated by algorithm 4 \((\alpha = 0.6, \beta = 0.4)\) ......................................................... 54
List of Figures

1.1 Basic block diagram of a data communication system - source: [2] .......................... 4
1.2 A typical networked control system - source: [3] ..................................................... 4

2.1 Structures of networked control systems - source [3] ............................................... 12
2.2 Networked control systems used in this thesis ............................................................ 12
2.3 Supervised Learning diagram - source: [4] ................................................................. 16
2.4 Unsupervised Learning diagram - source: [4] ............................................................. 16
2.5 Reinforcement Learning diagram - source: [4] ............................................................. 17
2.6 A unit of a typical neural network - source: [5] ............................................................ 22
2.7 Network architectures - source [5] .............................................................................. 22

3.1 NCS architecture considering only one channel ............................................................ 27
3.2 Simulation of the output and error of the system represented in Figure 3.1 ................ 29
3.3 NCS1 - NCS with one agent and three channels .......................................................... 29
3.4 Simulation of the output and errors of the system represented in Figure 3.3 .......... 30
3.5 NCS2 - NCS with two agents and three channels ......................................................... 31
3.6 Simulation of the output and errors of the system represented in Figure 3.5 .......... 32
3.7 Architecture of the Q-network used (source [6]) ............................................................ 38

4.1 Response of a Q protocol generated by algorithm 1 for $\alpha = 0.8, \beta = 0.2$ .......... 46
4.2 Q protocols obtained by algorithm 1 .......................................................................... 47
4.3 Q protocols obtained by algorithm 3 .......................................................................... 49
4.4 Response of a Q protocol generated by algorithm 2 for $\alpha = 0.6, \beta = 0.4$ .......... 51
4.5 Q protocols obtained by algorithm 2 .......................................................................... 53
4.6 Pareto frontiers for the Q protocols obtained by algorithm 2 .................................... 54
4.7 Percentage of packet collisions in the shared channel as a function of the channels used by each agent ........................................................................... 55
4.8 Values of the error as a function of the channels used by each agent ...................... 56
4.9 Zoom of Figure 4.6c for protocols generated from $\alpha = 1, \beta = 0$ to $\alpha = 0.7, \beta = 0.3$ ................................................................................ 57
4.10 Effects of the application of the control variates technique to algorithm 4 for Q protocol generated with $\alpha = 0.9$ and $\beta = 0.1$ ................................................................................ 58
4.11 Q approximation error of agent 1 for protocols generated by algorithm 4 with $\alpha = 1$ and $\beta = 0$ ................................................................................ 59
4.12 Effects of the application of the features vectors 4.2 and 4.3 to algorithm 4 for protocols generated with $\alpha = 1$ and $\beta = 0$ ......................................................... 60
Chapter 1

Introduction

1.1 Motivation

One of the most important chapters of the history of Mankind is the history of telecommunications. From primitive times, when people would tell their location with smoke signals or use drums to signal others of danger, to the Renaissance period, when ships would transmit information to one another using maritime flags, to the Industrial Revolution, when the electric telegraph and, later, the telephone were invented and allowed sending information from very distant points, the need for people to communicate has catalyzed the evolution of Mankind. The modern days have presented us with a more sophisticated means of communication that changed the way information is shared: wireless data communications, which has connected the world through the most known and most used data network, the Internet. The constant flow of information, available to everyone, brought society to a new age and transformed the world into a global village.

Data communication is, by definition, the exchange of data between two devices via some form of transmission medium which can be wired or wireless [2]. A block diagram of a data communication system is shown in Figure 1.1. A popular choice of transmission medium for this type of communication are data networks, that allow the transmission of digital data, in form of packets, from a transmitter to a receiver, that can be devices like computers or mobile phones. There are three main types of network [2]:

1. **Local Area Network (LAN)**: It is usually a small network that is restricted to a small geographic area, like a building or a campus.

2. **Wide Area Network (WAN)**: WANs cover a broad range of geographic area, and are used to connect LANs and other types of networks together so that users and computers can communicate with computers in other regions. An example of a WAN is the Internet.

3. **Metropolitan Area Network (MAN)**: MAN is a network that connects the users with computer resources in a geographic area that is larger than LAN but not quite as large as WAN.

A specific technology that operates with data networks are Networked Control Systems, or NCSs, which can be defined as control systems where the control components (sensors, actuators, controllers) are connected to each other through a wireless network [7]. A conceptual model of a networked control system is illustrated in Figure 1.2.

NCSs are an upgrade to classical control systems. These last ones are hardwired, which creates several problems:
1. The components have to stay in a close distance to each other.

2. If the control system has a lot of components, the wiring increases the complexity and the overall cost in designing and implementing the control systems.

NCSs present a great deal of advantages comparing to classical control systems. Connecting the control system components such as sensors, controllers and actuators via a wireless network can effectively reduce the complexity of systems. Furthermore, network controllers allow data to be shared efficiently, as it is easier to fuse the global information to take intelligent decisions over a large physical space. They eliminate unnecessary wiring, which is extremely important for control systems that cannot have wires (e.g. controlling autonomous vehicles in environments dangerous for humans). It is easy to add more sensors, actuators and controllers with very little cost and without heavy structural changes to the whole system. Most importantly, they connect cyber space to physical space making task execution from a distance easily accessible (a form of tele-presence).

The potential uses of NCS are numerous and cover a wide range of industries. The two main areas of application for this type of systems are industrial automation and cooperative robotics. Among the applications in industrial automation, there is a special interest in the Smart Grid, where a high sensitivity to communications failures may be experienced. Also, NCS are useful to operate in dangerous environments (e.g. nuclear plant maintenance), given that it introduces the possibility of the human operator to check the performance of the control system without being physically present in the dangerous environment. A review of applications in cooperative robotics may be found in [9],[10]. In general, autonomous agents connected through a network offer solutions to problems from environmental monitoring, search and rescue in disaster situations, surveillance, and border patrol to crop spraying. Cooperative robotics require that the control system components of the robots are connected through a wireless network due to their mobility.

However, the use of a communication network in the feedback control loop makes the analysis and design of an NCS complex, and introduces some issues that degrade the control system’s performance.
and even cause system instability, like the possibility of packet loss. This specific problem will be addressed in this thesis. Packet loss can be caused by internal or external interferences, and can be modeled with a certain probability of failure in sending a packet. This is a problem of major importance, as consecutive packet losses will interfere with the stability and behavior of both the control system and the system to be controlled. One of the big causes of packet loss is the existence of faulty communication channels in some networks, leading to a high probability of packet transmission failure. This challenge gave rise to many important research topics, amongst them the development of data communication protocols, that coordinate the information transfer through the network.

A protocol is a set of rules that govern data communications. It defines what is communicated, how it is communicated, and when it is communicated. Without a protocol, two components may be connected but not communicating, just as a person speaking Portuguese cannot be understood by a person who speaks only English. There are a lot of different techniques in which protocols rely. For example, prioritizing the node with the greatest weighted error, fixing the transmission sequence of nodes in a predetermined order, and sending information packets in a redundant way. As the goal is to rule the communication in a real-time system, these protocols must work in real time.

In [1] it is proved that something as simple as sending a copy of the information packet in a different channel at the same time as the original can increase the transmission rate without compromising the stability of the system, which makes it an interesting characteristic to build protocols on. However, redundant transmission creates complex problems, given that there must be a tradeoff between the success rate of the information packet transmission and the usage of the communication resources, i.e., while transmitting multiple copies of the same packet may increase the success rate of transmissions, it also uses more communication resources and spends more energy in transmitting. Also, in the case that multiple components from different systems share the same communication network, there needs to be an extra care for each component transmitting redundant messages, given that if two or more components use the same communication channel at one moment, there will be packet collision and loss of information.

The redundant transmission property explained in [1] is a technique that utilizes an adaptive allocation of the communication resources. These adaptive techniques are well-suited for NCSs because they permit an increase in the reliability of communication without increasing the transmission delays, to which NCSs have low tolerance. Ideally, the resources would be allocated having in consideration the context of the NCS, that changes with time. If a control system's performance is satisfactory at a certain time (if, for example, the controller has a good state estimate), then it is desirable to allocate a minimal amount of communication resources to this system, saving energy and allowing other systems to use the network. On the other hand, if system disturbances are large to the point of compromising control performance, one should allocate all the available communication resources to the system, to increase the probability of transmitting successfully.

In 3G and 4G generations of wireless communications, channel adaptive techniques are able to provide large improvements under almost any performance metric. These techniques use an adaptive allocation of communication resources as the channel conditions change with time. An important adaptive method is diversity schemes, which corresponds to the transmission of redundant signals through mostly independent channel realizations. Diversity schemes may involve using multiple time slots, frequency slots, antennas or network paths. Identically to the redundant transmission property in [1], many diversity schemes are dynamically exploited in data networks by scheduling transmissions according to the network status, e.g., [15], [16]. However, opposite to [1], these techniques do not take into account nor benefit from the dynamical nature of NCSs.

Given the context of an NCS, how many redundant copies should be transmitted? If the communication network is shared, how often should the redundant copies be transmitted? In sum, what are
the protocols that optimize the transmission rate of an NCS for a determined context? The answer to these questions is not always trivial, specially because it is intimately connected to the characteristics of each NCS. For example, if an NCS’s communication network has a high probability of dropping packets, probably the protocols generated will determine the transmission of more redundant copies than for an NCS with a more reliable communication network.

The ideal would be that the NCS learns on its own the optimal protocol for a determined context. Also, given that each NCS has a characteristic behavior, it would be perfect if it would learn from its own experiences. With this in mind, in this thesis are applied techniques of reinforcement learning (RL) to generate communication protocols for NCSs.

RL is an area of machine learning, and it focus on how agents ought to take actions in order to maximize some notion of cumulative reward. The learning is performed through interaction with the environment, and a reinforcement system (rewards or punishments) gives feedback to the agent concerning the cumulative value of the actions it took [17]. Thus, implementing RL with a feedback function that somehow evaluates the actions taken by an NCS, with special focus on the control performance and the use of communication resources for a determined context, can ultimately make the NCS learn the optimal protocol.

Reinforcement learning is a recognized optimization method, and it has been widely used to design adaptive protocols for different wireless networks. In [15], the authors admit that online RL algorithms can be used to find optimal communication resources allocation. In [18], a survey of applications of RL to develop adaptive routing protocols for mobile ad-hoc networks has been made. Mobile ad-hoc networks are transient networks dynamically formed by a collection of arbitrary located wireless and mobile nodes communicating without any pre-established network infrastructures. The dynamic nature of these networks makes static routing protocols inefficient, thus the use of RL to find optimal adaptive protocols. Other applications of RL to design protocols for wireless networks are found in [19],[20],[21]. While in [19],[20] they use reinforcement learning to develop medium access control (MAC) protocols, ruling the access of a network to several nodes connected to it, in [21] is developed a protocol to use cooperative communication for quality of service (QoS) provisioning in a resources limited wireless sensor network.

The RL technique used in this thesis is Q-learning. This method returns the action-value function $Q^\pi(s,a) = E_{\pi}[R_t|s_t = s, a_t = a]$, which tells us the value of taking an action $a$ in some state $s$ when following a certain policy $\pi$. Q-learning aims to find the optimal policy, by maximizing the Q function at every iteration [17]. It is a tabular method, as for every tuple of state and action $(s,a)$, it returns the value of taking action $a$ in state $s$. Such an approach works reasonably well for small state spaces, but the time to converge and the time per iteration increases rapidly as the space gets larger [22]. Applying Q-learning, as a tabular method, to NCSs is not effective, as their dynamic behavior usually introduces a large state space.

In 2013, a small company in London called DeepMind published their pioneering paper [23]. It demonstrated how an artificial intelligence agent can learn to play games by just observing the screen without any prior information about those games. The results were remarkable. The same model was used to learn seven different games, and in three of them the algorithm performed even better than a human. The algorithm they introduced is called Deep Q Network (DQN), in which they use neural networks (NN) to approximate the Q function in the Q-learning algorithm, that aims to approximate the reward given based on a state. Neural networks are a powerful tool used in machine learning to generalize linear or non-linear functions that map inputs to outputs [5]. The publication of this paper has been acclaimed as the first step towards general artificial intelligence: an AI that can survive a variety of environments, instead of being confined to strict domains.

Deepmind was later bought by Google, and in 2015 they published [24], in which they applied the DQN algorithm to 49 different games. The outcome was again outstanding, as the algorithm results out-
performed human results for around half of the games. The success of [24] inspired a similar approach to the game of Go [25], viewed as the most challenging of classical games for AI due to its enormous search space and the difficulty of evaluating board positions and moves. The results were revolutionary. The program defeated for the first time a human player in the game of Go, and not just any player, but the European Go champion, by five games to zero.

After been introduced in [23] and [24], deep reinforcement learning (DRL) has been used in other areas: in [25] has been applied in optimizing chemical reactions, in [27] has been proposed a DRL framework for online personalized news recommendations, [28] showed how to use DRL to automatically teach systems to allocate and schedule computer resources to waiting jobs, with the objective to minimize the average job slowdown. More recently, DON was applied in [20] to develop MAC protocols and in [29] to dynamically allocate wireless communication channels in Cyber-Physical Systems (CPS), which are systems built through integration of sensors, communication networks, controllers, dynamic (physical) processes and actuators.

Published in 2017, the Zap Q-learning algorithm [30] is based on a two time-scale stochastic approximation algorithm that uses a linear parametrization to approximate the Q-function of the Q-learning algorithm. Its main contribution is the quick convergence to the optimal Q function, which makes it a computationally efficient method. In [30] was simulated a reinforcement learning problem, and the Bellman error was compared for the following variant algorithms of Q-learning: Watkins’ algorithm [31], Watkins’ algorithm with a “polynomial learning rate” $\alpha_n = n^{-0.6}$ [32], Watkins’ algorithm with Rupert-Polyak-Juditsky (RCPJ) averaging [33],[34],[35], Speedy Q-learning [36] and two versions of Zap Q-learning with different learning rates. The results shown that the Bellman error of both versions of the Zap Q-learning algorithm was the fastest to converge to zero.

The Zap Q-learning algorithm showed promising results in [30]. However, and given that it was only published in 2017, there are not many applications of the algorithm published. In fact, the authors uploaded in June 2018 a “users guide” [37]. This paper provided the reader with some tips to improve the performance of the algorithm for application in software, as well as some accelerations techniques.

Instead of using Q-learning as a tabular method, in this thesis we use techniques that aim to approximate the Q function, and can handle huge state spaces. We investigate the use of Q-learning as implemented in [24], where neural networks are used as an approximator for the Q function. In this thesis, neural networks are trained so that they generalize the Q function that maps the state (input) to the value of taking each action in that state (output). We also apply a linear parametrization as implemented in [30] which seeks to approximate the Q function linearly using a set of features.

These techniques are applied to two different NCS architectures:

1. $NCS_1$ - NCS with one agent, transmitting in a network with three communication channels (see Figure 2.2a);

2. $NCS_2$ - NCS with two agents, using the same communication network with three channels, one for each agent and one to be shared by the agents (see Figure 2.2b).

In this context, “agent” is the sensor of each system. The agent’s actions are transmitting in the network. For $NCS_1$, the protocols generated will state how many channels will be used at each packet transmission. As for $NCS_2$, the protocols computed will decide how often each agent sends a redundant transmission through the shared channel.

Bellman error: $B^\pi_{n+1} = -Q^\pi(x_n, u_n) + c(x_n, u_n) + \beta \cdot \min_{u'_n} Q^\pi(x_{n+1})$, where $Q^\pi(x_n, u_n)$ is the approximated Q function of the last iteration, $c(x, u)$ is the cost estimated at the last iteration, $\beta$ is a discount rate, and $\min_{u'_n} Q^\pi(x_{n+1})$ is the approximated Q function of the current iteration estimated by a greedy policy. This is identically zero if and only if $Q^\pi = Q^\star$. 

7
1.2 Objectives

1.2.1 Main Objective

To design communication protocols for networked control systems considering multiple communication channels and multiple agents, using dynamic programming techniques associated with neural networks and linear parameterizations.

1.2.2 Specific Objectives

1. To implement the Q-learning algorithm based on [24] in order to develop data communication protocols in networked control systems.

2. To implement the Q-learning algorithm based on [30] in order to develop data communication protocols in networked control systems.

3. To design and evaluate protocols considering one agent transmitting in multiple communication channels.

4. To design and evaluate protocols considering two agents sharing the same communication network.

5. To compare the protocols generated by the algorithms based on [24] with the algorithms based on [30].

1.3 Contributions

Reinforcement learning has been widely used to develop adaptive protocols in wireless data networks, with proven results [19], [20], [21], [29]. There is not much work done on applying this method to generate protocols in networked control systems. However, the dynamical behavior of NCS and the instability related with high packet drop rates and time delayed transmissions make dynamical protocols with adaptive resource allocations a good strategy to maintain the stability of the systems to control, while making a smart management of the communication resources.

Communication protocols are highly dependent of the NCS architecture, i.e., factors like the number of the nodes accessing the network and the behavior of the systems to control change from NCS to NCS, and affect how optimal protocols should act to optimize the transmission rate. This makes RL a good strategy to develop protocols for NCSs, given that it can adapt to new environments without needing to know them. The optimal policies for an NCS in a determined context may not always be easy to find, but a RL algorithm provided with the right feedback mechanism will learn them, optimizing the transmission rate, which may help ensuring the stability of the system being controlled.

As explained previously, Q-learning, a RL technique, is usually used as a tabular method that, for every tuple of state and action \((s, a)\), returns the value of taking action \(a\) in state \(s\). Its use in NCSs is not advisable, as the potential dynamical behavior of the systems to control may cause a great variance in the values of the state, making the "action-state" table have huge proportions and the training of the algorithm to be unmanageable.

For that reason, in this thesis, we use Q-learning algorithms with two different approximations, neural networks [24] and linear parametrization [30]. They are applied to two different networked control systems architectures in order to generate data communication protocols that rule the transmission rate in those NCSs. These protocols are based on the redundant transmission property explained in [1]. It is
expected that, varying certain algorithm parameters before training (parameters that intend to “inform” the agent of the tradeoff between the success rate of the transmission and the usage of the communication medium), the algorithms generate protocols that regulate the number of redundant transmission to send at each time step. We expect that the protocols developed by both methods obtain, given the context, an optimal estimation error - cost of communication tradeoff for the NCSs considered.

The good results in [24] and [30] led us to believe that these would be two good Q-learning approximations to apply in the project of developing protocols for NCSs. The Deep Q Network algorithm of [24] has been proven to generalize and learn really well in complex problems [25], [27], [29], [26], and has shown good results when applied to the design of protocols in wireless networks [20], [29]. Despite the high level of computation that the training of Neural Networks require, they are really powerful approximators and are capable of generalize a great part of linear and non-linear functions. In this thesis, our algorithms based on the DQN algorithm were able to develop protocols that ruled the transmission rate of both NCSs, allowing them to have an optimal estimation error - cost of communication tradeoff for the contexts simulated.

The Zap Q-learning proposed in [30] presented as the main highlight its fast convergence, making it a light and computationally efficient learning algorithm. However, stochastic approximation algorithms tend to not be robust to high valued variances. That, and the fact that applications of the algorithm have not yet been published, make its appliance to the NCSs studied a “wild card”. In fact, our algorithms based on the Zap Q-learning were much faster than the Q-learning algorithms with neural networks, and, for NCS₁, had good results developing communication protocols with an optimal estimation error - cost of communication tradeoff for different contexts. However, for NCS₂, the features proposed for the Q-learning approximation were not able to fully approximate the optimal Q function, and some of the protocols obtained were not able to reduce the estimation error to its optimal value in the NCS.

1.4 Thesis Outline

This report is divided in five chapters. The first one presents a brief introduction and the objectives of this thesis. The second chapter contains a bibliographical revision on networked control systems (NCS), on the paradigms of learning, including reinforcement learning, on neural networks and on stochastic approximation. In the third chapter we present the methodology, which is the use of the Q-learning algorithms to compute communication protocols in the NCSs studied in this thesis. Chapter four has the results to the application of the algorithms explained in methodology to NCS₁ and NCS₂. Finally, chapter five contains the conclusions of this thesis, as well as the future work.
Chapter 2

Theoretical Background

This chapter provides the theoretical background and a bibliographical review on the most important topics to contextualize this thesis. Section 2.1 explains the concepts of Networked Control Systems and Data Communication Protocols, section 2.2 introduces the learning paradigm used in this thesis, Reinforcement Learning, and sections 2.3 and 2.4 aim to explain the function approximation techniques used to approximate the $Q$-function derived from the $Q$-learning algorithm.

2.1 Networked Control Systems

2.1.1 Introduction

A control system is a device or set of devices to manage, command, direct or regulate the behavior of other devices or systems. In engineering and mathematics, control theory deals with the behavior of dynamical systems. For many years, researchers have given us precise and optimal control strategies emerging from classical control theory. However, classical control systems connect their components using cables, which causes inconveniences, like the restriction of the components to be near each other, and the increasing complexity of the control system design as more components are added.

The advent of communication networks, however, introduced the possibility of remotely controlling a system, which gave birth to networked control systems (NCSs), that can be defined as a feedback control systems wherein the control loops are closed through a real-time communication network [3].

In the typical structure of NCSs, all components including sensors, controllers, actuators, and systems to be controlled are physically separated, spatially distributed and can transmit information to each other through a shared digital communication network [38]. Among system components, the information is transmitted in the form of successive data packets.

2.1.2 Categories and Components

According to [3], there are two general approaches to design NCSs. The first approach is to have several subsystems (composed by sensor, actuator and controller), where the controllers of these subsystems and a central controller $CM$ are connected through a network, forming a hierarchical structure, as depicted in Figure 2.1a. The central controller sends a set point to each of the subsystem controllers. Each one of the subsystems try to satisfy the set point by themselves. The sensor measurements of the subsystems are then sent to the central controller through the network.

The second approach of networked control is the direct structure, as shown in Figure 2.1b. This structure has a sensor and an actuator of a control loop connected directly to a network. In this case,
both components are attached to a plant, while a controller is separated from the plant by a network connection. In this work, there will be a special focus on the direct structure.

The NCS architectures studied in this thesis, $NCS_1$ (Fig. 2.2a) and $NCS_2$ (Fig. 2.2b), both follow the direct structure 2.1b. In $NCS_1$ the sensor is connected to the controller via a network. $NCS_2$ has two control systems, but there is no hierarchical structure nor central controller that rules both systems. The communication between the sensor and controller of each system is performed through a wireless network, and each controller autonomously controls the process of each plant.

Both the hierarchical and direct structures have their own pros and cons, and the use of each one of the structures is based on application requirements and designer’s preference. Many networked control systems are a hybrid of the two structures, where the subsystems in the hierarchical structure 2.1a are NCSs of the direct structure 2.1b and these are still controlled by a central controller which sets the point for all the controllers of the subsystems [3].

In an NCS, in order to function, whatever is the architecture and the components chosen, these components have to enable four functions: information acquisition (sensors), command (controllers), and communication and control (actuators). The defining feature of an NCS, and the main reason NCSs have been widely studied in the past times, is that control and feedback signals are exchanged among the systems components in the form of information packets through a network. NCS provide several advantages over point-to-point wired systems such as improvement in reliability through reduced volume of wiring, simpler systems integration, easier troubleshooting and maintenance, and the possibility for distributed processing [39].
2.1.3 NCS Challenges

The communication channel may be considered as the backbone of the networked control system. When choosing the communication or data transfer type, one gives preference to characteristics like reliability, security, ease of use and availability. However, the use of a communication network in the feedback control loop makes the analysis and design of an NCS complex.

Precise and optimum control strategies based on classical control theory, like PID control, optimal control, adaptive control, robust control and many other advanced forms of these control algorithms are not easily applied over a network, as one cannot assume synchronized control and non-delayed sensing and actuation. According to [11], this creates three issues: network-induced delay, loss of information packets and multiple-packet delay.

Network-induced delay (sensor-to-controller delay and controller-to-actuator delay) occurs while exchanging data among devices connected to the shared medium. This delay, either constant or time varying, can degrade the performance of control systems designed without considering the delay and can even destabilize the system.

There is loss of information on the transmission, as the network can be viewed as a web of unreliable transmission paths. Some packets not only suffer transmission delay but, even worse, can be lost during transmission. This usually happens when there are fading channels (communication channels in which the signal degrades over the distance) or packet collisions (happens when two nodes try to send a packet at the same time using the same channel). Normally, feedback-controlled plants can tolerate a certain amount of data loss, but it is important to determine whether the system is stable when only transmitting the packets at a certain rate and to compute acceptable lower bounds on the packet transmission rate.

Multiple-packet transmission happens when plant outputs have to be transmitted using multiple network packets, due to bandwidth and packet size constraints of the network. Sensor or actuator data are transmitted in separate network packets, and they may not arrive at the controller and plant simultaneously. There are two main reasons for multiple-packet transmission:

1. Packet-switched networks can only carry limited information in a single packet due to packet size constraints. Thus, large amounts of data must be broken into multiple packets to be transmitted.

2. Sensors and actuators in an NCS are often distributed over a large physical area, and it is impossible to put the data into one network packet.

2.1.4 NCS Solutions

According to the authors of [11], there are two main approaches for accommodating all of these issues in NCS design. One way is to treat the network protocol and traffic as given conditions and design control strategies that explicitly take the above-mentioned issues into account. The other approach is to design the control system without regard to the packet delay and loss but design a communication protocol that minimizes the likelihood of these events. In this thesis we will specially focus on the design of communication protocols.

2.1.4.1 Communication Protocol Design

Data Communication Protocols are a set of strategies that control and manage the way communication and data transfer occur between systems. The behavior of networked control systems is very much related to the way its nodes access and share the communication network. So, protocols are elaborated
with the objectives of managing and optimizing the way the data is transmitted in the network, without changing the structure of the controller [11].

In an NCS, only one node can access the network resource (channel) and be allowed to transmit data per transmission in order to a successful communication to take place. As such, various communication protocols have been introduced to provide different communication scheduling approaches among nodes [38].

The protocols of NCSs can be divided into two categories: static and dynamic protocols. In [13] has been used a static protocol, \textit{Round-Robin}, that is a periodic protocol in which the transmission sequence of nodes is fixed by a predetermined order. It handles all transmission requests in a circular first-in-first-out (FIFO) order and gives priority to none, so that all nodes may be able to use the same communication resources in the same amount of time and also have the same amount of waiting time each cycle. If the packet is not transmitted within the allotted time, it is preempted and then moved at the back of the line so that the next process in line is able to use the communication channel for the same amount of time. Static protocols, like Round-Robin, are simple to implement. However, they do not adapt the transmission rate accordingly to the state of the system to control, which makes them generally inferior to dynamic protocols for most NCSs.

There has been a greater focus in dynamic protocols, as it has been obtained better results with them. There were proposed several strategies in order to decrease the packets drop rate in NCSs. In [12] has been developed a dynamic protocol, \textit{Try-Once-Discard}, in which the opportunity to transmit data is only assigned to the node with the largest weighted estimation error (difference between the current and the most recently transmitted value of node’s signals). This protocol is mainly focused on assuring the stability of the NCSs, but not in the adaptive allocation of communication resources as a function of the estimation error - cost of communication relation. It also imposes a centralized control that analyzes the weighted error of all nodes. For the multi-agent NCS \textsubscript{2} (Figure 2.2b), the approaches in this thesis aim to generate protocols that makes the agents cooperate in sharing the communication resources, without the need for a centralized control of all agents errors.

In [29], the Deep Q-Network algorithm from [24] is used to dynamically allocate wireless communication channels in Cyber-Physical Systems (CPSs). The approach used in [29] is similar to the one used in this thesis, as, by definition, CPSs are systems built through integration of sensors, communication networks, controllers, dynamic (physical) processes and actuators (NCSs can be considered a particular case of CPSs). The problem studied is the following: there are \( N \) sensors, each one measuring a different process, that communicate through \( M \) wireless channels to a central controller. Given that there are fewer communication resources than sensors (\( M << N \)), it is used the DQN algorithm to find the optimal allocation policy that minimizes, at every time step, the sum of the mean squared estimation error of all processes. However, opposite to our approach, in [29] they use centralized control that receives the state of all processes, and, given that \( M << N \), it is expected that, at every time step, all channels are used, so there is no concern with the communication cost.

In [40] has been presented a controller and a protocol co-design method for the stabilization of NCSs with communication restrictions and random packet loss. The focus of [40] was also ensuring the stability of NCSs, but lacked to introduce the relation between the estimation error and the cost of communication and its importance to the allocation of communication resources to the different nodes. In this thesis, the protocols are developed to work with the NCS existing controllers, and it does not require the design of a controller.

In [21], has been studied the cooperation between agents for data dissemination (sensors forwarding data packets to other sensors) in Wireless Sensor Networks, in order improve the network performance. Despite not being applied to NCSs, this method shows similarities to the approach used in this thesis, as it uses Q-learning to develop adaptive communication protocols. However, it uses a tabular version
of the Q-learning algorithm which, as explained previously, is not efficient applied to NCSs. In [1] it is studied the redundant send of information packets, which will be used, along with reinforcement learning techniques, to develop new communication protocols, as explained previously.

Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) are two of the most studied and used protocols in the industry. They are part of the Internet protocol suite (TCP/IP), which is the conceptual model and set of communication protocols used on the Internet and similar computer networks. The TCP/IP decides how data should be packeted, addressed, transmitted, routed and received [2]. TCP and UDP are protocols of the transport layer of the TCP/IP. This layer aims to ensure that messages are delivered error-free, in sequence, and with no losses and duplication. The transport layer is responsible for data communication between other layers of the TCP/IP, and provides message traffic control [2].

UDP is a communication protocol used primarily for establishing low-latency and loss-tolerating connections between applications on the Internet. It has no mechanism to estimate if the packets were delivered, nor to retransmit if there was loss. This causes a very high drop rate. However, given that it does not retransmit packets, it is an ideal protocol for network applications in which time-delays are not admissible, such as gaming and voice and video communication, which can suffer some data loss without affecting perceived quality [41].

TCP is a protocol that defines how to establish and maintain a network conversation via which application programs can exchange data. It determines how to break applications data into packets that networks can deliver, sends packets to and accepts packets from the network layer, manages flow control, and - because it is meant to provide error-free data transmissions - handles retransmission of dropped packets as well as acknowledgement of all packets that arrive [42].

Despite being widely used, UDP and TCP are not a good fit as protocols for NCSs. UDP makes no attempt at error correction, which causes high packet drops in the network, leading to poor performance and eventually instability in the systems to control. In TCP, transmissions and the need to reorder packets after they arrive can introduce time-delays in the transmissions. In an NCS, while some packet drops may not cause instability in the system to control, delayed transmissions are essentially useless, and NCSs are highly sensitive to time-delays.

2.2 Learning

An agent is learning if it improves its performance on future tasks after making observations about the world [22]. A learning agent has a performance element that decides which actions to take, and a learning element that can change the performance element to be more efficient as the agents learns. A critic component is used to evaluate how well the agent is doing and provide feedback to the learning component, and a problem generator that can deviate from the usual routine and explore new possibilities.

In this thesis, the goal is that the NCS agents learn protocols that optimize the transmission rate, by gaining experience interacting with the world.

2.2.1 Learning Paradigms

According to [22] and [4], there are three main types of learning:

1. Supervised Learning, also called learning with a teacher - The agent observes some example input–output pairs and learns a function that maps from input to output. In this type of learning there is a teacher who has knowledge of the environment and the training is made by minimizing the responses of the learning system (which can be, for example, a neural network) and of the
teacher to the input signals. The main idea is to make the learning system imitate the teacher. This process is illustrated in Figure 2.3

![Supervised Learning Diagram](image)

Figure 2.3: Supervised Learning diagram - source: [4]

2. **Unsupervised Learning** - The agent learns patterns in the input even though no explicit feedback is supplied, as there is no external teacher or critic to oversee the learning process. The learning system must develop its own representation to the entry signals. The most common unsupervised learning task is clustering: detecting potentially useful clusters of input examples. The unsupervised learning diagram is in Figure 2.4

![Unsupervised Learning Diagram](image)

Figure 2.4: Unsupervised Learning diagram - source: [4]

3. **Reinforcement Learning** - The learning of an input-output mapping is performed through continued interaction with the environment in order to minimize a scalar index of performance. The goal of learning is to minimize a cost-to-go function, defined as the expectation of the cumulative cost of actions taken over a sequence of steps instead of simply the immediate cost. It may turn out that certain actions taken earlier in that sequence of time steps are in fact the best determinants of overall system behavior. The function of the agent is to discover these actions and feed them back to the environment. The learning process is represented in Figure 2.5

![Reinforcement Learning Diagram](image)

Figure 2.5: Reinforcement Learning diagram - source: [4]

In this thesis, reinforcement learning will be used, allied with dynamic programming. Both topics will be described next.
### 2.2.2 Reinforcement Learning

#### 2.2.2.1 Introduction

Reinforcement Learning (RL) is an area of machine learning inspired by behaviorist psychology, concerned with how software agents ought to take actions in an environment, usually represented by a Markov Decision Process (MDP), so as to maximize some notion of cumulative reward.

A MDP is a sequential decision problem for a fully observable, stochastic environment with additive rewards. It assumes the Markov Property: the effects of an action taken in a state depend only on that state and not on the prior history. A MDP model consists of a set of states $S$ (with an initial state $s_0$), a set $A$ of actions in each state, a transition model $P(s' | s, a)$ which defines the probability of reaching state $s'$ by taking action $a$ in state $s$, and a reward function $R(s)$ that returns the reward value of being in state $s$.

In some literatures, Reinforcement Learning is assumed as a variant of Unsupervised Learning, given that agents learn what to do in the absence of labeled examples of what to do, only using a reinforcement system (rewards or punishments) imposed by the user.

In the absence of feedback from a teacher, an agent can learn a transition model for its own moves, but without some feedback about what is good and what is bad, the agent will have no grounds for deciding which move to make. For example, in a game of chess, the agent needs some kind of feedback to know that something good happened when it checkmated the opponent, and that something bad happened when the agent got checkmated. This kind of feedback is called reward, or reinforcement.

According to [17], there are three important distinguishing features of RL problems:

1. **Being closed-loop** - the learning system's actions influence its later inputs;

2. **Not having direct instructions as to what actions to take** - the learner is not told which actions to take, as in many forms of machine learning, but instead must discover which actions yield the most reward by trying them out;

3. **The consequences of actions, including reward signals, play out over extended time periods** - In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards.

To face these problems, a RL agent has to: have **explicit goals**, be able to **sense aspects of the environment** and **choose actions to influence the environment**. It is usually assumed that the agent has to operate despite significant uncertainty about the environment it faces.
Reinforcement learning handles the difficult problem of correlating immediate actions with the delayed returns they produce. Like humans, reinforcement learning algorithms sometimes have to wait a while to see the fruit of their decisions. They operate in a delayed return environment, where it can be difficult to understand which action leads to which outcome over many time steps.

The authors of [22] give a good analogy to explain RL: "Imagine playing a new game whose rules you don’t know; after a hundred or so moves, your opponent announces, ‘You lose.’ This is reinforcement learning in a nutshell.”

2.2.2.2 Exploration and Exploitation

Given that the environment is unknown to the agent, the action with the best accumulated reward at a time instant may not actually be the best action. Thus, a greedy agent (one that always choose the best action) can lead to suboptimal results, as the learned model is not the same as the true environment - what is optimal in the learned model can therefore be suboptimal in the true environment [22].

In RL, actions do more than provide rewards according to the current learned model, they also contribute to learn the true model by affecting the percepts that are received. So, the agent must make a tradeoff between exploitation - choosing the action that maximizes the reward, and exploration - choosing a random action in order to explore the environment.

This is not a trivial problem, as pure exploitation may make the agent to get stuck in suboptimal policies, while pure exploration to improve the agent’s knowledge is of no use if one never puts that knowledge into practice. The exploration-exploitation dilemma has been intensively studied by mathematicians for many decades. In [24], it is used an algorithm that decays the exploration rate (probability that the agent will choose a random action in order to explore the environment) over the training of the learning system (in this case, a neural network).

2.2.2.3 Dynamic Programming

Dynamic programming (DP) is a technique that deals with situations where decisions are made in stages, with the outcome of each decision being predictable to some extent before the next decision is made. A key aspect of such situations is that decisions cannot be made in isolation. Rather, the desire for a low cost at the present must be balanced against the undesirability of high costs in the future [4].

The term dynamic programming refers to a a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment represented as a Markov decision process. The key idea of DP, and reinforcement learning generally, is the use of value functions to organize and structure the search for good policies [17].

A policy, denoted by \( \pi(s,a) \), describes a way of acting. It is a function that takes as input a state and an action and returns the probability of taking action \( a \) in state \( s \). The goal of RL is to learn a optimal policy, \( \pi^* \), which tells the agent how to act in order to maximize reward in every state.

To learn the optimal policy, value functions are used. There are two types of value functions used in reinforcement learning: the value function, denoted \( V(s) \), and the action-value function, or Q function, denoted \( Q(s,a) \) [43]. The second one will be used in this thesis.

The value function, represented in 2.1, describes the value of a state when following a policy. It is the expected reward when starting from state \( s \) acting according to policy \( \pi \):

\[
V^\pi(s) = E_{\pi}[R_t | s_t = s]
\] (2.1)
where \( R_t \) is the future cumulative discounted reward:

\[
R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}
\]  

(2.2)

where \( \gamma \) is the discount rate, \( 0 < \gamma < 1 \).

The two main benefits of writing the reward as in (2.2) is that the reward is well defined for infinite series, and it gives greater weights to imminent rewards than to future rewards. The smaller the value of \( \gamma \), the more greedy the algorithm becomes - gives more importance to imminent rewards than to future ones [43]. From (2.1) one can derive the Bellman equation for the value function:

\[
V^\pi(s) = E_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s]
\]  

(2.3)

The action value function, represented in (2.4) tells us the value of taking an action \( a \) in state \( s \) when following a certain policy \( \pi \). It is the expected reward given the state and action under the policy \( \pi \):

\[
Q^\pi(s, a) = E_\pi[r_{t+1} + \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s, a_t = a]
\]  

(2.4)

From (2.2) and (2.4) one can derive the Bellman equation for the action-value function:

\[
Q^\pi(s, a) = R^a_{ss'} + \gamma E_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_{t+1} = s']
\]  

(2.5)

If we pull out the first reward from the sum, we can rewrite it like so:

\[
Q^\pi(s, a) = R^a_{ss'} + \gamma \left( V^\pi(s') + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_{t+1} = s' \right)
\]  

(2.6)

The expectation in (2.6) describes what is the expected reward if from state \( s \) we continue to follow policy \( \pi \), so it can be written explicitly by summing all possible returned states:

\[
R^a_{ss'} = E[r_{t+1} | s_t = s, s_{t+1} = s', a_t = a]
\]

\[
E_\pi[\gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s, a_t = a] = \gamma E_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_{t+1} = s']
\]  

(2.7)

where \( R^a_{ss'} \) is the expected reward when starting in state \( s \), taking action \( a \), and moving into state \( s' \).

By substituting the expectations in (2.6) we get:

\[
Q^\pi(s, a) = R^a_{ss'} + \gamma \left( V^\pi(s') + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_{t+1} = s' \right)
\]  

(2.8)

Note that (2.3) is in the same form as the end of this equation. We can therefore write (2.8) as:

\[
Q^\pi(s, a) = R^a_{ss'} + \gamma V^\pi(s') + \gamma \max_{a'} Q(s', a')
\]  

(2.9)

The value of a state \( s \) is equal to the maximum Q value for that state [22]:

\[
V^\pi(s) = \max_a Q^\pi(s, a).
\]

Substituting in (2.9)

\[
Q^\pi(s, a) = R^a_{ss'} + \gamma \max_{a'} Q(s', a')
\]  

(10.20)

where \( s' \) is the state we get when we take action \( a \) in state \( s \). An optimal policy \( \pi^* \) is the policy that yields
the highest expected value \[22\]. Denoting \( \pi_s^* \) as the optimal policy starting in state \( s \):

\[
\pi_s^* = \arg\max_{\pi} V^\pi(s) \\
= \arg\max_{\pi} \left( \max_a Q^\pi(s,a) \right) \\
= \arg\max_a Q^\pi(s,a)
\] (2.11)

This type of learning is called \textbf{Q-learning}, and is a RL technique that does not require a model of the environment. For any finite MDPs, Q-learning eventually finds an optimal policy, in the sense that the expected value of the total reward return over all successive steps, starting from the current state, is the maximum achievable \[22\].

\[2.2.2.4\] Generalization

It was previously assumed that the action-value function, or Q function, learned by the agents is represented in a tabular form with one output value (\( Q \)) for each input tuple (\( s, a \)). Such approach works reasonably well for small state spaces, but the time to converge and the time per iteration increases rapidly as the state space gets larger \[22\].

Besides the memory needed for large problems, there is also a problem with the lack of time and data needed to fill them accurately. In other words, the key issue is that of \textbf{generalization}. How can experience with a limited subset of the state space be usefully generalized to produce good approximation over a much larger subset?

In many tasks to which we would like to apply reinforcement learning, most states encountered will never have been experienced exactly before. This will almost always be the case when the state or action spaces include continuous variables. The only way to learn anything at all on these tasks is to generalize from previously experienced states to ones that have never been seen \[17\].

The kind of generalization required is often called \textbf{function approximation} because it takes examples from a desired function (e.g., a action-value function) and attempts to generalize from them to construct an approximation of the entire function. In reinforcement learning it is important that learning be able to occur online, while interacting with the environment. This requires methods that are able to learn efficiently from incrementally acquired data \[17\].

\textbf{Gradient-descent} methods are among the most widely used of all function approximation methods and are particularly well suited to online reinforcement learning. Two methods for gradient-based function approximation have been used widely in RL. One is \textbf{multilayer artificial neural networks} using the error backpropagation algorithm, in which is included \textbf{neuro-dynamic programming}. The second popular form is the \textbf{linear parametrization} \[17\].

\textbf{Neuro-dynamic programming}, that combines the mathematical formalism of classical dynamic programming and the learning capability of neural networks, offers a powerful approach to approximate the optimal Q function. Traditional approaches to dynamic programming are hardly applicable to problems of this kind because of the enormous size of the state space that would have to be explored \[4\].

\textbf{Linear parametrization} seeks to approximate the action-value function as a linear function of the parameter vector \( \theta \) and the features vector \( \phi \) \[17\]. Both methods are explained and used in this dissertation.
2.3 Neural Networks

2.3.1 Introduction

An Artificial Neural Network (ANN) can be described as a massively parallel distributed processor made up of simple processing units (neurons), which has a natural propensity for storing experiential knowledge and making it available for use [4]. It resembles the brain in two aspects:

1. Knowledge is acquired by the network from its environment through a learning process.
2. Inter-neuron connection strengths, known as synaptic weights, are used to store the acquired knowledge.

Training of these networks is normally performed in a supervised manner. One assumes that a training set is available, which contains both input patterns and the corresponding desired target patterns. The training is normally based on the minimization of some error measure between the network’s outputs and the desired outputs.

An ANN derives its computing power through its massively parallel distributed structure and its ability to learn and generalize, i.e., produce reasonable outputs for inputs not encountered during training. These two information-processing capabilities make it possible for neural networks to solve complex problems.

Using neural networks has several benefits [5]:

- **Nonlinearity** - Neural networks have the capability to generalize either linear and/or nonlinear functions
- **Input-Output Mapping** - As explained before, a neural network can realize supervised learning, as it is able to deduce the function that maps the inputs into outputs.
- **Adaptivity** - ANNs have a built-in capability to adapt their synaptic weights to changes in the surrounding environment. In particular, a neural network trained to operate in a specific environment can be easily retrained to deal with minor changes in the operating environmental conditions.

2.3.2 Network Architectures and Properties

Neural networks are typically formed by units of the type shown in Figure 2.6. Each of these units forms a weighted sum of its inputs, to which a constant term is added. This sum is then passed through a nonlinearity which is often called its activation function. The nonlinearities in the network’s units can be any differentiable functions. This property is desirable for enabling gradient-based optimization methods, like the error backpropagation algorithm [5]. The activation functions more commonly used are sigmoids, curves with two horizontal asymptotes, monotonically increasing, with a single point where the curvature change signs, like the one represented in Figure 2.6. Other examples are the rectified linear unit (ReLU) and linear functions, which are used in this thesis and are explained in section 3.4.1.

Interconnections between units have weights, which multiply the values that go through them. Besides the variable inputs that come through weighted links, units normally have a fixed input, which is often called bias. It is through the variation of the weights and bias that networks are trained to perform the operations that are desired from them [5]. A neural network is just a collection of units connected together, the properties of the network are determined by its topology and the properties of the neurons (weights, bias and nonlinearity) [22].
There are two distinct network architectures, **feed-forward** and **recurrent**. A feed-forward network has connections only in one direction - that is, it forms a directed acyclic graph. Every node receives input from "upstream" nodes and delivers output to "downstream" nodes, i.e., there are no loops [22].

A recurrent network, on the other hand, feeds its outputs back into its own inputs. This means that the activation levels of the network form a dynamical system that may reach a stable state or exhibit oscillations or even chaotic behavior. Moreover, the response of the network to a given input depends on its initial state, which may depend on previous inputs. Hence, recurrent networks (unlike feed-forward networks) can support short-term memory. This makes them more interesting as models of the brain, but also more difficult to understand [22]. In this thesis, the neural networks used have a feed-forward architecture. In Figure 2.7 are examples of both network architectures.

![Feed-forward network](image1)

![Recurrent network](image2)

**Figure 2.7: Network architectures - source [5]**

### 2.4 Stochastic Approximation

#### 2.4.1 Introduction

**Stochastic approximation** (SA) algorithms are recursive update rules that can be used to solve optimization problems and fixed point equations (including standard linear systems) when the collected data is subject to noise. Stochastic approximation methods attempt to find roots or extrema of functions which cannot be computed directly, but only estimated via noisy observations. This situation is common, for instance, when taking noisy measurements of empirical data, or when computing parameters of a statistical model [44].

Mathematically, the goal of these algorithms is to find the solution $\theta^*$ to $\hat{f}(\theta^*) = 0$ for a function $\hat{f} : \mathbb{R}^n \to \mathbb{R}^n$. SA is itself an approximation of successive approximations [45]. It is assumed that
\( \hat{f}(\theta) = E[f(\theta, \Phi)] \), where \( f : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^n \) and \( \Phi \) is a random variable with unique marginal distribution. Substituting, we obtain in (2.12) the goal of stochastic approximation:

\[
\hat{f}(\theta^*) = E[f(\theta, \Phi)]|_{\theta=\theta^*} = 0 \tag{2.12}
\]

which is the expected value of a function depending on a random variable \( \Phi \). The algorithm uses random samples of \( f(\theta, \Phi) \) to efficiently approximate properties of \( \hat{f} \) such as zeros or extrema [46].

Finding this solution is often difficult, and according to [46] there are three main reasons for it:

1. The function \( f \) and the distribution of the random vector \( \Phi \) may not be known - there may only be knowledge about the structure of the problem;
2. Even if everything is known, computation of the expectation may be expensive. For root finding, it is needed the computation of the expectation for many values of \( \theta \);
3. The recursive algorithms are often slow, and their variance may be infinite.

The standard stochastic approximation algorithm is defined by:

\[
\theta_{k+1} = \theta_k + \alpha_k f(\theta_k, \Phi_{k+1}), \quad k \geq 0 \tag{2.13}
\]

It is always assumed that the scalar gain sequence \( \{\alpha_n\} \) is non-negative, and satisfies:

\[
\sum \alpha_k = \infty, \quad \sum \alpha_k^2 < \infty \tag{2.14}
\]

Expression (2.13) converges under general conditions. However, its rate of convergence can often be improved dramatically through the introduction of a matrix gain [45].

### 2.4.2 Stochastic Gradient Descent

Gradient-descent methods are amongst the most widely used of all function approximation methods and are particularly well suited to online reinforcement learning. In gradient-descent methods, the parameter vector is a column vector with a fixed number of real valued components, \( \theta=(\theta_1, \theta_2, ..., \theta_n)^T \), and the approximate action-value function \( Q(s,a,\theta) \) is a smooth differentiable function of \( \theta \) [17].

Gradient-descent methods try to minimize the error between the Q-function \( Q(s,a) \) and the approximated Q-function \( Q(s,a,\theta) \) on the observed examples by adjusting the parameter vector after each example by a small amount in the direction that would most reduce the error on that example:

\[
\theta_{k+1} = \theta_k - \frac{1}{2} \alpha \nabla Q(s_k, a_k) - Q(s_k, a_k, \theta_k) \nabla Q(s_k, a_k, \theta_k) \tag{2.15}
\]

where \( \alpha \) is a positive step-size parameter, and \( \nabla Q(s_k, a_k, \theta_k) \) denotes the vector of partial derivatives with respect to the components of the weight vector:

\[
\nabla Q(s_k, a_k, \theta_k) = \left( \frac{\partial Q(s_k, a_k, \theta_k)}{\partial \theta_{k,1}}, \frac{\partial Q(s_k, a_k, \theta_k)}{\partial \theta_{k,2}}, ..., \frac{\partial Q(s_k, a_k, \theta_k)}{\partial \theta_{k,n}} \right)^T \tag{2.16}
\]

This derivative vector is the gradient of \( Q(s_k, a_k, \theta_k) \) with respect to \( \theta_k \). This kind of method is called gradient descent because the overall step in \( \theta_k \) is proportional to the negative gradient of the example’s squared error, which is the direction on which the error falls most rapidly [17].
2.4.3 Linear Parametrization

One of the most important cases of gradient-descent function approximation is that in which the approximate function, \( Q(s, a, \theta) \), is a linear function of the parameter vector, \( \theta \). Corresponding to every state \( s \) and action \( a \), there is a vector of features \( \psi^a(s) = (\psi^a_1(s), \psi^a_2(s), ..., \psi^a_n(s))^T \), with the same number of components of \( \theta \). The approximate action-value function is given by the dot product between the transpose of the parameter vector and the features vector, as is shown in equation 2.17.

\[
Q(s, a, \theta) = \theta^T \psi^a(s) = \sum_{i=1}^{n} \theta_i \psi^a_i(s) \tag{2.17}
\]

In this case the approximate action-value function is said to be **linear in the parameters** [17].

It is natural to use gradient-descent updates with linear function approximation. The gradient of the approximate action-value function with respect to \( \theta \) in this case is given by equation 2.18.

\[
\nabla Q(s, a, \theta) = \psi^a(s) \tag{2.18}
\]

The general gradient-descent update 2.15 is reduced to a particularly simple form in the linear case:

\[
\theta_{k+1} = \theta_k + \alpha_k (Q(s_k, a_k) - Q(s_k, a_k, \theta_k)) \psi^a(s_k) \tag{2.19}
\]

In addition, in the linear case there is only one optimum \( \theta^* \) that minimizes the error \( Q(s, a) - Q(s, a, \theta) \). Thus, any method guaranteed to converge to or near a local optimum is automatically guaranteed to converge to or near the global optimum [17].

2.4.3.1 Features

Linear learning methods are also of interest because in practice they can be very efficient in terms of computation. Whether or not this is so depends critically on how the states are represented in terms of the **features**. Choosing features appropriate to the task is an important way of adding prior domain knowledge to reinforcement learning systems. Intuitively, the features should correspond to the natural features of the task, those along which generalization is most appropriate [17].

In general, there also has to exist features for combinations of these natural qualities. This is because the linear form prohibits the representation of interactions between features, such as the presence of feature \( i \) being good only in presence of feature \( j \). In cases with such interactions one needs to introduce features for conjunctions of feature values when using linear function approximation methods [17].

2.4.4 Stochastic Newton-Raphson

According to [37], there are two well known Stochastic Approximation techniques that are known to have optimal rate of convergence (measured in terms of asymptotic variance): the **Stochastic Newton-Raphson (SNR)** algorithm (a matrix gain algorithm that resembles the deterministic Newton-Raphson method), and the Ruppert-Polyak averaging technique. The **Zap Q-learning** ([30]), on which the algorithms from section 3.4.2 were based on, is itself based on a variant of SNR, designed to more closely mimic its deterministic cousin. For this reason it will be explained in detail.

The SNR introduces a matrix-gain to the standard stochastic approximation step 2.13.

\[
\begin{align*}
\theta_{k+1} &= \theta_k - \alpha_{k+1} A_{k+1}^{-1} [A_{k+1} \theta_k - b_{k+1} ] \\
A_{k+1} &= A_k + \alpha_{k+1} [A_{k+1} - A_k], \quad \alpha_{k+1} = \frac{1}{k+1}
\end{align*}
\tag{2.20}
\]
where $A_{k+1} = A(\Phi_{k+1})$ is a $n \times n$ matrix, $b_{k+1} = b(\Phi_{k+1})$ is a $n \times 1$ vector, $\Phi_{k+1}$ is a random variable, $k \geq 0$, and $\theta_0$ and $A_1$ are initial conditions \[45\].

### 2.4.4.1 Zap Stochastic Newton-Raphson

This is a two time-scales algorithm with a higher step-size for the matrix recursion, and is defined by:

\[
\begin{align*}
\theta_{k+1} &= \theta_k - \alpha_{k+1} \hat{A}^{-1}_{k+1} [A_{k+1} \theta_k - b_{k+1}] \\
\hat{A}_{k+1} &= \hat{A}_k + \gamma_{k+1} [A_{k+1} - \hat{A}_k]
\end{align*}
\] (2.21)

It is different from the original SNR algorithm \[2.20\] because of the two time-scale construction \[45\]. The second step-size sequence $\{\gamma_k\}$ is non-negative, satisfies \[2.14\] and also

\[
\lim_{k \to \infty} \frac{\alpha_{k+1}}{\gamma_{k+1}} = 0
\] (2.22)

where $\alpha_{k+1} = \frac{1}{\gamma_{k+1}}$, $k \geq 0$. 

25
Chapter 3

Methodology

This chapter focuses on how the theory explained in chapter 2 is used to build the Q-learning algorithms applied to the different networked control systems. The dynamics of the NCSs are also explained, both the one-agent and two-agents systems. NCS1 and NCS2 were already displayed in section 2.1.2 and are again shown, for convenience, in Figures 3.3 and 3.5 respectively.

3.1 Networked Control Systems Dynamics

3.1.1 NCS with One Agent and One Channel

We consider a linear time-invariant plant with

\[
\begin{align*}
x(k + 1) &= Ax(k) + Bu(k) + w(k) \\
y(k) &= Cx(k)
\end{align*}
\]

where \(x \in \mathbb{R}^m\) denotes the state of the process, \(u \in \mathbb{R}^n\) the control input, \(y \in \mathbb{R}^p\) the output vector of the plant, \(w \in \mathbb{R}^m\) an \(m\)-dimensional zero-mean Gaussian white noise process with positive definite covariance matrix \(R_w\), \(A \in \mathbb{R}^{m \times m}\) is the state matrix, \(B \in \mathbb{R}^{m \times n}\) is the input matrix, and \(C \in \mathbb{R}^{m \times m}\) is the output matrix. It is assumed that the system is controllable.

Figure 3.1: NCS architecture considering only one channel

Considering the networked control system illustrated in Figure 3.1 it is assumed that the sensor does not fail and measures all the states \(x(k)\). On the other hand, the communication channel between the sensor and the controller is subject to communication failures. We admit that the transmitter is equipped
with a feedback channel that allows it to know which packets are dropped. Due to the nature of NCSs, data packets are typically small [1], and will be considered the basic unit of information in this thesis.

It is assumed that the communication channel is an erasure channel with independent and identically distributed (iid) packet drops. An erasure channel is a communication channel model where packets are either received or lost (the signal may degrade over the time, making it unrecognizable to the receiver, as mentioned in section 2.1.3). It can be modeled as a Bernoulli distribution, where the probability of transmission failure is \( p \), and the probability of transmission success is \( 1 - p \), \( p \in [0, 1] \). For every time step of the system, the sensor measures the state from the plant, and sends a packet to the controller through the network channel. The transmission is successful if the packet sent by the sensor reaches the controller and is recognized by it, and fails if the packet is lost.

Given these failures, there is a need to compute an estimate of the state, \( \hat{x}(k) \). Defining \( \delta_s \) as the set of instants when the communication between the sensor and the controller was successful (no failures):

\[
\hat{x}(k + 1) = \begin{cases} 
Ax(k) + Bu(k) & k \in \delta_s \\
A\hat{x}(k) + Bu(k) & k \notin \delta_s 
\end{cases}
\] (3.2)

For this process, the following control law was considered, using the estimated state \( \hat{x} \):

\[
u = -K\hat{x}
\] (3.3)

where \( K \in \mathbb{R}^{n} \times \mathbb{R}^{m} \) is the optimal gain matrix such that (3.3) minimizes the quadratic cost function:

\[
J(u) = E\left[ \sum_{k=1}^{\infty} \hat{x}(k)^TQ\hat{x}(k) + u(k)^TRu(k) \right]
\] (3.4)

where \( Q = I_{m \times m} \) and \( R = I_{n \times n} \) for the discrete-time state-space model in equation 3.2.

Subtracting 3.2 with 3.1 one can obtain the expression for the estimation error \( e(k) = \hat{x} - x \):

\[
e(k + 1) = \begin{cases} 
-w(k) & k \in \delta_s \\
Ae(k) - w(k) & k \notin \delta_s 
\end{cases}
\] (3.5)

The closed loop dynamics of the system can then be represented in terms of this error using:

\[
x(k + 1) = (A - BK)x(k) - BKe(k) + w(k)
\] (3.6)

In equation 3.1 we have a linear system driven by Gaussian white noise, and the optimal gain matrix \( K \) is obtained by minimizing the expectation of a quadratic cost criterion \( J(u) \). Then, we have a Linear-Quadratic-Gaussian (LQG) control problem. In this thesis, we apply feedback control using the Certainty Equivalence Principle, which states that the optimal control strategy in the case of incomplete information results from the optimal control strategy in the case of complete information through replacing of variables by its estimate [47], in this case, replacing the state \( x(k) \) by the estimated state \( \hat{x}(k) \). We can then assume that, the lower the estimation error \( e(k) \) is, the better the control performance of the system will be.

An example of the behavior of the NCS in Figure 3.1 was simulated, and is represented in Figure 3.2. This system has two state variables and is a possible implementation of 3.1 where the matrices are \( A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \), \( B = \begin{pmatrix} 0 \\ 0.2 \end{pmatrix} \), \( C = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \), and the covariance matrix of the white noise is \( R_w = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \). Also, the probability of communication failure is \( p = 0.3 \).

In the simulation represented in Figure 3.2 it is possible to observe the relation between both components of the estimation (prediction) error \( e(k) \) and the system output \( y(k) \). By equation 3.5, when the
packet delivery is successful, the prediction error is equivalent to the white noise $-w(k)$, and when there is a packet drop, both the prediction error and the system output grow by a factor $A$, the same from the model state equations 3.1. In Figure 3.2, the packet drops can be observed where the signal of the prediction error differs from the white noise.

To conclude, for an NCS with only one loop, the estimation error evolves as:

$$e(k+1) = \begin{cases}  
-w(k) & \text{with probability } 1-p \\
Ae(k) - w(k) & \text{with probability } p
\end{cases}$$

(3.7)

### 3.1.2 NCS with One Agent and Multiple Channels

In the situation where the networked control system is equipped with more than one communication channel, there is the possibility of redundant transmission [1], i.e., each packet can be transmitted, at each time step, on more than one channel, in order to increase the probability of success in the transmission. However, the use of multiple channels has the consequence of increasing the usage of the medium, which increases the cost of communication, given that more energy is used to transmit the redundant messages, and occupies more communication channels, which may interfere with other communication devices that may use the network resources. One can define the usage of the medium as the percentage of channels used at each time instant.

It is assumed that, by means of some diversity scheme, a number of independent redundant channels is available for data transmission. These are erasure channels with independent identically distributed
(iid) packet drops, similarly to section 3.1.1. Further, by means of an acknowledgement mechanism, the sensor knows which measurements were received by the controller.

At each time step, the sensor sends measurements to the controller with a certain level of redundancy, which can be defined as the percentage of channels used to transmit redundant packets over all the channels available for redundant transmission. For example, for the NCS of Figure 3.3, channel 1 is always used to transmit information between the sensor and the control, and channels 2 and 3 are available for redundant transmission. So, the level of redundancy is, at each time step:

- 0% - if only one packet is sent through channel 1;
- 50% - if two packets are transmitted using two channels;
- 100% - if three packets are transmitted using three channels.

The state equations used to define the system 3.1.1 can also be used to define an NCS with multiple channels like the one in Figure 3.3. The only difference lies with the probability of packet transmission failure. Considering \( n \) independent channels, equation 3.7 can be rewritten as:

\[
e(k+1) = \begin{cases} 
-w(k) & \text{with probability } 1 - \prod_{i=1}^{n} p_i \\
Ae(k) - w(k) & \text{with probability } \prod_{i=1}^{n} p_i 
\end{cases}
\]  

(3.8)

where \( p_i \) represents the probability of packet transmission failure for each one of the \( n \) independent channels.

\[ A = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 0.2 \\ 0.7 \end{pmatrix}, \quad C = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad R_w = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad p = 0.3. \]

As three channels are used at the same time instant, the probability of transmission failure is much smaller \( p = \prod_{i=1}^{3} 0.3 = 0.027 \). This can be observed in Figure 3.4, as the error is generally coincident with the white noise of the system, which by equation 3.8 represents success in transmission.

A simulation of the NCS transmitting in all of the three independent channels is represented in Figure 3.4. The state space matrices, the noise covariance matrix and the probability of failure are the same as in simulation presented in Figure 3.2. As three channels are used at the same time instant, the probability of transmission failure is much smaller \( p = \prod_{i=1}^{3} 0.3 = 0.027 \). This can be observed in Figure 3.4, as the error is generally coincident with the white noise of the system, which by equation 3.8 represents success in transmission.

### 3.1.3 NCS with Multiple Agents and Multiple Channels

When one considers several agents sharing communication channels, the usage of the medium is a critical indicator, as there is the possibility of a single process taking over all the channels, and the other processes become unstable due to the lack of communication feedback.
The NCS with multiple agents of Figure 3.5, similarly to the NCS of section 3.1.2, has three network channels that are erasure channels with independent identically distributed packet drops. It has two identical plants and control systems. The communication between the sensor and the controller of each system is made through a network, and both systems share the same network. It is composed by three erasure channels, each one with probability of packet dropping of $p$. Sensor 1 always use uses channel 1 to transmit to controller 1, sensor 2 always uses channel 2 to transmit to controller 2, and channel 3 is used by both for redundant transmissions. Again, the sensors know which measurements were received by the respective controller.

The NCS with multiple agents introduces a new and relevant problem, the competition for the communication medium. Following a stochastic approach, the problem of having multiple agents competing for the same network is not trivial, and it can be of difficult solution.

If the estimation error of both agents is low enough to ensure the stability of the systems to control, the solution is simple: the agents should use the fewer communication channels possible, thus transmitting only in the respective channels, and not in the shared channel. However, if the estimation error of the agents is high and they need to use the shared channel to send redundant transmissions, the problem becomes more complicated. As both agents transmit more frequently through channel 3, their estimation error decreases, but the packet collisions increase, resulting on the loss of information shared through that channel, which causes the estimation error to increase. This way, the goal is that both agents learn to cooperate and to adjust their transmission rate in order to optimize their estimation error - channels used tradeoff.

In order to develop protocols that can coordinate the communication in such NCSs, two goals must be achieved:

1. Guarantee the stability of the several control loops;
2. Obtain a good cost relation between the control distortion (low prediction error) and the usage of the medium (low communication cost, proportional to the usage of communication channels).

In this thesis we will explore protocols that aim to find a optimal tradeoff between the distortion rate and the communication costs. However, it has to be assured that the agents use the communication resources with the same frequency. The simulation of the NCS in Figure 3.5 is represented in Figure 3.6. The state-space model is the same from equation 3.1 and the state-space matrices are the same used in sections 3.1.1 and 3.1.2.

In this simulation, it was assumed that agent 1 always transmitted in channel 1, agent 2 always transmitted in channel 2, and both share channel 3. The probability of transmission failure for each of the three channels is $p = 0.3$. The probability of each agent transmitting in the shared channel was set high, $p_1 = p_2 = 0.75$, in order to observe the effect of collisions in the prediction error of both agents.

Figure 3.5: $NCS_2$ - NCS with two agents and three channels
Collisions happen when both agents try to communicate using channel 3. It was assumed in this thesis that a collision happens when, at the same time step, both agents decide to transmit in the shared channel, despite if any packet was dropped. In this case, both the agents’ information is lost, so the collisions number is an important factor that affect the estimation (prediction) error. This behavior is shown on purpose in Figure 3.6. There have happened 540 collisions in 1000 iterations, that affected the systems performance, as both error components of each agent have high value peaks that differentiate from the white noise that affects the system.

3.2 Cost Function

As explained in section 2.2.2, reinforcement learning agents need some feedback about what is good or bad, in order to decide what actions to do. This can be achieved by imposing a cost function that returns a value depending on how the agent interacts with the environment.

Obtaining protocols that have a good tradeoff between the transmission rate and the error becomes a complex task as the quantity of agents using the same channel increases. This way, the problem is now choosing which agents will transmit and in what channels they will transmit. The objective is to obtain a strategy/protocol that chooses the best actions to be taken, considering the present and future cost of them.

Mathematically, the cost function was defined as the weighted sum between the temporal mean of the quadratic estimation error (or mean squared error) $r_{err}$ and the temporal mean of the usage of the medium $r_{com}$ [7]. Given an initial state $s_0$ and a protocol $\pi$, the cost function $J_{\pi}(s_0)$ can be defined...
as:

\[ J_\pi(s_0) = \alpha \cdot r^\pi_{\text{err}}(s_0) + \beta \cdot r^\pi_{\text{com}}(s_0) \]  

(3.9)

where \( \alpha \in [0, 1] \), \( \beta \in [0, 1] \), \( \alpha + \beta = 1 \), are weight factors that determine the magnitude of the effect of \( r^\pi_{\text{err}} \) and \( r^\pi_{\text{com}} \) in the cost function \( J_\pi(s_0) \).

For each instant \( k \), \( e(k) \) is the estimation error and \( v(k) \) is the number of channels used. It is computed the expected value, given the existence of several random variables, from which we do not know the future values, as the white noise that affects the system and the process of losing communication packets. Also, for NCS \( S_2 \), the actions chosen by the protocol are a random variable, as they only define the probabilities of the agents using the shared communication channel.

\[ r^\pi_{\text{err}}(s_0) = E^\pi_{s_0} \left[ \sum_{k=0}^{\infty} e(k)^t e(k) \gamma^k \right] \]

\[ r^\pi_{\text{com}}(s_0) = E^\pi_{s_0} \left[ \sum_{k=0}^{\infty} v(k) \gamma^k \right] \]  

(3.10)

where \( \gamma \in [0, 1] \) is a discount factor and \( E^\pi_{s_0} \) denotes the expected value given a protocol \( \pi \) and an initial state \( s_0 \). In equation 3.9 is established the dependence between the choice of a protocol and the values of the future cost.

According to the Bellman equation (section 2.2.2.3), the minimal cost function can be given by:

\[ J^*(s, a) = \min_a E \{ c(s, a) + \gamma J^*(s'|s, a) \} \]  

(3.11)

where \( c(s, a) \) is the weighted immediate cost for each state \( s \) and action \( a \):

\[ c(s, a) = \alpha \cdot ||e(s)||^2 + \beta \cdot v(a) \]  

(3.12)

The expression 3.11 shows that the minimal cost \( J^* \) in a given state \( s \) can be obtained choosing an action \( a \) which minimizes the expected value of the sum between the immediate cost \( c(s, a) \) and the future cost \( J^*(s'|s, a) \), where \( s' \) is the state we get, considering that the action \( a \) was taken in state \( s \), weighted by a discount factor \( \gamma \).

A numerical solution for 3.11 is impracticable when there is a big number of possible states and actions [22] (problem known by the Curse of dimensionality). To avoid this problem, the minimal cost function \( J^* \) has to be approximated in different ways.

\( Q \)-learning presented in section 2.2.2.3 can be used to approximate the cost function and provide suboptimal protocols. It is based on an alternative approach to the Bellman Equation using an expected cost function \( Q(s, a) \):

\[ Q(s, a) = E \{ c(s, a) + \gamma \min_a Q(s'|s, a) \} \]  

(3.13)

Equation 3.13 states that, given a state \( s \) and an action \( a \), it is possible to obtain the minimum possible cost \( Q(s, a) \) if, from state \( s' \), the action chosen is the one that minimizes \( Q(s', a) \). The action-value function \( Q \) returns the minimum accumulated cost for an action \( a \) given a state \( s \). Then, in order to find the optimal policy given a determined state, one finds the action that minimizes 3.13:

\[ \pi(s) = \arg\min_a Q^*(s, a) \]  

(3.14)
where $Q^*(s,a)$ is the optimal Q function and $\pi(s)$ is the optimal protocol for state $s$.

An iterative way to compute $Q^*(s,a)$ is:

$$Q_{k+1}(s,a) = E\{c(s,a) + \gamma \min_{a'} Q_k(s', a'|s,a)\}$$  \hspace{1cm} (3.15)

where

$$\lim_{k \to \infty} Q_k(s,a) = Q^*(s,a)$$

As explained in section 2.2.2.4, tabular methods for Q-learning are not time and space efficient for a large number of possible states or actions. This is the reason why function approximators are used to generalize the Q-function. In order to compute 3.15, two different approximations, neural networks (based in [24]) and linear parametrization (based in [30]), were used.

Using these techniques, one parametrizes $Q_k(s,a) \approx Q(s,a,\theta_k)$. In order to find $Q^*$, one has to adjust, at every iteration, the parameters $\theta$, taking into account the cost function 3.9. In a neural network this is done by updating the synaptic weights during the training stage, and in the linear method one updates at every iteration the weights for each features vector $\psi(s)$.

For both approximations, in order to update the parameters $\theta$ at every iteration, the optimal Q value $Q^*$ and the Q value computed at the iteration in question $Q_k$ are compared using the loss function $L$. Substituting $Q^*$ by the approximated target $\hat{Q}(s,a,\hat{\theta}) = c(s,a) + \gamma \min_{a'} \hat{Q}(s', a', \hat{\theta})$, the function $L$ is given by:

$$L = (\hat{Q}(s,a,\hat{\theta}) - Q_k(s,a))^2$$

$$= (\hat{Q}(s,a,\hat{\theta}) - Q(s,a,\theta_k))^2$$  \hspace{1cm} (3.16)

The parameters $\theta$ are trained by minimizing the loss function 3.16, and $\hat{\theta}$ are obtained from the previous iterations. The target network parameters $\hat{\theta}$ are only updated with the Q-network parameters ($\theta$) every $C$ steps and are held fixed between individual updates for the neural networks approximation (see section 3.4.1), while for the linear parametrization approximation they are updated at every iteration (see section 3.4.2).

### 3.3 Definition of the State, Action and Cost

We explain in this section how the state $s$, action $a$ and cost $c$ were defined for both the networked control systems studied: $NCS_1$, illustrated in Figure 3.3 where the system has one agent and three channels, and $NCS_2$, illustrated in Figure 3.5 where the system has two agents and three channels.

#### 3.3.1 State $s$

The state to be considered is extremely connected to the formulation of the problem, and provides useful information about the environment. For $NCS_1$, the state $s$ is the estimation error of the system. As for $NCS_2$, the state is represented by the estimation error of each agent and by a variable that indicates the sum of collisions on the third channel from the ten previous iterations of the system. For example, if the system is on the 25th iteration, this variable will tell how many collisions occurred from the 15th to the 24th iterations. This is an important piece of information, as it provides to both agents an idea of the other agents transmission rate. The state for both NCSs at each iteration is represented next:
State $NCS_1$
\[
\begin{cases}
    e_k[0] \text{ first component of the estimation error} \\
    e_k[1] \text{ second component of the estimation error}
\end{cases}
\]

State $NCS_2$
\[
\begin{cases}
    e_i^k[0] \text{ first component of the estimation error of the } i^{th} \text{ agent} \\
    e_i^k[1] \text{ second component of the estimation error of the } i^{th} \text{ agent} \\
    C_{0k} \text{ collision buffer}
\end{cases}
\]

In $NCS_1$, the agent has access to its estimation error, and in $NCS_2$ each agent has access to its estimation error (but not the other agent's error), and to the collision buffer.

### 3.3.2 Action $a$

Similarly to the definition of the state $s$, the definition of the action $a$ that the protocol will take is also connected to how the problem is formulated. Again, each NCS has a different formulation: for $NCS_1$ the action $a$ is the number of channels used by the agent at each iteration, while for $NCS_2$ the action $a$ is the probability of each agent transmitting on the shared channel at each iteration.

Action $NCS_1$
\[
\{a \mid \text{Number of channels used}, \quad 1 \leq n \leq 3\}
\]

Action $NCS_2$
\[
\{p_i \mid \text{Probability of the agent } i \text{ transmitting in channel 3}, \quad 0 \leq p \leq 0.5\}
\]

In $NCS_1$, the agent can choose to transmit in 1, 2 or 3 channels at each iteration of the system, the action size is three. In $NCS_2$, the maximum probability of each agent transmitting in the shared channel is 0.5. Values larger than this would cause a lot of collisions, which would increase the estimation error of both agents, and so, the instability of the system. Also, the action space was discretized, so that $p_i \in \{0, 0.1, 0.2, 0.3, 0.4, 0.5\}$. This way, the action size is smaller, in this case equal to six, which will positively affect the performance of the learning algorithms.

### 3.3.3 Cost

For both NCSs, the instantaneous cost $c$ is a weighted sum between the estimation error and the number of channels used (equation 3.12). The difference between the NCSs is that, for $NCS_2$, it is summed the estimation errors of both agents, as well as the number of channels used. Note that in both cases the error is squared so that the cost function (function to be minimized) is always positive.

Cost $NCS_1$
\[
\begin{cases}
    c_k = \alpha [(e_k[0])^2 + (e_k[1])^2] + \beta v_k \quad \alpha, \beta \in [0, 1] \land \alpha + \beta = 1
\end{cases}
\]

Cost $NCS_2$
\[
\begin{cases}
    c_k = \alpha [(e_1^k[0])^2 + (e_1^k[1])^2 + (e_2^k[0])^2 + (e_2^k[1])^2] + \beta (v_1^k + v_2^k) \quad \alpha, \beta \in [0, 1] \land \alpha + \beta = 1
\end{cases}
\]

Algorithms of section 3.4.1 were built in a way that the agent tries to maximize the reward. As, in this case, maximizing the reward is the same as minimizing the cost, the reward was chosen as the opposite of the cost. So, for both NCSs, $r_k = -c_k$.

### 3.3.4 Environment

In table 3.1 are the properties of the environment for $NCS_1$ (Figure 3.3) and for $NCS_2$ (Figure 3.5):
In the environment of $NCS_1$ there is only one agent. The agent's state information allows it to fully observe the environment at each iteration, given that the estimation error provides all the information the agent needs to act (the greater the square of the error, the more channels the agent must use to try to reduce it). The policy is deterministic, as it is sure that the agent will transmit in $n$ channels if it decides (even if the transmission does not happen due to the probability of failure $p$).

On the other hand, in the environment of $NCS_2$ there are two agents sharing the communication medium. The only information each one of them possesses is its own error, and the number of collisions in the last ten iterations, so each agent does not know for sure if the other agent will transmit, or how is its transmission rate, the environment is only partially observable for each agent. The policy is also stochastic, as the action is a probability of transmitting in the shared channel.

For both environments, the time is discrete and the environment does not change dynamically between iterations, i.e., only the agents actions affect the environment.

### 3.4 Q-learning Algorithms

In this section we explain the Q-learning algorithms used in this thesis. Given that we applied two function approximation methods to each one of the two networked control systems, there are four algorithms:

1. Q-learning with neural networks for $NCS_1$, with one agent and three channels (represented in Figure 3.3)
2. Q-learning with neural networks for $NCS_2$, with two agents and three channels (represented in Figure 3.5)
3. Q-learning with linear parametrization for $NCS_1$, with one agent and three channels (represented in Figure 3.3)
4. Q-learning with linear parametrization for $NCS_2$, with two agents and three channels (represented in Figure 3.5)

#### 3.4.1 Q-learning with Neural Networks

These algorithms are based on [24] and have the end goal to train $Q(s, a)$ in order to converge to $Q^*(s, a)$.

Reinforcement learning may be unstable or divergent when a nonlinear function approximator such as a neural network is used to represent $Q$. This instability comes from the correlations present in the sequence of observations, the fact that small updates to $Q$ may significantly change the policy and data distribution, and the correlations between $Q$ and the target values [24].

The technique used in [24] to overcome this problem is called experience replay, a mechanism that uses a random sample of prior actions instead of the most recent action to proceed. This removes the correlations in the observation sequence and smooths changes in the data distribution. In practice, this is applied by using a memory bank $D$ that stores the state transitions, the past actions and the
corresponding rewards. This bank retrieves elements randomly to update the parameters \( \theta \) using loss function \( L \). Iterative update adjusts \( Q \) towards target values that are only periodically updated (at every \( C \) steps), which also allows to reduce correlations with the target.

Another particularity of the algorithm in [24] is that with uses two neural networks: one with weights \( \theta \), that are updated at every iteration, and other one with "frozen" weights \( \hat{\theta} \), that are updated every \( C \) steps. The target (line 12 of algorithm 1) is computed at every iteration using the frozen weights \( \hat{\theta} \). This makes the algorithm more stable compared to standard online Q-learning, where an update that increases \( Q(s_k, a_k) \) often also increases \( Q(s_{k+1}, a) \) for all \( a \), thus also increases the target \( \hat{Q}_j \), possibly leading to oscillations or divergence of the policy. Generating the targets using an older set of parameters adds a delay between the time an update to \( Q \) is made and the time the update affects the target \( \hat{Q}_j \), making divergence or oscillations much more unlikely [24].

### Algorithm 1  Q-learning with neural networks for NCS\(_1\)

1: Initialize replay memory \( D \) to capacity \( N \)
2: Initialize action-value function \( Q \) with random weights \( \theta_0 \)
3: Initialize target action-value function \( \hat{Q} \) with random weights \( \hat{\theta} = \theta_0 \)
4: for episode=1,M do
5: Initialize the initial state \( s_1 \)
6: for \( k=1,K \) do
7: With probability \( \epsilon \) select a random action \( a_k \)
8: otherwise select \( a_k = \text{argmax}_a \hat{Q}(s_k, a, \theta_k) \)
9: Execute action \( a_k \) in system and observe reward \( r_k \) and next state \( s_{k+1} \)
10: Store transition \((s_k, a_k, r_k, s_{k+1})\) in \( D \)
11: Sample random minibatch of transitions \((s_j, a_j, r_j, s_{j+1})\) from \( D \)
12: Set \( \hat{Q}_j = r_j + \gamma \text{max}_a \hat{Q}(s_{j+1}, a', \hat{\theta}) \)
13: Perform a gradient descent step on \( L = (\hat{Q}_j - Q(s_j, a_j, \theta_0))^2 \) with respect to the network parameters \( \theta_k \)
14: Every \( C \) steps set \( \hat{\theta} = \theta_k \)

In order to solve the exploration - exploitation dilemma explained in section 2.2.2.2 it is used a \( \epsilon \) – greedy exploration - with probability \( \epsilon \) choose a random action, otherwise go with the "greedy" action with the highest Q-value. As implemented in [24], the \( \epsilon \) decreases iteratively from 1 to 0.1, in a linear way. In the beginning the system makes completely random moves to explore the state space maximally, and then it settles down to a small fixed exploration rate.

The minibatch stores randomly sampled elements of the memory bank \( D \), and has a fixed batch size (it was chosen the same size as in [24], 32). At each iteration, instead of doing an online gradient descent step on the loss function \( L \), for only one element, the algorithm performs a batch gradient descent using all elements of the minibatch, batch size 32. More samples at each gradient descent step means more information to approximate \( Q^* \), which improves the chances of convergence of the algorithm.

There is one main differences between algorithms 1 and 2. In algorithm 1 there are two neural networks: one with weights \( \theta \) that are updated at every iteration, and other with weights \( \hat{\theta} \) that are updated every \( C \) steps \((\hat{\theta} = \theta)\). In algorithm 2 instead of having networks with "frozen" weights \( \hat{\theta}_1 \) and \( \hat{\theta}_2 \), that are updated every \( C \) steps by the weights of the networks of each agent, \( \theta_1 \) and \( \theta_2 \), respectively (as used in the original DQN algorithm in [24] and in algorithm 1), we update the weights of each agent’s network with the weights of the other agent’s network. In order to refresh the neural networks and reduce correlation, at every \( C_1 \) steps, \( \theta_1 = \theta_2 \), and at every \( C_2 \) steps, \( \theta_2 = \theta_1 \). Note that both operations may never occur at the same time step, as one of the networks would not be refreshed, so \( C_1 \) and \( C_2 \) must be chosen in order that this problem never happens.

For NCS\(_2\), as the two systems to control are identical and are affected by white noise with similar
Algorithm 2 Q-learning with neural networks for NCS

1: Initialize replay memory $D_1, D_2$ to capacity $N$
2: Initialize action-value functions $Q_1, Q_2$ with random weights $\theta_0^1, \theta_0^2$
3: for episode = 1,M do
4:     Initialize the initial state $s_1^1, s_1^2$
5:     for $k = 1,K$ do
6:         With probability $\epsilon_1, \epsilon_2$ select random actions $a_k^1, a_k^2$
7:         otherwise select $a_k^1 = \text{argmax}_a Q_1(s_k^1, a, \theta_k^1), a_k^2 = \text{argmax}_a Q_2(s_k^2, a, \theta_k^2)$
8:         Execute actions $a_k^1, a_k^2$ in system and observe reward $r_k$ and next state $s_{k+1}^1, s_{k+1}^2$
9:     Store transition $(s_k^1, a_k^1, r_k, s_{k+1}^1), (s_k^2, a_k^2, r_k, s_{k+1}^2)$ in $D_1, D_2$
10: Sample random minibatch of transitions $(s_j^1, a_j^1, r_j^1, s_{j+1}^1)$ from $D_1$
11: Set $Q_j^1 = r_j^1 + \gamma \max_{a'} Q_2(s_{j+1}^1, a', \theta_k^2)$
12: Perform a gradient descent step on $L_1 = (Q_j^1 - Q_1(s_j^1, a_j^1, \theta_k^1))^2$ with respect to the network parameters $\theta_k^1$
13: Sample random minibatch of transitions $(s_j^2, a_j^2, r_j^2, s_{j+1}^2)$ from $D_2$
14: Set $Q_j^2 = r_j^2 + \gamma \max_{a'} Q_1(s_{j+1}^2, a', \theta_k^1)$
15: Perform a gradient descent step on $L_2 = (Q_j^2 - Q_2(s_j^2, a_j^2, \theta_k^2))^2$ with respect to the network parameters $\theta_k^2$
16: Every $C_1$ steps set $\theta_k^1 = \theta_k^2$
17: Every $C_2$ steps set $\theta_k^2 = \theta_k^1$

distribution $N \sim (0, 1)$, is expected that the agents have similar transmission rates, so that none of them monopolizes the communication resources. Updating, in [2], the weights of an agent’s neural network with the weights of the other agent’s neural network helps both agents learning the same protocol, as function 3.16 aims to minimize the difference between both agents Q values. Also, we avoided creating two more “frozen” neural networks to update each agent’s neural network, like is used in [24] and in algorithm 1.

The architecture used for the neural networks (Figure 3.7) has a separate output unit for each possible action, and only the state representation is an input to the network. The outputs correspond to the predicted Q-values of the individual actions for the input state. The main advantage of this type of architecture is the ability to compute Q-values for all possible actions in a given state with only a single forward pass through the network [24].

![Figure 3.7: Architecture of the Q-network used (source [6])](https://www.tensorflow.org)

The programming language used to implement the algorithms was **Python**, with **TensorFlow** (an open source library for numerical computation and large-scale machine learning). [https://www.tensorflow.org](https://www.tensorflow.org)
and, built on TensorFlow, Keras (a high level application programming interface, or API) and, built on TensorFlow, Keras (a high level application programming interface, or API, \url{https://keras.io/}).

The networks for both algorithms 1 and 2 are multilayer perceptrons, with four layers (an input and an output layer with two more hidden layers - assuming the convention that hidden layers are the ones between input and output layers [5]) of nonlinearly-activating nodes, connected in a feedforward architecture. The training of the networks is based on the minimization of the loss function \(3.16\) through the backpropagation algorithm [5].

The activation function for the nodes of the hidden layers is the rectified linear unit (ReLU), a non-linear function \(f(x) = \max(0, x)\), which returns \(x\) if \(x > 0\) and 0 if \(x \leq 0\). Traditionally, the logistic sigmoid or hyperbolic tangent are used as activation functions in hidden layers. However, these functions saturate, and during backpropagation, they may produce a gradient of zero for large inputs, which can slow or even halt the learning process. The biggest advantage about using the ReLU function is that it does not saturate for large positive inputs [48]. The activation function for the output layer is the linear function \(f(x) = x\).

The choices in the architecture were based on the Cybenko's theorem [49], that states that a perceptron with a single layer of neurons with a sigmoid activation function and with a linear output unit can uniformly approximate any continuous function in any closed bounded set. This guarantees that even perceptrons with a single hidden layer can approximate essentially all useful functions. However, it does not provide any constructive method to find it, neither it gives any bounds on the number of hidden units needed to approximate a given function to a desired level of accuracy [5]. It was decided to add one more hidden unit and use ReLU as the activation function.

The training of multilayer perceptrons by the backpropagation algorithm is often rather slow. The essential reason for this is that the error surface, as a function of the weights, normally has narrow ravines. In these regions, the use of a large learning rate parameter \(\eta\) will lead to a divergent oscillation across the ravine. A small \(\eta\) will lead the weight vector to the 'bottom' of the ravine, and convergence to the minimum will then proceed along this bottom, but at a very low speed, because the gradient and \(\eta\) are both small [5]. To accelerate the training, we used Adam (adaptive moment estimation), a method that dynamically computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. Details of this algorithm are in [50].

3.4.2 Q-learning with Linear Parametrization

These algorithms are based on [30], and similarly to the algorithms in section 3.4.1, they have the end goal to train \(Q(s, a)\) in order to converge to \(Q^*(s, a)\).

As explained in section 2.4.3.1, there is a features vector for every action, in this case three, and each features vector contains a lot of information about the state. Given that the cost function is quadratic, it was assumed that the optimal Q-function also was, and so the features for algorithm [3] were chosen:

\[
\psi(s, a^1) = \begin{bmatrix} (e_1)^2, (e_2)^2, e_1 \cdot e_2, e_1, e_2, 1, \end{bmatrix}^T
\]

\[
\psi(s, a^2) = \begin{bmatrix} 0, (e_1)^2, (e_2)^2, e_1 \cdot e_2, e_1, e_2, 1, \end{bmatrix}^T
\]

\[
\psi(s, a^3) = \begin{bmatrix} 0, (e_1)^2, (e_2)^2, e_1 \cdot e_2, e_1, e_2, 1 \end{bmatrix}^T
\]

where \(e_1\) and \(e_2\) are the first and second components of the estimation error vector for state \(s\).

Each vector has 6 features, and the remaining values are zero. The Q-value computed at every
iteration for each action is calculated taking only into account the values of the parameter vector \( \theta \) corresponding to each action, given that the Q-value is the dot product between the parameter vector and the feature vector of the action: \( Q(s, a^i, \theta_k) = \theta_k^T \cdot \psi(s, a^i) \), \( i = 1, 2, 3 \). The action size is 3, and every action has 6 features, so the size \( d \) of the parameter vector \( \theta \) is equal to 18. The bias terms '1' in each of the features vectors are necessary to properly scale the function values independently of the features.

Similarly to algorithms \( [1] \) and \( [2] \), it is used a \( \epsilon \) - greedy exploration - with probability \( \epsilon \) choose a random action, otherwise go with the "greedy" action with the lowest Q-value (algorithms \( [3] \) and \( [4] \) were implemented to minimize the cost, and not to maximize the rewards). The \( \epsilon \) decreases iteratively from 1 to 0.1, in a linear way. In the beginning the system makes completely random moves to explore the state space maximally, and then it settles down to a small fixed exploration rate.

### Algorithm 3: Q-learning with linear parametrization for NCS\(_1\)

1. Initialize the initial state \( s_0 \)
2. Initialize \( \theta_0, \zeta_0 \in \mathbb{R}^d, A_0 \in \mathbb{R}^{d \times d} \)
3. for \( k = 0, K \) do
   4. If \( k = 0 \) then
      5. Update \( \psi(s_0, a^i) \), \( i = 1, 2, 3 \)
      6. Set \( Q(s_0, a^i, \theta_0) = \theta_0^T \cdot \psi(s_0, a^i) \) \( i = 1, 2, 3 \)
      7. With probability \( \epsilon \) select a random action \( a_0 \)
      8. Otherwise set \( a_0 = \arg\min_a Q(s_0, a, \theta_0) \)
      9. Set \( Q_0 = Q(s_0, a_0, \theta_0) \)
      10. Execute action \( a_0 \) in system and observe immediate cost \( c_k \) and next state \( s_{k+1} \)
   11. Update \( \psi(s_{k+1}, a^i) \), \( i = 1, 2, 3 \)
   12. Set \( Q(s_{k+1}, a^i, \theta_k) = \theta_k^T \cdot \psi(s_{k+1}, a^i) \) \( i = 1, 2, 3 \)
   13. With probability \( \epsilon \) select a random action \( a_{k+1} \)
   14. Otherwise set \( a_{k+1} = \arg\min_a Q(s_{k+1}, a, \theta_k) \)
   15. Set \( \phi_{k+1} = \arg\min_a Q(s_{k+1}, a, \theta_k) \)
   16. Set
      \[
      Q_{k+1} = Q(s_{k+1}, a_{k+1}, \theta_k), \psi_{k+1} = \psi(s_{k+1}, a_{k+1}), \phi_{k+1} = \arg\min_a Q_{k+1}(s_{k+1}, a, \theta_k), Q'_{k+1} = Q(s_{k+1}, \phi_{k+1}, \theta_k), \psi'_{k+1} = \psi(s_{k+1}, \phi_{k+1})
      \]
   17. Set
      \[
      \Delta_{k+1} = c_k + \beta \cdot (\Delta_{k} - Q_{k})
      \]
   18. Compute the matrix \( \Lambda_{k+1} = \zeta_k[\beta \cdot \psi'_{k+1} - \psi_k]^T \)
   19. Set \( \Lambda_{k+1} = \Lambda_k + \gamma_{k+1}[\Lambda_{k+1} - \Lambda_k] \)
   20. Compute the Moore-Penrose pseudo-inverse \( \hat{\Lambda}_{k+1} = [k^{-1} I + \hat{\Lambda}_{k+1} \Lambda_{k+1}]^{-1} \hat{\Lambda}_{k+1} \)
   21. Update the parameter vector \( \theta_{k+1} = \theta_k - \alpha_{k+1} \hat{\Lambda}_{k+1} \zeta_k d_{k+1} \)
   22. Update the eligibility vector \( \zeta_{k+1} = \lambda \beta \zeta_k + \psi_{k+1} \)

The temporal difference error \( d_{k+1} \) is computed at line 17, where \( c_k \) is the immediate cost (section 3.3.3) at every iteration, \( \beta \) is a discount rate, \( Q_{k+1} \) is the Q-value of the action chosen with greedy policy and \( Q_k \) is the Q-value of the action chosen with a \( \epsilon \) - greedy policy in the previous iteration.

At line 18 is computed the matrix \( \Lambda \), that is a linear combination of the eligibility vector \( \zeta \) and the difference between the features vector of the action chosen with greedy policy and the features vector of the action chosen with a \( \epsilon \) - greedy policy in the previous iteration.

The original Zap Q-learning (30) required an inversion of matrix \( \hat{\Lambda}_{k+1} \) at every iteration. This was computationally expensive, and would greatly increase the variance of the results. This problem was identified by the authors, and in June 2018 they published a "user's guide " (37), where they advised to use the Moore-Penrose pseudo-inverse (line 20) instead of the inverse.

Algorithm 4 is similar to algorithm 3 with the main difference that is applied to two agents instead of one. With this in mind, we changed the features vector \( \psi \). So, for each agent, the features vector for each action is represented in 3.18.
Algorithm 4 Q-learning with linear parametrization for NCS$_2$

1: Initialize the initial states $s^1_0,s^2_0$
2: Initialize $\theta^1_0, \theta^2_0, \zeta^1_0, \zeta^2_0 \in \mathbb{R}^d, A^1_0, A^2_0 \in \mathbb{R}^{d \times d}$
3: for $k=0,K$ do
   4:     if $k=0$ then
   5:         Update $\psi^1(s^1_0,a^i), \psi^2(s^2_0,a^i), i = 1,2,3,4,5,6$
   6:         Set $Q^1(s^1_0,a^1,\theta^1_0) = (\theta^1_0)^T \cdot \psi^1(s^1_0,a^1), Q^2(s^2_0,a^1,\theta^2_0) = (\theta^2_0)^T \cdot \psi^2(s^2_0,a^1), i = 1,2,3,4,5,6$
   7:         With probability $\epsilon_1, \epsilon_2$ select random actions $a^1_k, a^2_k$
   8:         Otherwise set $a^1_k = \arg \min_a Q^1(s^1_k,a), a^2_k = \arg \min_a Q^2(s^2_k,a)$
   9:         Set $Q^1_k = Q^1(s^1_k,a^1_k,\theta^1_k), Q^2_k = Q^2(s^2_k,a^2_k,\theta^2_k), \psi^1_k = \psi^1(s^1_k,a^1_k), \psi^2_k = \psi^2(s^2_k,a^2_k)$
10:        Execute actions $a^1_k, a^2_k$ in system and observe immediate cost $c_k$ and next state $s^1_{k+1}, s^2_{k+1}$
11:       Update $\psi^1(s^1_{k+1},a^i), \psi^2(s^2_{k+1},a^i), i = 1,2,3,4,5,6$
12:       With probability $\epsilon_1, \epsilon_2$ select random actions $a^1_{k+1}, a^2_{k+1}$
13:       Otherwise set $a^1_{k+1} = \arg \min_a Q^1(s^1_{k+1},a,\theta^1_k), a^2_{k+1} = \arg \min_a Q^2(s^2_{k+1},a,\theta^2_k)$
14:       Set $\phi^1_{k+1} = \arg \min_a Q^1(s^1_{k+1},a^1_k,\theta^1_k), \phi^2_{k+1} = \arg \min_a Q^2(s^2_{k+1},a^2_k,\theta^2_k)$
15:       Set $Q^1_{k+1} = Q^1(s^1_{k+1},a^1_{k+1},\theta^1_k), Q^2_{k+1} = Q^2(s^2_{k+1},a^2_{k+1},\theta^2_k), \psi^1_{k+1} = \psi^1(s^1_{k+1},a^1_{k+1}), \phi^1_{k+1} = \arg \min_a Q^1_{k+1}(s^1_{k+1},a^1_k,\theta^1_k)$
16:       Set $Q^2_{k+1} = Q^2(s^2_{k+1},a^2_{k+1},\theta^2_k), \psi^2_{k+1} = \psi^2(s^2_{k+1},a^2_{k+1}), \phi^2_{k+1} = \arg \min_a Q^2_{k+1}(s^2_{k+1},a^2_k,\theta^2_k)$
17:       Compute the matrices $A^1_{k+1} = \zeta^1_k(\beta \cdot \psi^1_k - \psi^1_k)^T, A^2_{k+1} = \zeta^2_k(\beta \cdot \psi^2_k - \psi^2_k)^T$
18:       Set $A^1_{k+1} = A^1_{k+1} + \gamma_{k+1}[A^1_{k+1} - A^1_k], A^2_{k+1} = A^2_{k+1} + \gamma_{k+1}[A^2_{k+1} - A^2_k]$
19:       Compute the Moore-Penrose pseudo-inverses $A^1_{k+1}^{-1} = [k^{-1}I + A^1_{k+1} A^1_k]^{-1} A^1_{k+1}, A^2_{k+1}^{-1} = [k^{-1}I + A^2_{k+1} A^2_k]^{-1} A^2_{k+1}$
20:       Update the parameter vectors $\theta^1_{k+1} = \theta^1_k + \alpha_k A^1_{k+1}^T d^1_{k+1}, \theta^2_{k+1} = \theta^2_k - \alpha_k A^2_{k+1}^T d^2_{k+1}$
21:       Update the eligibility vectors $\zeta^1_{k+1} = \lambda \beta \zeta^1_k, \zeta^2_{k+1} = \lambda \beta \zeta^2_k + \psi^1_k$

where $i = 1,2$ represent agent 1 and agent 2, $e^1_i$ and $e^2_i$ are the first and second components of the estimation error vector for each agent and $Co$ is the collision buffer.

The difference from [3.17] is the inclusion of the collision buffer, that brings knowledge about the activity of the other agent, so now the features vector has 7 features, and the remaining cells are zero. Similarly to [3.17] the Q-value computed every iteration for each action is calculated taking only into account the values of the parameter vector $\theta^i$ corresponding to each action.
The action size is 6, and every action has 7 features, so the size \( d \) of the parameter vector \( \theta^i \) is equal to 42. Given the enormous size of \( d \), the Moore-Penrose technique used in algorithm [3] was not enough to reduce the variance resultant from the matrix inversion needed at each step, so it was needed to implement the **Control Variates** technique.

### 3.4.2.1 Control Variates

The Control Variates (CV) technique is a variance reduction technique. Assume the simulation objective is to estimate the mean of a random variable \( Y \). The CV method relies on one or more auxiliary random variables called *controls* and uses information about these variables (their known means) to reduce the variance of the estimator for \( E[Y] \) [51]. An effective control \( X \) needs to satisfy two requirements:

1. \( X \) needs to be correlated with \( Y \) (carry some information about \( Y \));
2. \( E[X] \) needs to be known.

For algorithm [4], the objective is to reduce the variance of \( Y_1 = \beta \cdot Q^1(s_{k+1}^i, \phi_k^i, \theta_{k+1}^i) - Q^1(s_k^i, \phi_k^i, \theta_k^i) \) and of \( Y_2 = \beta \cdot Q^2(s_{k+1}^i, \phi_k^i, \theta_{k+1}^i) - Q^2(s_k^i, \phi_k^i, \theta_k^i) \), which are the Q approximation errors for agent 1 and agent 2, respectively. Assuming a protocol \( \pi \) with constant probabilities of transmitting in the shared channel for both agents \( p_1 = p_2 = 0.5 \), we computed the future value function for both agents \( (V_1^\pi(s^+) \text{ and } V_2^\pi(s^+)) \) and used it as controls, as they approximate the Q function and their expected value can be computed.

The Bellman equation for the value function is:

\[
V_i^\pi(s) = c(s, a) + E[V_i(s^+)|a = \pi(s)] \tag{3.19}
\]

where \( s \) represents the state, \( a \) represents the action, and \( c \) is the immediate cost function in [3.12]

Following protocol \( \pi \), the cost of communication is constant. So, the value function for each agent was computed considering only the estimation error.

\[
V_i^\pi(\epsilon_i) = c_i^T H \epsilon_i + \frac{\gamma}{1 - \gamma} \text{tr}(HR_w), \quad i = 1, 2 \tag{3.20}
\]

where \( \epsilon_i \) is the estimation error vector of agent \( i \), \( R_w \) is the covariance matrix of the white noise (in this case \( R_w = I_{2 \times 2} \)), \( \gamma \in [0, 1] \) is a discount factor and \( H \) is the solution to the discrete Lyapunov equation:

\[
\sqrt{\gamma \cdot p_f^\pi(A^T HA)} - H + Q = 0 \tag{3.21}
\]

where the cost of the estimation error for agent \( i \) is \( c_e(\epsilon_i) = c_i^T Q \epsilon_i \) (in this case \( Q = I_{2 \times 2} \)), \( A = \begin{pmatrix} 0 & 1 \\ \frac{1}{2} & 1 \end{pmatrix} \) is the state matrix and \( p_f^\pi = p \cdot (1 - p_a + p_a(p + p_a)) \) is the communication failure probability for following protocol \( \pi \), where \( p_a = p_1 = p_2 = 0.5 \) and \( p \) is the probability of failure in each communication channel (in this case \( p = 0.3 \)).

The Lyapunov function gives us the solution to the Bellman Equation for a fixed protocol. So, solving the Lyapunov function in [3.21], we find the solution to the Bellman equation in [3.19] and the value functions for the constant actions protocol. Expressions [3.20] and [3.21] were deduced in [1]. For each time step \( k \), the future value function for the agents can be written as:

\[
E[V_i^\pi(\epsilon_{k+1})] = E[p_f^\pi(Ae_k^i - w^i)^T H(Ae_k^i - w^i) + (1 - p_f^\pi)(w^i)^T H w^i] + \frac{\gamma}{1 - \gamma} \text{tr}(HR_w), \quad i = 1, 2 \tag{3.22}
\]
where $e_i^k$ is the estimation vector of agent $i$ at iteration $k$ and $w_i^k$ is the vector of Gaussian noise that affects the system of agent $i$. The constant term $w$ in (3.22) can be ignored as is not relevant for the CV method. The expectation of the control for each agent at each time step is:

$$E[V^\pi_i(e_{k+1})] = p_i^T e_i^k A^T H A e_i^k + \frac{\gamma}{1 - \gamma} tr(H R w)$$

(3.23)

### 3.5 Validation Methods for the Results

The results obtained by the Q-learning algorithms with neural networks (section 3.4.1) were extracted from validation simulations. Firstly, the system is trained and the neural networks weights are updated. After training, the weights are frozen, and the actions taken on validation are generated by the protocol in question, there is no random action. The parameters used in training and validation are in table 3.2.

A epoch can be defined as a training sweep through the whole training set [5], i.e., is when an entire dataset is passed forward and backward through the neural network. For algorithms 1 and 3, this dataset is the *minibatch*, with batch size 32 (see section 3.4). The number of epochs for both algorithms are in table 3.2. These values were obtained empirically, observing the evolution of the loss function throughout the training. We have chosen, for each algorithm, a number of epochs big enough so that the loss function would converge to a minimum value.

For the Q-learning algorithms with neural networks, the training has 50 simulation windows, each one with 2000 iterations, which results in a training set of size 100000 iterations. The validation also has 50 simulation windows, each one with 2000 iterations, so the dimension of the validation set is also 100000 iterations.

<table>
<thead>
<tr>
<th>Number of simulation windows (M)</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size of the simulation windows (K)</td>
<td>2000</td>
</tr>
<tr>
<td>Interval of values of $\alpha$ and $\beta$</td>
<td>$[0,1]$</td>
</tr>
<tr>
<td>Probability of transmission failure (per channel)</td>
<td>30%</td>
</tr>
<tr>
<td>Epochs for algorithm 1</td>
<td>2</td>
</tr>
<tr>
<td>Epochs for algorithm 3</td>
<td>200</td>
</tr>
</tbody>
</table>

Table 3.2: Parameters used in training and validation of the algorithms from section 3.4.1

The Q-learning algorithms with linear parametrization (section 3.4.2) are not separated into training and validation cycles. Instead, there is only one simulation window for the learning process. The training is performed from the beginning to the half of the cycle, and the results are extracted from half of the cycle till the end of it, given that, by the beginning of that period, the learning process is practically finished and there is little variation between the actions taken by the protocol. In table 3.3 are the parameters used.
<table>
<thead>
<tr>
<th>K (size of the simulation window)</th>
<th>400000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interval from where results are extracted for algorithm 3</td>
<td>[200000,400000]</td>
</tr>
<tr>
<td>Interval from where results are extracted for algorithm 4</td>
<td>[250000,500000]</td>
</tr>
<tr>
<td>Interval of values of $\alpha$ and $\beta$</td>
<td>[0,1]</td>
</tr>
<tr>
<td>Probability of transmission failure (per channel)</td>
<td>30%</td>
</tr>
</tbody>
</table>

Table 3.3: Parameters of the algorithms from section 3.4.2

The data extracted from the simulations include the average of the error and the usage of the medium (average amount of channels used throughout the simulation), both essential to understand the efficiency of the protocols generated.
Chapter 4

Results

In this chapter, we present the results obtained for the algorithms of section 3.4 for both $NCS_1$, networked control system with one agent and three communication agents (figure 3.3), and $NCS_2$, networked control system with two agents and three communication channels (Figure 3.5).

Communication protocols are obtained by applying the algorithms and changing the constants $\alpha$ and $\beta$ of the immediate cost function (as explained in section 3.3.3) between two extremes:

- For $\alpha = 1$, $\beta = 0$, the cost function is entirely based on the estimation error, and not on how many channels were used. It is expected that the agent(s) use all channels available in order to lower the error the most possible.

- For $\alpha = 0$, $\beta = 1$, the cost function is only influenced by the the cost of communication, and not the estimation error. It is expected that the agent(s) ignore the error, and use as little channels as possible.

The state space matrices used for the control systems are in table 4.1. The state equations of the linear time-invariant plants in both NCSs are explained in section 3.1, as well as the controller used.

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\begin{bmatrix} 1 &amp; 1 \ 0 &amp; 1 \end{bmatrix}$</td>
<td>$\begin{bmatrix} 0.2 \ 0.7 \end{bmatrix}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$C$</th>
<th>$R_w$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\begin{bmatrix} 1 &amp; 0 \ 0 &amp; 1 \end{bmatrix}$</td>
<td>$\begin{bmatrix} 1 &amp; 0 \ 0 &amp; 1 \end{bmatrix}$</td>
</tr>
</tbody>
</table>

Table 4.1: State space matrices for $NCS_1$ and $NCS_2$

4.1 Results for $NCS_1$

The context of this NCS is simpler, given that there is only one agent and no competition for the communication channels. The simulation of this NCS has the purpose of testing the Q-learning algorithms, with both approximations, for a simpler problem.
4.1.1 Q-learning with Neural Networks

In this section are shown the results of the application of algorithm 1. Its training parameters are in table 4.2.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M$ (number of simulation windows)</td>
<td>50</td>
</tr>
<tr>
<td>$K$ (size of the simulation windows)</td>
<td>2000</td>
</tr>
<tr>
<td>Interval of values of $\alpha$ and $\beta$</td>
<td>[0,1]</td>
</tr>
<tr>
<td>Probability of transmission failure (per channel)</td>
<td>30%</td>
</tr>
<tr>
<td>Total number of channels</td>
<td>3</td>
</tr>
<tr>
<td>$C$ (step size for the update of $\hat{\theta}$)</td>
<td>10</td>
</tr>
<tr>
<td>Epochs for the training of the network</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 4.2: Parameters used in training of the algorithm 1

Figure 4.1 exemplifies how protocols are generated. For the parameters of table 4.2, it shows the response of a protocol generated after training.

Figure 4.1 illustrates the quadratic error and the channels used by the protocol during a validation simulation, where the quadratic error is the same used in the cost function (section 3.3.3), which is the sum of the squared components of the estimation error vector at every iteration, $e_k[0]^2 + e_k[1]^2$. The average quadratic error (or mean squared error) and the average of channels used are stored at each of the 50 validation simulations, and the final values for each protocol are the average of the values obtained at the simulations.

The quality of a protocol is evaluated by the relation between the estimation error and the cost of communication. In order to evaluate the protocols generated, are proposed some trivial protocols. Comparing these protocols with the "Q protocols" generated by algorithm 1, one can have an idea if there was, or not, a good training. The following trivial protocols were proposed:

- **Protocol 1** - Use only one channel;
- **Protocol 2** - Use all channels available;
- **Protocol 3** - Use the channels available in a random manner
Protocol 1 transmits every time in only one channel, protocol 2 transmits every time the same packet in the three channels available, and protocol 3 chooses to transmit in one, two or three channels at each iteration, following a uniform distribution. Table 4.3 shows the comparison between the trivial protocols and the Q protocol illustrated in Figure 4.1 ($\alpha = 0.8, \beta = 0.2$). The parameters chosen to evaluate the protocols are the usage of the medium and the mean squared error.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage of the Medium (%)</td>
<td>33.3%</td>
<td>100%</td>
<td>66.7%</td>
<td>98.3%</td>
</tr>
<tr>
<td>Mean Squared Error</td>
<td>4.1892</td>
<td>2.0894</td>
<td>2.5856</td>
<td>2.1172</td>
</tr>
</tbody>
</table>

Table 4.3: Comparison between the trivial protocols and the Q protocol generated by algorithm 1 ($\alpha = 0.8, \beta = 0.2$)

In a more general way, different protocols can be obtained by changing the weights $\alpha, \beta$ of cost function 3.12. These protocols create a function of suboptimal protocols, that correlate the estimation error and the communication cost in different ways. For $\alpha = \{0, 0.1, 0.2, ..., 1\}$, we ran algorithm 1 to compute the Q protocols. The Q protocols obtained are shown in Figure 4.2.

![Figure 4.2](image)

(a) Quadratic error as a function of the weight $\alpha$ (b) Channels used as a function of the weight $\alpha$

![Pareto frontier NCS1](image)

(c) Pareto frontier for the Q protocols as a function of the quadratic error

Figure 4.2: Q protocols obtained by algorithm 1

Figures 4.2a and 4.2b show the evolution of the quadratic error and the channels of each protocol as a function of the weight parameter $\alpha$. As $\alpha$ increases, the estimation error decreases and the channels
used at each iteration increase.

Figure 4.2c illustrates the correlation between the mean squared error and the channels used by all the suboptimal protocols generated, as well as the trivial protocols of table 4.3. It was drawn a Pareto frontier to approximate a function of the Q protocols with the best mean squared error - number of channels used relation.

### 4.1.2 Q-learning with Linear Parametrization

In this section are shown the results of the application of algorithm 3. Its parameters are in table 4.4.

<table>
<thead>
<tr>
<th>K (size of the simulation window)</th>
<th>400000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interval from where results were extracted</td>
<td>[200000,400000]</td>
</tr>
<tr>
<td>Interval of values of $\alpha$ and $\beta$</td>
<td>[0,1]</td>
</tr>
<tr>
<td>Probability of transmission failure (per channel)</td>
<td>30%</td>
</tr>
<tr>
<td>C (total number of channels)</td>
<td>3</td>
</tr>
<tr>
<td>Learning rate $\alpha_{n+1}$</td>
<td>$1/n + 1$</td>
</tr>
<tr>
<td>Learning rate $\gamma_{n+1}$</td>
<td>$1/n \alpha_{n+1}$</td>
</tr>
</tbody>
</table>

Table 4.4: Parameters used for algorithm 3

As explained in section 3.5, there is no separation between the training and validation of algorithms 3 and 4, so the values of the quadratic error and number of channels chosen by the protocol are extracted from iteration 200000 to the final of the simulation. In this interval of iterations the agent has gone through most of the learning process, and the actions taken by the protocol have little variance.

Similarly to section 4.1.1, it is chosen a Q protocol generated by algorithm 3 to be compared with trivial algorithms, in order to measure the success of the algorithm. The weight parameters chosen to generate the protocol are also $\alpha = 0.8, \beta = 0.2$ and the trivial protocols 1,2,3 are the same ones of section 4.1.1. The results are in table 4.5.

<table>
<thead>
<tr>
<th>Trivial Protocols</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage of the Medium (%)</td>
</tr>
<tr>
<td>Mean Squared Error</td>
</tr>
</tbody>
</table>

Table 4.5: Comparison between the trivial protocols and the Q protocol generated by algorithm 3 ($\alpha = 0.8, \beta = 0.2$)

Changing the weights $\alpha$ and $\beta$ of cost function 3.12, one obtains many suboptimal protocols that correlate the estimation error and the number of channels chosen by the agent.

Figures 4.3a and 4.3b show the relation between the mean squared error and the channels chosen by each suboptimal protocol and the weight factor $\alpha$ (remember that the weight factor $\beta = 1 - \alpha$ and the increase of $\alpha$ makes the estimation error a "more important" factor than the usage of the medium in cost function 3.12).

The Pareto frontier drawn in Figure 4.3c illustrates how the mean squared error and the number of channels chosen by each suboptimal protocol generated by algorithm 3 correlate, and compares them with the trivial protocols of table 4.5.

48
4.1.3 Results Analysis for $NCS_1$

In this section the results are explained and analyzed, with special concern to the performance of algorithms $[1]$ and $[3]$ that aim to generate protocols for the usage of the communication medium of $NCS_1$ (Figure 3.3).

$NCS_1$ introduced a simple problem: in the feedback loop of a control system, the communication between the sensor and the controller is made through a network with three erasure channels, each one with probability of packet dropping of $p = 0.3$. Given two weight factors $\alpha$ and $\beta$, the algorithms applied to this networked control system, algorithm $[1]$ with neural networks approximation and algorithm $[3]$ with a linear parametrization approximation, are expected to produce optimal protocols that uses the redundant transmission of packets discussed in $[1]$.

If the “user” thinks that the estimation error is too high for a normal performance of the system, it should increase $\alpha$ and decrease $\beta$. This way, the protocol generated by the algorithms should make the sensor send redundant packets through the channels available at each iteration. In sum, use more channels but decrease the estimation error and consequently the probability of transmission failure.

On the other hand, if the system is using too many channels, occupying resources and spending too much energy on sending redundant transmissions that are little useful, the user may increase $\beta$ and decrease $\alpha$, by which the protocol generated by the algorithms should make the sensor use less communication channels at each iteration. The estimation error will increase and the probability of failing the transmission will too, but it will use less channels, saving resources and energy. It is a user friendly
solution that is dynamic and adaptable for different contexts of control.

Observing the results of section 4.1, one can conclude that both algorithms produced protocols that fulfilled the expectations. For algorithm 1 (section 4.1.1), we can see in Figure 4.2a that the mean squared error decreases as $\alpha$ increases. Figure 4.2b shows us the evolution of the number of channels chosen by each protocol as alpha increases. The function is similar to a step function:

$$\text{Average number of channels used} = \begin{cases} 
1, & \alpha \in [0, 0.3] \\
2, & \alpha \in [0.4, 0.7] \\
3, & \alpha \in [0.8, 1] 
\end{cases} \quad (4.1)$$

The Pareto frontier for the channels chosen by the protocols as a function of the mean squared error is represented in Figure 4.2c. It is clear to conclude that the estimation error increases as the number of channels used decreases, and vice-versa. Also, for the extreme cases (number of channels equal to 1 or 3), the Q protocols generated have a similar error performance as the trivial protocols 1 and 2 (see table 4.3). As for the intermediate protocols, it is possible to see by the Pareto frontier crossed that the Q protocols have a better error-channels relation than trivial protocol 3.

As for algorithm 3 (see section 4.1.2), Figures 4.3a and 4.3b show us that the mean squared error and the channels used by the protocols decrease/increase, respectively, as the weight parameter $\alpha$ increases. As for Figure 4.3c and similarly to the results of algorithm 1, the protocols of the extremes have a similar error-channels relation as the trivial protocols 1 and 2 (see table 4.5). However, the estimation error increases in a more linearly as the channels used by the protocols decrease. These intermediate protocols, besides having a better error-channels ratio than intermediate trivial protocol 3, they also have a better ratio than the ones obtained by algorithm 1 in Figure 4.2c.

Both algorithms produced different protocols: algorithm 1 produced a “step” function (as is shown in equation 4.1), while algorithm 3 produced a “smoother” function (the Pareto frontier drawn in Figure 4.3c shows that the number of channels chosen by the protocols evolve in a linear way as a function of the mean squared error). Despite the different results, both achieved the main goals, to make the agent learn to send redundant packet transmissions to more channels as the weight factor $\alpha$ increases, decreasing the estimation error, and to have a equal or better error-channels relation than the trivial protocols presented, which is the prove that the training was well performed.

$NCS_1$ was used as a benchmark for the neural networks and linear approximations of the reinforcement learning algorithms. If both algorithms 1 and 3 were able to learn the solution to a simpler problem, as they were, then there’s a chance that these algorithms can be adapted and used in a more complex problem.

### 4.2 Results for $NCS_2$

As explained in section 3.1, $NCS_2$ is composed by two agents sharing the communication medium. Agent 1 will always transmit in channel 1, agent 2 will always transmit in channel 2, and both agents will share channel 3. The protocols generated by algorithms 2 and 4 state what are the probabilities of each agent transmitting in the shared channel, $p_1$ and $p_2$.

The training of each of the agents in $NCS_2$ is independent. However, they share the same cost function, which is computed considering the estimation error and usage of the medium of both agents (see section 3.3.3), and the collision buffer is present in both agents’ state (see section 3.3.1). Given these factors, we have here a cooperative game between the agents.
4.2.1 Q-learning with Neural Networks

In this section we show the results of the application of algorithm 2. Its training parameters are in Table 4.6.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>M (number of simulation windows)</td>
<td>50</td>
</tr>
<tr>
<td>K (size of the simulation windows)</td>
<td>2000</td>
</tr>
<tr>
<td>Interval of values of $\alpha$ and $\beta$</td>
<td>[0,1]</td>
</tr>
<tr>
<td>Probability of transmission failure (per channel)</td>
<td>30%</td>
</tr>
<tr>
<td>Total number of channels</td>
<td>3</td>
</tr>
<tr>
<td>Number of agents</td>
<td>2</td>
</tr>
<tr>
<td>Number of shared channels</td>
<td>1</td>
</tr>
<tr>
<td>$C_1, C_2$ (step sizes for the update of $\theta_1$ and $\theta_2$)</td>
<td>10</td>
</tr>
<tr>
<td>Epochs for the training of the network</td>
<td>200</td>
</tr>
</tbody>
</table>

Table 4.6: Parameters used in training of the algorithm 2

As explained in section 3.4.1, the updates $\theta_1 = \theta_2$ and $\theta_2 = \theta_1$ can never occur at the same time step. So, we chose to update $\theta_1$ if the number of the time step ends in 0, and to update $\theta_2$ if the number of the time step ends in 5. Figure 4.4 exemplifies how protocols are generated for $NCS_2$. For the parameters of Table 4.6, it shows the response of a protocol generated after training.

Figure 4.4 illustrates the quadratic error and the channels used by the protocol during a validation.
simulation, where the quadratic error is the same used in the cost function (section 3.3.3), which is the sum of the squared components of the estimation error vector of each agent at every iteration, 

$$[(e_1[k][0])^2 + (e_1[k][1])^2 + (e_2[k][0])^2 + (e_2[k][1])^2].$$

Similarly to section 4.1.1, the average quadratic error and the average of channels used are stored at each of the 50 validation simulations, and the final values for each protocol are the average of the values obtained at the simulations. The difference in this case is there are two agents, the protocol must choose an action for each agent at every time step. There is also another factor that has to be taken into consideration, the packet collisions, which happen when both agents transmit at the same time in the shared general. In general, a good protocol must not have an excessive rate of collisions.

As was done in section 4.1.1, are proposed some trivial protocols to evaluate and compare the protocols generated. Comparing these protocols with the "Q protocols" computed by algorithm 2, one can have an idea if there was, or not, a good training. The following trivial protocols were proposed:

- **Protocol 1** - Transmitting without using the third channel;
- **Protocol 2** - Transmitting in the third channel with probabilities \( p_1 = p_2 = 0.9 \);
- **Protocol 3** - Transmitting in the third channel with probabilities \( p_1 = p_2 = 0.5 \).

Protocol 1 introduces a null probability on transmitting in the shared channel (agent 1 only transmits in channel 1, agent 2 only transmits in channel 2), protocol 2 makes both agents transmit in the shared channel with probability 0.9, and protocol 3 makes the agents transmit in the shared channel probabilities \( p_1 = p_2 = 0.5 \). Table 4.7 shows the comparison between the trivial protocols and the Q protocol illustrated in Figure 4.4 (\( \alpha = 0.6, \beta = 0.4 \)).

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Usage of the Medium (%) (by agent)</strong></td>
<td>50%</td>
<td>95%</td>
<td>75%</td>
<td>( \approx 64.5% )</td>
</tr>
<tr>
<td><strong>Mean Squared Error</strong></td>
<td>5.2471</td>
<td>5.0390</td>
<td>4.7028</td>
<td>4.7736</td>
</tr>
<tr>
<td><strong>Collisions (%)</strong></td>
<td>0%</td>
<td>80.913%</td>
<td>24.814%</td>
<td>8.442%</td>
</tr>
<tr>
<td><strong>No Transmission Rate (%) (Agent 1)</strong></td>
<td>30.184%</td>
<td>28.280%</td>
<td>24.888%</td>
<td>25.612%</td>
</tr>
<tr>
<td><strong>No Transmission Rate (%) (Agent 2)</strong></td>
<td>30.155%</td>
<td>28.197%</td>
<td>24.922%</td>
<td>26.034%</td>
</tr>
</tbody>
</table>

Table 4.7: Comparison between the trivial protocols and the Q protocol generated by algorithm 2 (\( \alpha = 0.6, \beta = 0.4 \)).

Besides the usage of the medium and the mean squared error, already used in section 4.1 to compare Q protocols, for NCS2 are also compared the percentage of collisions in the shared channel and the percentage of transmissions that were not successful for each agent. All of these parameters are excellent markers to know if the training was well performed.

By varying the weights \( \alpha \) and \( \beta \) of the cost function 3.12 before the training of algorithm 2, we can obtain several protocols that act differently as the context where the NCS is inserted changes (user wants to use less channels, or wants to adjust the error, etc.). For \( \alpha = \{0, 0.1, 0.2, \ldots, 1\} \), we ran algorithm 2 and computed the Q protocols. The protocols obtained are shown in Figure 4.5.

Figures 4.5a and 4.5b illustrate the mean squared error and the channels for both agents of each generated protocol as a function of \( \alpha \). Figure 4.5c shows the correlation between the quadratic error and the channels for both agents of each generated protocol, as well as the results for the trivial protocols (table 4.7). In this Figure was drawn a Pareto frontier for each agent to approximate a function regarding the Q protocols with better mean squared error - number of channels used relation.
4.2.2 Q-learning with Linear Parametrization

In this section are shown the results of the application of algorithm 4. Its parameters are in table 4.8.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>K (size of the simulation window)</td>
<td>500000</td>
</tr>
<tr>
<td>Interval from where results were extracted</td>
<td>[250000, 500000]</td>
</tr>
<tr>
<td>Interval of values of $\alpha$ and $\beta$</td>
<td>[0, 1]</td>
</tr>
<tr>
<td>Probability of transmission failure (per channel)</td>
<td>30%</td>
</tr>
<tr>
<td>C (total number of channels)</td>
<td>3</td>
</tr>
<tr>
<td>Number of agents</td>
<td>2</td>
</tr>
<tr>
<td>Number of shared channels</td>
<td>1</td>
</tr>
<tr>
<td>Learning rate $\alpha_{n+1}$</td>
<td>$\frac{1}{n^{0.6}+1}$</td>
</tr>
<tr>
<td>Learning rate $\gamma_{n+1}$</td>
<td>$\frac{1}{n^{0.4}+1}$</td>
</tr>
</tbody>
</table>

Table 4.8: Parameters used for algorithm 4

In order to compare and evaluate the performance of the Q protocols generated by algorithm 4, some trivial protocols were simulated in the NCS$_2$ environment, exactly like in section 4.2.1. In table 4.9, those trivial protocols are compared with the Q protocol computed by algorithm 4 with weight parameters $\alpha = 0.6, \beta = 0.4$. 

Figure 4.5: Q protocols obtained by algorithm 2
Table 4.9: Comparison between the trivial protocols and the Q protocol generated by algorithm 4 ($\alpha = 0.6, \beta = 0.4$)

Again, we changed weights $\alpha$ and $\beta$ of the cost function between 0 and 1, obtaining several protocols with different error-channels used relation. For $\alpha = 0, 0.1, 0.2, ..., 1$, we simulated algorithm 4 and computed the Q protocols. The protocols obtained are displayed in Figure 4.6.

Figures 4.6a and 4.6b illustrate the evolution of the mean squared error and the channels for both agents as a function of $\alpha$. Figure 4.6c shows the Pareto frontier between the quadratic error and the channels for both agents of each generated protocol, as well as the results for the trivial protocols (table 4.9).

54
4.2.3 Results Analysis for $NCS_2$

In this section the results are explained and analyzed, with special concern to the performance of algorithms 2 and 4, that aim to generate protocols for the usage of the communication medium of $NCS_2$ (Figure 3.5).

While the problem of $NCS_1$ was a simpler one, the problem that $NCS_2$ gives us is a little more complex. This time, we have two identical control systems. The communication between the sensor and the controller of each system is made through a network, and both systems share the same network. It is composed by three erasure channels, each one with probability of packet dropping of $p = 0.3$. Sensor 1 always uses channel 1 to transmit to controller 1, sensor 2 always uses channel 2 to transmit to controller 2, and channel 3 is shared by both.

The goal of the reinforcement learning algorithms applied to this networked control system, algorithm 2 with neural networks approximation and algorithm 4 with a linear parametrization approximation, is to, given two weight factors $\alpha$ and $\beta$, design communication protocols that choose the optimal probability of each agent transmitting in the shared channel, $p_1$ and $p_2$, thus deciding how the agents share the communication medium.

How should the agents cooperate? For the extreme case of $\alpha = 0$ and $\beta = 1$, the agents should use the least communication channels possible. In this case, the solution is simple, they should only transmit in the respective channels, and not transmit in the shared channel. However, as $\alpha$ increases and $\beta$ decreases, the problem becomes more complicated. For example, in the other extreme case $\alpha = 1$ and $\beta = 0$, the agent should focus on minimizing the mean squared error as much as possible. For $NCS_1$, that would be as simple as making the agent transmit the same packet in all the channels available. The equivalent for $NCS_2$ would be for each agent to transmit in its respective channel and in the shared channel at every iteration. However, that would not have the desired effect, as there would be, at every time step, a packet collision in the shared channel, resulting in the loss of information shared through that channel. In the end, the error would be equal to the situation where they would only use the respective channels, with the downside of having transmitted twice the number of packets, spending double the energy! Figure 4.7 shows that the collision rate increases as the probability of both agents using the shared channel increases, as expected.

![Collisions in the Shared Channel](image)

Figure 4.7: Percentage of packet collisions in the shared channel as a function of the channels used by each agent

Another problem that may arise is that agents learn protocols that make them compete for the com-
munication resources. For example, if one agent “monopolizes” the shared channel, it will always transmit in its respective channel and in the shared one, while the other will only transmit in its respective channel. This may prevent packet collision, but ultimately the estimation error of the agent not using the shared channel will be high. The goal is for both agents to cooperate, and to transmit in the shared channel with a probability that helps them reduce both errors. In order to know the relation between the values of the mean squared error and the number of channels used by each agent for NCS, we simulated the system. For probabilities of transmitting in the shared channel of \( p_1 = p_2 = \{0.0, 0.1, 0.2, ..., 1.0\} \), we extracted the mean squared error of the system. The function obtained (Figure 4.8) shows the mean squared error as a function of the channels used by each agent. It is a quadratic function, and the minimum error is achieved when each agent uses 1.5 channels, i.e., they transmit in the shared channel with probabilities \( p_1 = p_2 = 0.5 \).

![Figure 4.8: Values of the error as a function of the channels used by each agent](image)

Figure 4.8 helps explaining why the action space for the agents was limited to \( p_1, p_2 \in [0, 0.5] \) (see section 3.3.2). Given that it is a quadratic function with minimum in 1.5 channels, it is symmetric to the vertical line \( x = 1.5 \). So the protocols obtained with the probabilities in the interval \([0, 0.5]\) would have the same error value as the ones obtained with the probabilities in the interval \([0.5, 1]\), but at a much larger communication cost. The action space was diminished in half, thus reducing the computation necessary to make the algorithms converge.

The Q-learning approximation with neural networks of algorithm 2 was able to obtain the desired results. Figure 4.5a clearly shows that the mean squared error decreases to its minimum as the weight factor \( \alpha \) increases from 0 to 1, while Figure 4.5b shows that the number of channels chosen by the agents rise from 1 to 1.5 (both probabilities of transmitting in the shared channel \( p_1 \) and \( p_2 \) rise from 0 to 0.5) as \( \alpha \) increases from 0 to 1.

Figure 4.5c illustrates the correlation between the mean squared error and the channels chosen by the agents for the protocols generated by algorithm 2. As was expected, the error decreases as the number of channels increase, reaching a minimum when the number of channels used by each agent is around 1.5. It is also possible to compare the functions drawn by the Pareto frontier for both agents with the trivial protocols of table 4.7. For the extreme cases \((\alpha = 0, \beta = 1 \text{ and } \alpha = 1, \beta = 0)\), the protocols generated have a similar error-channels used ratio than the trivial protocols 1 (transmitting without using the third channel) and 3 (transmitting in the third channel with probabilities \( p_1 = p_2 = 0.5 \)), respectively. Trivial protocol 1 has null probabilities of transmitting in the shared channel, exactly like the Q protocol.
generated for $\alpha = 0, \beta = 1$, and for trivial protocol 3 the average of used channels by each agent is 1.5, equal to the Q protocol generated for $\alpha = 1, \beta = 0$. Regarding the intermediate protocols computed, they have a better error-channels used tradeoff than trivial protocol 2 (transmitting in the shared channel with probabilities equal $p_1 = p_2 = 0.9$).

Another point in favor of the protocols generated by algorithm \textbf{2} is the cooperation between the agents. It was vital, in order to obtain good results, that the protocols would make the agents use the communication medium equally, i.e., for each protocol the probabilities of transmitting in the shared channel were equal. Observing Figures 4.5b and 4.5c, we can conclude that the goal was achieved, as the number of channels used by the agents in each protocol is almost identical.

Analyzing the results for algorithm \textbf{4} from section 4.2.2, it can be concluded that the Q-learning approximation with linear parametrization of algorithm \textbf{4} was not able to fully obtain the desired results, given that none of the protocols generated were able to minimize the mean squared error when $\alpha \rightarrow 1$. Observing Figures 4.6a and 4.6b, we can see that the quadratic error of the protocols generated decreases and that the channels used by both agents increase as $\alpha$ increases. However, it is also observed that, for the extreme case $\alpha = 1, \beta = 0$, the algorithm cannot produce a protocol that makes the agents transmit in the shared channel with probabilities equal to 0.5, achieving a maximum of only approximately 0.25, thus not achieving the minimum mean squared error desired.

The same problem is illustrated in Figure 4.6c. The functions drawn by the Pareto frontiers have approximately the same error-channels used ratio presented by algorithm \textbf{2} in section 4.2.1 for half of the protocols, from $\alpha = 0, \beta = 1$ to $\alpha = 0.5, \beta = 0.5$. However, as alpha increases from 0.5, the results were far from the expectations. The protocols computed were unable of increasing the number of channels used by the agents to 1.5, thus decreasing the error to its minimum value, as we can see in a zoomed image of Figure 4.6c, displayed in Figure 4.9.

As shown in Figure 4.9, we can observe that algorithm \textbf{4} was unable to generate protocols that would minimize the error function 4.8. The maximum number of channels is far from the desired 1.5 for each agent, thus leading to the conclusion that algorithm \textbf{4} could not fully approximate the Q-function.

Other negative aspect regarding the results of this linear parametrization approximation is that, in comparison to the results of the neural networks approximation of algorithm \textbf{2}, the protocols generated by algorithm \textbf{4} could not achieve such a good cooperation between the agents. As we can see in Figures 4.6b and 4.6c, in some protocols, the difference between the probabilities of transmitting in the shared...
channel can reach to 0.04, which can make a great impact over time.

4.2.3.1 Rebuilding Algorithm 4

The protocols generated by Q-learning algorithm approximated with a linear parametrization applied in $NCS_2$, algorithm 4, showed different results than the ones expected initially. We assumed that the problem could be related with some parameters of the algorithm, and thus we tried to rebuild it.

Observing Figure 4.6b we can see that the number of channels used by the agents increases as the weight parameter $\alpha$ increases, reaching a maximum of around 1.25 channels. This is a evidence that the agents “understood” that there is advantage on augmenting the transmission rate, just did not “learn” enough to reach the optimal maximum transmission probability of 0.5. This could be a sign that the learning rates, $\alpha_{n+1}$ and $\gamma_{n+1}$ (see table 4.8), decrease too fast, affecting the learning process of the agents.

A way to understand how the change of the learning rates affected the learning process was to draw the evolution of the Q approximation error, $\beta \cdot Q(s_{k+1}, \phi_k, \theta_k) - Q(s_k, a_k, \theta_k)$, during the simulation. If at a certain time, this error stops decaying and the learning process is not complete, it can be a sign that the learning rates drop too fast. On the other hand, if the error is oscillating too much during the simulation, could be a sign the learning rates are too high and not decaying fast enough, leading to the divergence of the algorithm. With this in mind, the learning rates were changed from $\alpha_{n+1} = \frac{1}{n+1}$, $\gamma_{n+1} = \frac{1}{n^0 \cdot n^0 + 1}$ to $\alpha_{n+1} = \frac{1}{n^0 \cdot n^0 + 1}$, $\gamma_{n+1} = \frac{1}{n^0 \cdot n^0 + 1}$. After running algorithm 4 with different learning rates, these were found to be the best values that would make the error decay, also avoiding its oscillatory behavior. The results did improve, but not enough for the agents to reach the optimal maximum probabilities of transmitting in the shared channel $p_1 = p_2 = 0.5$.

Another problem that was identified was the high variance present in the Q approximation error as a function of the simulation, effect of the big range of values that the Q values took. Stochastic approximation algorithms often suffer from variance problems, which makes it hard for these algorithms to converge. In order to reduce the variance, the control variates (CV) technique (see section 3.4.2.1) was applied to algorithm 4. This modification lowered the variance of the Q approximation error by a factor of 10. We compare on Figure 4.10 the Q approximation error for algorithm 4 with and without the control variates technique implemented. As we can see, the variance of the results was highly improved after the use of this method.

![Approximated Error agent2](image1)

(a) Q approximation error of agent 2 without CV

![Approximated Error agent2](image2)

(b) Q approximation error of agent 2 with CV

Figure 4.10: Effects of the application of the control variates technique to algorithm 4 for Q protocol generated with $\alpha = 0.9$ and $\beta = 0.1$.

The results of section 4.2.2 were all obtained for the features presented in 3.18 after all the changes done to algorithm 4 previously addressed in this section. Despite improving the convergence and results...
of the algorithm, these changes could not make the algorithm generate protocols with the maximum optimal probabilities of transmission in the shared channel for both agents $p_1 = p_2 = 0.5$. Observing the Q error approximation for the extreme case protocol ($\alpha = 1$ and $\beta = 0$), it suggests that the agent learns about everything it should learn in the first fifth of the simulation, as after it converges to zero, suffering only small oscillations. This led to the conclusion that we extracted the best possible results for this set of features 3.18, and the set of features was not able to fully approximate the action-value function as desired.

![Approximated Error Agent1](image)

Figure 4.11: Q approximation error of agent 1 for protocols generated by algorithm 4 with $\alpha = 1$ and $\beta = 0$

As initially explained in section 2.4.3.1, the choice of features is very important and delicate for linear learning methods, and a good initial choice can many times be the difference between success or failure of the learning process. This said, different features vectors were applied to algorithm 4. Given that the cost function associated with this system is always positive (see section 3.3.3), it was assumed that the approximated Q function $Q(s, a, \theta)$ would always have positive values. The following set of features were proposed:

\[
\psi_i(s, a^1) = \begin{bmatrix} (e_i^1)^2, (e_i^2)^2, (e_i^1 - e_i^2)^2, (e_i^1 - 1)^2, (e_i^2 - 1)^2, Co, 1, [0]_{35} \end{bmatrix}^T
\]

\[
\psi_i(s, a^2) = \begin{bmatrix} [0]_7, (e_i^1)^2, (e_i^2)^2, (e_i^1 - e_i^2)^2, (e_i^1 - 1)^2, (e_i^2 - 1)^2, Co, 1, [0]_{28} \end{bmatrix}^T
\]

\[
\psi_i(s, a^3) = \begin{bmatrix} [0]_{14}, (e_i^1)^2, (e_i^2)^2, (e_i^1 - e_i^2)^2, (e_i^1 - 1)^2, (e_i^2 - 1)^2, Co, 1, [0]_{21} \end{bmatrix}^T
\]

\[
\psi_i(s, a^4) = \begin{bmatrix} [0]_{21}, (e_i^1)^2, (e_i^2)^2, (e_i^1 - e_i^2)^2, (e_i^1 - 1)^2, (e_i^2 - 1)^2, Co, 1, [0]_{14} \end{bmatrix}^T
\]

\[
\psi_i(s, a^5) = \begin{bmatrix} [0]_{28}, (e_i^1)^2, (e_i^2)^2, (e_i^1 - e_i^2)^2, (e_i^1 - 1)^2, (e_i^2 - 1)^2, Co, 1, [0]_{7} \end{bmatrix}^T
\]

\[
\psi_i(s, a^6) = \begin{bmatrix} [0]_{35}, (e_i^1)^2, (e_i^2)^2, (e_i^1 - e_i^2)^2, (e_i^1 - 1)^2, (e_i^2 - 1)^2, Co, 1 \end{bmatrix}^T
\]

where $i = 1, 2$ represent the indexes for agent 1 and 2, $e_i^1$ and $e_i^2$ are the first and second components.
of the estimation error vector for each agent and $C_0$ is the collision buffer.

Note that the features in equation 4.2 have only positive terms, as the $e_1^i \cdot e_2^i$, $e_1^i$ and $e_2^i$ terms from 3.18 that could have negative values, have been replaced by $(e_1^1 - e_2^1)^2$, $(e_1^1 - 1)^2$ and $(e_2^1 - 1)^2$. Other small change was made to algorithm 4 regarding the parameter $\theta$. As explained in section 3.4.2, the Q value for each action is the dot product between the weights vector $\theta$ and the features vector $\psi(s,a)$. $Q(s,a,\theta_k) = \theta_k \cdot \psi(s,a)$. In order for the Q function to be positive, $\theta_k \cdot \psi(s,a)$ has to be positive too. This said, at every update of the $\theta$ parameter, if any of the indexes of the vector $\theta$ was negative, it would be equaled to zero, maintaining this vector with only non-negative values.

However, as we analyze the Q approximation error of algorithm 4 applied with these features, we can observe that it rises to astronomic values, and the algorithm diverges, as we can see in Figure 4.12a.

Other hypothesis considered was that the value function had factors powered to 4, instead of being only quadratic, thus the features in equation 4.3 were applied to algorithm 4. Similarly to features 4.2, the Q approximation error had very high values, and the algorithm diverged, as showed in Figure 4.12b.

\[
\psi^i(s,a^i) = \begin{bmatrix}
(e_1^1)^4, (e_2^1)^4, (e_1^1)^2, (e_2^1)^2, (e_1^1 - e_2^1)^2, (e_1^1 - 1)^2, (e_2^1 - 1)^2, 1, \ [0]_{40}
\end{bmatrix}^T
\]
\[
\psi^i(s,a^2) = \begin{bmatrix}
0, (e_1^1)^4, (e_2^1)^4, (e_1^1)^2, (e_2^1)^2, (e_1^1 - e_2^1)^2, (e_1^1 - 1)^2, (e_2^1 - 1)^2, 1, \ [0]_{32}
\end{bmatrix}^T
\]
\[
\psi^i(s,a^3) = \begin{bmatrix}
[0]_{16}, (e_1^1)^4, (e_2^1)^4, (e_1^1)^2, (e_2^1)^2, (e_1^1 - e_2^1)^2, (e_1^1 - 1)^2, (e_2^1 - 1)^2, 1, \ [0]_{24}
\end{bmatrix}^T
\]
\[
\psi^i(s,a^4) = \begin{bmatrix}
[0]_{24}, (e_1^1)^4, (e_2^1)^4, (e_1^1)^2, (e_2^1)^2, (e_1^1 - e_2^1)^2, (e_1^1 - 1)^2, (e_2^1 - 1)^2, 1, \ [0]_{16}
\end{bmatrix}^T
\]
\[
\psi^i(s,a^5) = \begin{bmatrix}
[0]_{32}, (e_1^1)^4, (e_2^1)^4, (e_1^1)^2, (e_2^1)^2, (e_1^1 - e_2^1)^2, (e_1^1 - 1)^2, (e_2^1 - 1)^2, 1, \ [0]_{8}
\end{bmatrix}^T
\]
\[
\psi^i(s,a^6) = \begin{bmatrix}
[0]_{40}, (e_1^1)^4, (e_2^1)^4, (e_1^1)^2, (e_2^1)^2, (e_1^1 - e_2^1)^2, (e_1^1 - 1)^2, (e_2^1 - 1)^2, 1
\end{bmatrix}^T
\]

(4.3)

where $i = 1,2$ represent the indexes for agent 1 and 2, $e_1^i$ and $e_2^i$ are the estimation error for each agent and $C_0$ is the collision buffer.

![Approximated Error agent 2 with features vectors 4.2](image1.png)

(a) Q approximation error of agent 2 for algorithm 4 with features vectors 4.2

![Approximated Error agent 2 with features vectors 4.3](image2.png)

(b) Q approximation error of agent 2 for algorithm 4 with features vectors 4.3

Figure 4.12: Effects of the application of the features vectors 4.2 and 4.3 to algorithm 4 for protocols generated with $\alpha = 1$ and $\beta = 0$
Chapter 5

Conclusion

Data networking technologies, like networked control systems, provide several benefits on linking data points, as networks enable remote data transfers and data exchanges among users, reduce the complexity in wiring connections and provide ease of maintenance, leading to many industrial companies showing interest in applying networks for remote industrial control purposes and factory automation.

The use of NCSs in modern day industry makes it important that more and more research is done to improve them. In this thesis, we tried to find a solution to one of the problems that affect NCSs, the loss of packets in the network channels. In order to do so, we implemented Q-learning algorithms based on [24] and [30] to two different NCS architectures. The goal was that these algorithms would generate data communication protocols that would rule the transmission rate at the NCSs, based in the redundant transmission property discussed in [1].

A great advantage provided by the production of the protocols using reinforcement learning in this thesis is that it is user friendly. By using a cost function 3.12 as a method of giving feedback to the agent(s) about the value of a determined action in a determined state, the user can choose the relevance given by the agent(s) to the estimation error and to the usage of the medium. By varying the weight parameters $\alpha$ and $\beta$ before running the algorithms, the user can generate protocols that privilege the reduction of the error, increasing the transmission rate, or vice-versa, or choose an intermediate value for both factors. In a dynamic control environment with a lot of variables that can affect the systems control, this is a very useful property.

In general, the protocols generated by the Q-learning algorithms with the neural networks and linear parametrization approximations showed good results, as the protocols were able to control the transmission rate of the NCSs. As $\alpha$ increased and $\beta$ decreased, the protocols generated would make the agent(s) use more channels to transmit the information, as expected. Besides this, for most cases the “Q protocols” had a equal or better error-channels used ratio than the trivial protocols simulated.

However, algorithm 4 was not able to produce protocols that would have the maximum optimal transmission rate for $NCS_2$, as explained in section 4.2.3. Given that this algorithm has a linear parametrization, can only approximate a linear function of the features and its convergence is very sensible to the initial choice of features, as showed in section 4.2.3.1. Comparing the results of both approximations for the cooperative game of $NCS_2$, it is clear that the algorithm with the neural networks approximation 2 generated better protocols than the algorithm with the linear parametrization 4. The universal property of neural networks make them a powerful tool to approximate any function, thus solving a great number of problems. However, applying them in reinforcement learning algorithms has several problems, like the long time and great number of experiences necessary to train the network, leading to large computational expenses. In comparison, the linear parametrization approximation algorithms based in [30] are much faster and computationally “lighter”. The choice of one or another approximation should be made
having in consideration the characteristics of the problem they ought to be applied to.

5.1 Future Work

The application of the Q-learning algorithm with neural networks approximation (algorithm 2 from section 3.4.1) to design protocols for NCS2, based on the redundant transmission studied in [1], has showed good results. We obtained protocols that optimized the estimation error - cost of communication tradeoff for both systems and made the agents cooperate in sharing the communication resources. The main highlights of our approach are:

- Adaptability to different control contexts (it was shown in this thesis by varying the weights $\alpha$ and $\beta$ before running the algorithms).
- We were able to obtain good results without the need to have a centralized control (the agents were able to cooperate without knowing the other agent’s state, or without a central controller ruling both systems transmission rate).
- It is model-free, i.e., algorithm 2 can be adapted to different NCSs.

Our work represented the first application of the algorithms based in [24] and in [30] to develop communication protocols in networked control systems. For this reason, the NCSs studied had a quite simple structure, and the protocols obtained were compared to trivial protocols. Our approach, specially the algorithms based in the DQN algorithm in [24], proved to have good results for NCS1 and NCS2. An obvious follow-up of our work would be applying this method to different and more complex NCSs, that introduce problems where the optimal protocols are not easy to find.

In this thesis, we used identical control systems in order to simplify the NCSs studied. However, future work may include applying algorithm 2 to NCSs with different systems to control, e.g., if the systems to control are not identical, and one requires an estimation error lower than the others (in this case the protocols generated have to “give priority” to one of the systems, and the usage of the medium for the agents has to be different). This may be achieved by changing the cost function, assigning a greater weight to the agents mean squared error. This way, the protocols obtained will, theoretically, make the priority agent use the communication resources more often than the other agents.

There are lot of advantages in applying algorithm 2 to different and more complex NCSs in the future. The fact that this method obtained good results without the need for centralized control is a big plus. Its appliance to NCSs with a lot of remote autonomous agents, e.g. autonomous mobile robots acting in environments dangerous for human, where the communication to a central controller is difficult or impossible, may be promising, as it may optimize the communication between them, making them cooperate, even if they are not aware of the other agents states.


