Follow this and additional works at: https://knowledgecenter.ubt-uni.net/etd

Part of the Computer Sciences Commons
INTELLIGENT SYSTEMS FOR INDUSTRY USING REINFORCEMENT LEARNING TECHNIQUE
Bachelor Degree

Lirandë Pira

July / 2016
Prishtine
INTELLIGENT SYSTEMS FOR INDUSTRY USING REINFORCEMENT LEARNING TECHNIQUE
Bachelor Degree

Lirandë Pira

July / 2016
Prishtine
INTELLIGENT SYSTEMS FOR INDUSTRY USING REINFORCEMENT LEARNING TECHNIQUE

Supervisor: Krenare Pireva
Co-supervisor: Vincenzo Piuri (University of Milan)

July / 2016

A thesis submitted in partial fulfillment of the requirements for the degree of Bachelor of Computer Science and Engineering
Declaration

Prishtine, 20/07/2016

I Lirandë Pira hereby declare that this thesis is my own original work, except the parts that are documented as required according to accepted citation and reference rules.

Therefore, I understand what plagiature is and I declare that I accept all the consequences and actions that can be taken by my supervisor as the status of UBT defines it, in case in this thesis plagiature is noticed.

Name, surname and signature

Lirandë Pira
ABSTRACT

The rise of Intelligent Systems has happened gradually, then suddenly. They are gradual because we are aware that this field of computing has come a long way along with the history of computers. Yet, the sudden astonishing changes that affect mankind seem to take everyone in surprise. Their occurrence is reshaping the real world and our interaction with our digital life is changing in profound ways. Can computers think? We don’t have evidence on that, whatever the answer to that question is. But what we know is that computers do learn. Indeed, the whole process of computer evolution revolves around machines that are able to follow instructions and practice and eventually get better at what they are initially produced to accomplish. Consequently, the questions that we try to answer are related to the types of learning that intelligent programs use with special regards to one of the most researched methods of Machine Learning – Reinforcement Learning. On the other hand, it is crucial to apply the intelligent self-learning machines in industry, environment, enterprise, medicine and all the other sectors where we need to see the substantial changes that correspond with the era of machines that can learn. The intersection point in this research is the application of intelligent programs in industry using a very specific learning technique – Reinforcement Learning.
ACKNOWLEDGEMENTS

I am grateful to my parent’s unceasing support and encouragement before, during and after writing this thesis.

I wish to express my sincere thanks to the supervisor Krenare Pireva and co-supervisor Vincenzo Piuri, for providing me with their sincere and valuable guidance throughout this research.

I also place on record, my sense of gratitude to everyone, who directly or indirectly, has lent their hand in this venture.
TABLE OF CONTENTS

LIST OF FIGURES ......................................................................................5

LIST OF TABLES .......................................................................................5

ABBREVIATIONS ......................................................................................6

1 INTRODUCTION ...................................................................................7

1.1 Brief definition of Intelligent Systems ......................................................7
1.2 Entering the Era of Machines that can Learn ...........................................8
1.3 Briefly on Reinforcement Learning ..........................................................9
1.4 Goal and Structure of Thesis .................................................................10

2 LITERATURE REVIEW .........................................................................12

2.1 Artificial Intelligence and Intelligent Systems ...........................................12
  2.1.1 Sub-domains of Artificial Intelligence ..............................................15
  2.1.2 Characteristics of Intelligent Systems ..............................................16
2.2 Machine Learning for Intelligent Systems ..............................................17
  2.2.1 Supervised Learning ......................................................................19
  2.2.2 Unsupervised Learning ..................................................................20
  2.2.3 Reinforcement Learning ..................................................................20
2.3 Reinforcement Learning Model ..............................................................21
  2.3.1 Exploration and Exploitation ..........................................................24
  2.3.2 Value Functions ............................................................................25
  2.3.3 Markov Decision Processes ............................................................26
  2.3.4 Applications of Reinforcement Learning .........................................27

3 PROBLEM DEFINITION ......................................................................30

3.1 Intelligent Systems interacting with the Environment .............................30
  3.1.1 Tic-Tac-Toe Example ....................................................................31
  3.1.2 Multi-Agent Environment: Rock Paper Scissors Example ..................32
  3.1.3 Turing Test .....................................................................................32
  3.1.4 Chinese Room Experiment ............................................................34

4 METHODOLOGY .................................................................................35

4.1 Reinforcement Learning Algorithms .....................................................35
  4.1.1 Dynamic Programming ..................................................................35
  4.1.2 Monte-Carlo Model ......................................................................37
  4.1.3 Temporal-Difference Learning .........................................................37
  4.1.4 Q-Learning Model .........................................................................38
LIST OF FIGURES
Figure 1: Machine learning basic environment scheme ........................................ 18
Figure 2: Agent-Environment interaction scheme ................................................. 22
Figure 3: Sample environment that represents the trial-and-error learning .......... 23
Figure 4: Exploration-Exploitation dilemma in the environment of the agent .... 24
Figure 5: A simple graphic representation of an MDP ............................................. 26
Figure 6: Tic-Tac-Toe Example of Agent-Environment Interaction .................. 31
Figure 7: Turing Test environment scheme ......................................................... 33
Figure 8: System Architecture of the components of the quadrotor ................. 50
Figure 9: Identified actions of the quadrotor and their possible behaviors ........ 52
Figure 10: The dimensions of quadrotor’s actions ............................................. 53
Figure 11: RL algorithm used for policy iteration ................................................. 55
Figure 12: Parrot AR.Drone system and control .................................................. 56
Figure 13: The GUI of the MATLAB quadcopter control toolbox .................... 58
Figure 14: Response in rz direction using RL controller ..................................... 59
Figure 15: An explanation of PGSD model-free algorithm ................................. 61
Figure 16: The architecture of the NI myRIO-1900 device used in implementing the PGSD RL algorithm in quadrotors movement ............................................. 61
Figure 17: The result of a model-free RL algorithm in controlling the attitude of the quadrotor ................................................................. 62

LIST OF TABLES
Table 1: The possible outcomes of two opponents in the Rock-Paper-Scissors ........ 32
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>IS</td>
<td>Intelligent Systems</td>
</tr>
<tr>
<td>RL</td>
<td>Reinforcement Learning</td>
</tr>
<tr>
<td>MDP</td>
<td>Markov Decision Processes</td>
</tr>
<tr>
<td>DP</td>
<td>Dynamic Programming</td>
</tr>
<tr>
<td>TD</td>
<td>Temporal Difference</td>
</tr>
<tr>
<td>GPS (Ch2)</td>
<td>General Problem Solver</td>
</tr>
<tr>
<td>GPS (Ch5)</td>
<td>Global Positioning System</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>QA</td>
<td>Question Answer</td>
</tr>
<tr>
<td>DAGs</td>
<td>Directed Acyclic Graphs</td>
</tr>
<tr>
<td>GM</td>
<td>Graphical Models</td>
</tr>
<tr>
<td>ASIMO</td>
<td>Advanced Step in Innovative Mobility</td>
</tr>
<tr>
<td>DARPA</td>
<td>Defense Advanced Research Projects Agency</td>
</tr>
<tr>
<td>VDBE</td>
<td>Value-Difference Based Exploration</td>
</tr>
<tr>
<td>SARSA</td>
<td>State-Action-Reward-State-Action</td>
</tr>
<tr>
<td>ANN</td>
<td>Artificial Neural Networks</td>
</tr>
<tr>
<td>KBRL</td>
<td>Knowledge-Based RL</td>
</tr>
<tr>
<td>DRL</td>
<td>Deep Reinforcement Learning</td>
</tr>
<tr>
<td>DDoS</td>
<td>Distributed Denial-of-Service</td>
</tr>
<tr>
<td>POI</td>
<td>Points-of-Interest</td>
</tr>
<tr>
<td>PGSD</td>
<td>Policy Gradient via Signed Derivative</td>
</tr>
<tr>
<td>LQR</td>
<td>Linear Quadratic Regulator</td>
</tr>
<tr>
<td>VTOL</td>
<td>Vertical Take-Off and Landing</td>
</tr>
<tr>
<td>IMU</td>
<td>Inertial Measurement Unit</td>
</tr>
<tr>
<td>ESC</td>
<td>Electronic Speed Controllers</td>
</tr>
<tr>
<td>UAV</td>
<td>Unmanned Aerial Vehicles</td>
</tr>
</tbody>
</table>
1 INTRODUCTION

On the summer of 2014, I took a course on Machines-Humans-Robots conducted by Professor Minrou Asada whose lab was also designing autonomous humanoid robots to take part in the “RoboCup”, a Robot Soccer World Cup. Ever since 1997 this championship has been held in various different countries, where robots play soccer against humans. Every year nearly 400 teams from countries around the globe take part in this competition. As of 2014 robots hadn’t won any of the championships. But according to the data they were improving year by year. My instant question was “What if they win this time?” “Nothing, they have beaten us at different things before” was his answer. “The idea is to make them good enough to ‘beat’ the humans so that we use their power in combination to ours” he further inquired.

The key idea that we need to remember from this story – the idea that we will further explore in this thesis – is “they were improving year by year”. The intelligent programs – robots in this case – build by Professor’s Asada’s lab, were learning, one crucial trait that stands at the core of Artificial Intelligence, Machine learning and thus, Reinforcement learning.

1.1 Brief definition of Intelligent Systems

The most accepted definition of Artificial Intelligence (John McCarthy, 1955) is the “study of design and intelligent systems”. Subsequently, Intelligent Systems are a new current of embedded systems connected to the internet that have the capacity to analyze and manipulate data. These intelligent programs have the capacity to respond to the world around them. Intelligent Systems come in many forms starting from iRobots automated vacuums, to fingerprint recognition, to Google’s self driving cars.

Human intelligence as we know is still at the core of Artificial Intelligence, which means that also AI is concerned with learning, reasoning, planning, processing and more. Dubbed as systems that know what they are doing (Brachman, 2002), nowadays Intelligent Systems
represent the fastest growth areas for processors. They are all around us and they are changing our lives and our worlds in profound and exciting ways.

The standard examples of intelligent systems that are wildly understood are computers, terminals, or servers. However, intelligent systems come in different forms, many of which are user-friendly and in interaction with people in everyday basis, such as: controlled traffic lights, quality control, financial market prediction, scheduling, household robots and more.

1.2 Entering the Era of Machines that can Learn

The rapid migration of devices into the Internet has led to the growth of a couple of big concepts trending in the industry including Internet of Things (IoT), Big Data and Machine Learning.

- Internet of Things allows machine-to-machine communication where each device is connected to the internet by possessing an Internet Protocol (IP) address. The cloud-based architecture allows machine-to-machine communication – also called the Internet of Everything – represents one gigantic project where all the humans, machines, devices, gadgets and more, connect to the Internet which enables them to communicate and share data.

- Big Data is a term coined to represents the large amount of data currently available on the Web. According to Auto-ID center at MIT, the Web consists roughly of 50 petabytes (1 Petabyte = 1024 Terabytes) of data available. This is why big data is the key to the learning process. It implies repetitious time an intelligent program goes through before it gets really good at what it is programmed about. The same is true for the best programmed computers with massive data processing power and top notch architecture.
• The two concepts are highly interconnected to each other and so is the third one, Machine learning. Data alone is not enough; it needs an infrastructure to process it in real time. By designing better algorithms. Literally any field in which you need to manipulate with data is a potential client of machine learning.

What we are witnessing now is the new global trend of devices that allows adopted individual technology for every person or business. Personalization as an attribute is a key concept where the machine learning is being applied.

1.3 Briefly on Reinforcement Learning

Reinforcement Learning, our main focus on this thesis, is one approach in machine learning that is largely defined as learning while interacting with the environment. Reinforcement learning it is recognized to have been inspired by behaviorist psychology, in which agents act without being explicitly told what to do and later receiving “feedback” known as reward. The agent has to decide which is the best choice based on its existing state. It involves a process of learning from exploration and exploitation, which we will discuss later.

As we mentioned its influence from other fields, Reinforcement learning has been available in the psychology of animal learning and also in the optimal control concerned with dynamic programming using value functions. The most recent application of Reinforcement Learning has been using the temporal difference methods. Reinforcement learning in regards to the application in intelligent programs and the vast field of machine learning as we know it today – which is when the three types of areas merge – dates back in the 1980s (Sutton & Barto, 1997) and signifies the modern field of Reinforcement Learning.

The idea behind reinforcement learning is massive. The essence of it is equivalent to the idea of designing human beings in a laboratory in the field of medicine. It is about
developing a whole thinking mindset and this is when better and better algorithms are needed.

### 1.4 Goal and Structure of Thesis

The general aim of this research on Intelligent Systems for Industry using Reinforcement Learning Technique is to give a fundamental view on the particular method of Machine Learning known as Reinforcement Learning. As the aim of Machine Learning is to make intelligent programs through the process of learning while interacting, Reinforcement is one model that can be taken into account. Reinforcement is a semi-discovered area which requires the development of the autonomous part of learning, thus making Reinforcement Learning one of the most researched and most unpredictable methods of Machine Learning.

It is unpredictable in the sense of individuality that each Intelligent System is able to show, yet conventional because it studies the underlying aspects of the responsiveness of Intelligent Systems as a whole. Reinforcement learning as we know has existed ever since human race exists. It is only natural for mankind to learn by interacting with the environment, legacy which we are trying to pass on to intelligent programs we make.

This thesis is structured in an approach to give relevant information on the field of Artificial Intelligence and Intelligent Systems, from a more generalized overview, continuing to specific related details. Thus, the reader will easily follow the flow of the information presented.

The informative approach of the thesis together with the relevant examples while presenting some of the most domineering aspects in the field, gives an essential view on Intelligent Systems using Reinforcement Learning as a technique.

The Literature Overview (2) is introductory and informative which will bring thorough understanding on the main distinguishing features of Artificial Intelligence, Machine Learning and Reinforcement Learning, as at the same time are the key words of this thesis. This section gives a more detailed survey on Reinforcement Learning introducing the trial
and error learning and delayed reward learning. An introduction on the tradeoff between exploration and exploitation, Value Function, Markov Decision Processes is also provided which lays off the basis for further explanation on the Algorithms and Models of RL. In this section we also give an overview of the potentially applicable fields of RL, as well. In Problem Definition (3) we will present the problems on interaction of agents with the environment which brings the underlying questions on Reinforcement Learning. A profound analysis of single-agent and multi-agent environment will also be provided. The Methodology (4) presents the most important elementary possible solutions such as temporal-difference learning and Q-Learning model. Results (5) present a case study of Quadrotors autonomous movement studied by different papers so far. Finally, the Conclusions and Discussion (6) gives a summary of the thesis and the potential impact overview of the machines we will study throughout this research.
2 LITERATURE REVIEW

2.1 Artificial Intelligence and Intelligent Systems

Artificial Intelligence is an area with a rich history yet constantly altering and growing. Artificial Intelligence is mainly about expanding human capabilities beyond our imagination with its abundant applications and enormous possibilities. Intelligence is defined as the “The ability to acquire and apply knowledge and skills” (Oxford Dictionary). The goal of Artificial “Intelligence” is no different.

Answering the question when did AI started to develop, we should look back a long time ago. On this note, speaking about the history of AI is speaking about the history of the whole computing industry, since the simple calculators designed in the 17th century or other devices before that represent the first attempts to design computers known as intelligent systems studied by the field of Artificial Intelligence. However, let us look back on the main events of AI development. In the 5th century BC Aristotle designed the syllogistic systems the first formal deductive reasoning systems. After a long dormant period in the 15th – 16th century we find the printing using movable types and clocks as measuring machines. 1642 Pascal invented the first mechanical calculator with the addition as the only operation. Later in 1671 Leibniz made improvements in the calculator by adding multiplication and division. 19th century brings Charles Babbage who designed Difference Engine – known as the first successful automatic calculator – which could perform simple mathematical calculations and later the Analytical Engine that could carry out more complicated calculations, with some suggestions from Ada Lovelace. In the same century George Boole proposed the logic theory known as Boolean algebra on which the whole logic of integrated systems in microprocessors is built on nowadays.

The 20th century brings the most significant achievements in AI. Emil Post developed a programming language without thinking of a machine in which it could be implemented. Alonzo Church – Alan Turing’s supervisor – wrote Introduction to Mathematical Logic which comprises some of his earlier remarkable results. Alan Turing is undoubtedly the one
figure that founded the fundamental principles which are required to prove the evidence of artificial intelligence. Turing test, which we will discuss later, remains the biggest challenge for the existence of Artificial Intelligence. During the 1943-1945 J. Presper Eckhert and John Mauchly build the first general-purpose electronic digital computer known as ENIAC. 1956 was the year in which the Dartmouth conference was held. It brought accomplished computer scientists such as John McCarthy, Marvin Minsky, Nathaniel Rochester, Claude Shannon etc., as evidenced. The proposal of John McCarthy included the discussion of computers, natural language processing, neural networks and theory of computation among others. Nowadays this conference is known to be the seminal event for the Artificial Intelligence as a field.

After the Dartmouth Conference many institutions such as MIT and Carnegie Mellon University began to form AI research centers. In 1957 Newel-Shaw-Simon developed the General Problem Solver (GPS) which was meant to work as a machine that can solve any problem. In 1958 John McCarthy developed LISP, a programming language that stands for “List Processing” since he believed in using a data structure made of lists in code and also in data. Since the language was heavy in machinery it was better used in the 1970s with the more advanced technology of that time.

The time when the AI takes leaps in development is around the 60s when the General Motors develops the first industrial robot by Joseph F. Engelberger, who is known to be the Father of Robotics. In 1963 DARPA and MIT signed a 2.2 million dollar grant to be used in Artificial Intelligence advancements. Over the next two decades Japan took over in building certain robots that could be used to perform human tasks. In 1966 MIT build ELIZA, the first computer program that tried to communicate in the natural language by modeling the behavior of the psychiatrist. In 1964 the fuzzy logic was proposed by the University of Berkeley. Japan was the first country to use the fuzzy-logic based automatic train operation control systems. In 1987 Japan build the first subway system using fuzzy logic. Soon enough it was embraced by other countries and industries as well. Today, almost every intelligent program has the fuzzy logic-based type of logic implemented.
In the 1970s starts the development of Expert Systems such as DENDRAL (developed by the Soviet Union), MYCIN (Stanford University) and PROSPECTOR (developed by NASA). As for the Neural Networks they weren’t much developed until the 80s because of the large processing power they required. Another highlight in the field of AI is the occurrence of the Evolutionary Computation which involves four main domains: genetic and evolutionary algorithms, evolutionary programming, evolution strategies and genetic programming. Another significant achievement is in the field of computer games. In checkers Chinook beats the world champion Marion Tinsley in 1994 and Deep Blue beat Gary Kasparov in chess in 1997.

As we can see, the history of computing and that of AI carry on simultaneously, however when the term Artificial Intelligence was coined it was used to describe the computers that were very much alike to human intelligence in resemblance. As the Data Scientist Jeremy Howard said “Your previous understanding of what is possible is different.”

While we have presented a brief history of the AI as a field and its main developments, it is important to now have an overview of the applications of AI.

- **Robotics and Robot Motion Planning.** The two fields are concerned with constructing smart agents capable of dealing with human or non-human like activities. In machine learning this is the type of intelligent systems we usually talk about. While the Robot Motion Planning is concerned with the movement of robots and moving patterns that could satisfy different constrains of movement and potentially advance the movement.

- **Image Processing** involves the manipulation with data that comes in forms of image formats (.jpg, .png, .mng etc.) while using mathematical calculations. The most recent example of this is the Google’s image search which matches data that comes in forms of image extensions. Image Processing allows the information to be interpreted from image as input to another image as output (the output may also the other data type format).
Natural Language Processing is mainly concerned with the interaction between computer languages and natural/human languages. This is also a field that entirely uses Machine learning algorithms to reach its purpose of understanding the environment.

2.1.1 Sub-domains of Artificial Intelligence

The sub-domains presented above at the same time introduce the major problems AI is facing nowadays.

- Knowledge Engineering: Concerned with producing knowledge based systems which will try to solve complex problems by using knowledge and reasoning.
- Automatic Theorem Proving: The usage of computer programs to prove mathematical theorems.
- Machine Learning: Producing intelligent programs that enhance over time while interacting with the environment (we will discuss in the sections below).
- Heuristic Search Methods: Information on the closest node which may lead to the best solution.
- Computer vision: Computer field that is concerned with image understanding potentially in 3D by processing and analyzing data.

The modern approaches towards dealing with the AI issues that are being used nowadays are:

- Neural Networks or often called Artificial Neural Networks present a model of data processing, inspired by the brain neurons, to process data that depend on large inputs. It is composed of many nodes (neurons) linked together to perform a specific purpose such as data classification or pattern recognition. The first artificial neuron was produced in 1943 by the neurophysiologist Warren McCulloch and the logician Walter Pits, but Neural Networks Simulations appear to be a recent development since the weak technology did now allow the usage of these advancements. Nowadays, they are used in finding patterns in data that humans or other technological advancements cannot find, thus they are considered expert systems on
this field. The learning methods for adaptive Neural Networks are the ones used by Machine learning: Supervised, Unsupervised and Reinforcement Learning.

- **Bayesian Networks** are part of the Graphical Models (GMs) where each node represents a variable which the edges between the nodes represent the possible dependencies between the nodes. These dependencies are measured by using statistical and computational methods. Undirected Graphs are called Markov networks while the Directed Acyclic Graphs (DAGs) is where the Bayesian Networks belong to.

- **Evolutionary Algorithms** are based on the powerful principle of evolution: survival of the fittest which represents a very interesting field in the modern heuristic algorithms. Evolutionary Algorithms represent the field present in the Artificial Evolution. They use mechanisms motivated from the Darwinian evolution such as reproduction, selection, recombination and mutation.

### 2.1.2 Characteristics of Intelligent Systems

What we are going to discuss in this section is what makes these typically known programs intelligent systems. To answer that question please find below an overview of the commonly accepted definitions of intelligent systems.

As we mentioned previously, Intelligent Systems present a type of embedded-systems that are connected to the internet and have the capacity to manipulate large complex data. However, whenever we refer to an “Intelligent System” or “Intelligent Program” the following attributes are valid for each of them.

An Intelligent System has to posses the following attributes:

- The ability to extract and store knowledge
- Possess the human like reasoning (analysis) process
- Learning from experience (or training)
- Finding solutions through processes similar to natural evolution
2.2 Machine Learning for Intelligent Systems

The reason we need to briefly cover the Machine Learning section as well, is because as it is explained in the Goal and Structure of Thesis, we provide the top down approach, moving from generalization towards the specifics.

Machine learning, often called “programming by example” is the field of study that is concerned with building computer programs that can enhance over time due to experience while in interaction with the environment. Therefore, the main goal of machine learning is to design learning algorithms that can be used from computer programs to learn over time without the intervention of humans. In a further explanation, rather than writing software or programs that enable the computer to perform a task, the goal of machine learning is to find methods which will allow the computer to come up with programs on its own, based on the examples we provide. Learning is defined as “the acquisition of knowledge or skills through study, experience, or being taught” (Oxford Dictionary). The similar definition, as we explained, holds for the machine learning as well. Learning, in machine learning consists of using sophisticated learning algorithms based on which the intelligent program can produce an output. Machine learning is about building the artifacts and also about the math, science, computing, engineering behind it. Machine learning as a field stands at the core of Artificial Intelligence due to its incredibly large variety of applications in technology, medicine, transportation, military, advertising etc.

The idea that computer programs can learn on their own, while given specific tasks, dates back in the early 60s. On this note, in 1959, Arthur Samuel described machine learning as a "Field of study that gives computers the ability to learn without being explicitly programmed". A note on Arthur Samuel, back in 1958 he was trying to make a computer beat himself at checkers, so what it did was let the machine pay against itself for hundredth of times and learn how to play. By 1962 that computer had beaten the Connecticut state champion. What was received out of this example is that, given proper algorithms and tasks, computers can learn.
To give a familiar recognition of machine learning in order to make its understanding less abstract, allow us to mention a couple of everyday examples of machine learning applications:

- Spam recognition
- Fingerprint matching
- Weather prediction
- Face recognition
- Exchange rate predictions
- Personalized sites
- Search rankings
- Shape detection
- Speller correction

The basic understanding learning environment scheme of machine learning can be found below in Figure 1.

![Figure 1: Machine learning basic environment scheme](image)

In 1997, Tom Mitchell further expanded the definition of machine learning by indentifying the actors in the learning process: “A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E”. Thus, the learning agent is characterized by the subsequent elements:

- Task T
• Experience E
• Performance measure P

One of the core problems in Machine learning is generalization. We assume that in a well behaved function – where the data given as input and as output follows a pattern, or the same order – we will be able to induce information, as in move from the specific examples to generalization.

Most of the biggest companies in the field of technology try to include machine learning in their application. Think of Google’s category mail filtering or self-driving cars, Amazon’s product suggestions or Facebook’s recognition of faces in pictures.

Machine learning at the same time presents the field which computer skeptics find to be beyond our time and necessity. In other words, according to doubters, it represents the time when robots start to “take over”. This is a coherent debate and with every passing year becomes even more relevant as the power of intelligent programs grows exponentially. However, the majority of Data Scientists suggest that rather than competing with intelligent programs, we should collaborate in order to maximize the results. With Machine Learning flourishing rapidly every year, one thing is for sure, evolution doesn’t stop with humans.

Finally, as machine learning is concerned with producing intelligent programs through a process of learning and evolving, there are a couple of learning approaches as we discuss below.

2.2.1 Supervised Learning

As the name suggests, supervised learning is supervised by a “teacher” that evaluates the behavior of the computer program after having given a task or an input and received an output. It is mostly used when we (humans) know how to classify the data we just need to sort it in order to test the classifier (agent) by giving data sets; so that this step can be repeated in the future with no particular trouble. This is how the agent learns.
The most common example referred to while explaining supervised learning is the teacher – student analogy. After a student takes an exam, the teacher marks it and gives a grade to the student. In machine learning terms, the exam paper represents the input, the completed exam paper is the output and the final grade represents the reward or the evaluation. Real life examples of supervised learning are spam filtering, voice recognition etc.

2.2.2 Unsupervised Learning

In unsupervised learning no teacher is available. The agent after being given data finds patterns and makes relationships herein. It’s about receiving a collection of data and figuring how you might divide it in whichever way. All of the ways we divide the data is equally good because there are no explicit instructions – because of the lack of the supervision – how should the data be classified. The common example given to explain unsupervised learning is learning to joggle. As no teacher is present the subject tries out various times to keep the objects in the air until it does so.

Among the most common unsupervised learning methods is cluster analysis. The agent is provided a number of data in order to find the hidden patterns. Of course it will take a while until the agent learns to recognize the similarities in patterns. Thus, in unsupervised learning there are no rewards or punishments.

2.2.3 Reinforcement Learning

Often categorized as a type of unsupervised learning, reinforcement learning is about learning while interacting with the environment thus updating oneself after every punishment or reward. When introducing Reinforcement Learning, a famous quote by Russell and Norvig from Introduction to Artificial Intelligence is often introduced: “Imagine playing a new game whose rules you don t’ know; after a hundred or so moves, your opponent announces, ‘You lose’”. 
Some literatures also mention the Semi-Supervised Learning (Scudder, 1965; Fralick 1967), which is concerned with learning from unlabeled and labeled data as well. This is mainly practiced in Deep-Learning, where some layers have labeled data (supervision is present) and others don’t (supervision is not present). It is used due to the arguably high cost it takes for the data to have been previously labeled, and making full usage of the unlabeled data.

As the main idea of this research is to present reinforcement learning in moderate detail, more can be found below.

**2.3 Reinforcement Learning Model**

We have presented the main idea behind the RL in previous sections of this thesis but in this section we try to explore the field and findings from a close-up computational perspective. In addition to the previous definitions Reinforcement learning represents the fundamental way humans and other beings learn. We are presented to different tasks in an unfamiliar environment where we make choices based on time-bound conditions. The idea of Reinforcement Learning, as many other ideas in AI, is copied from the nature. In this simplified version of the aim of RL we can conclude that the overall aim is to design algorithms for self-learning agents like us.

Before we dive in depths of the topic, let us present below the most important terminology used in RL:

- **Agent**: Represents the learner which is also the decision maker.
- **Environment**: The external system that the agent is embedded/placed in.
- **Action**: The set of achievements of what the agent can do.

- **Task**: The specification of instances in the RL problem.
- **State**: The possible changes in the environment the agent receives after each task.
• **Reward**: The feedback (rewards or punishments) the agent receives after each action.

![Agent-Environment interaction scheme](image)

**Figure 2: Agent-Environment interaction scheme**

Components of the Agent:

- **Policy**: It maps the situations to actions by using the decision making control strategy.

- **Reward function**: Determine the usage of the planning tools over the agent’s actions. This includes the possible rewards or penalizations (also included in the “reward” term) that the agent might receive.

- **Value function**: Measuring the reward of a possible action. The value of a state represents the value of a long term result achieved from that state under a particular policy. The agent learns the policy by learning the value functions (trial and error method), i.e., after receiving the feedback. It maps the states to real numbers that represent the reward.

- **Model of the environment**: it is optional in some cases however it represents the agent’s view of the environment which maps states and actions (with the usage of Value Functions) to probability distributions over states.

An intuitive way to understand the relation between the agent and the environment is to consider the following conversation (Kaelbling, Littman & Moore, 1996).

**Environment**: You are in state 40. You have 6 possible actions.
Agent: I’ll take action 2.
Environment: You received a reinforcement of 7 units. You’re now in state 12. You have 4 possible actions.
Agent: I’ll take action 3.
...
There are two characteristics that make Reinforcement Learning different from the other types of learning: *trial-and-error* – since the agent is not told what actions to take instead it should try based on its own capacity of decision-making – and *delayed rewards* – a decision may affect not only the current action but a series of other actions too.

![Figure 3: Sample environment that represents the trial-and-error learning](image)

In the figure above we try to present a problem by visualizing it into a maze. As in any labyrinth, the agent has to find the way out. What is important to notice is that the agent is given an environment and a task: try to escape from the labyrinth. Whether the rewards are given to the agent for each correct move or only when it finds the way out, that is for the environment to decide. All these can be decided and uniformalized by the means of Reward Functions. Different penalties could also be added, such as giving minus (-) points every
times the agent hits the bricks or moves in the wrong direction. Another simulator would be to add more agents and give bigger reward to the first one that finds the way out.

In literature quite often are introduced the *episodic tasks* and *continuing tasks*. In the former one, the agent-environment interaction splits into a series of separate episodes, while in the second one, the interaction does not break down into smaller episodes (such as the example with the maze). However, in each of these tasks the purpose of the agent (RL learner) is to maximize the reward it achieves. It is mathematically easier to calculate the episodic tasks since one particular action affects only the number of reward received during episodes. Whereas in the continuing tasks, the number of all the tasks affects of the possible rewards the agent receives in the end.

### 2.3.1 Exploration and Exploitation

Let us consider the following example. Imagine a room in which there is one white ball and one black ball placed in two different parts of the room. Let us further assume that after several attempts, our agent has already found the location of the white ball in the room. However, it has been given a signal that notifies that there is one bigger object in the room, which at the same time means a bigger reward. What are the odds that our agent would pursue a new expedition – explore – to find the location of the black ball?

![Diagram](image)

*Figure 4: Exploration-Exploitation dilemma in the environment of the agent*
As the example explains it, exploitation is the set of actions which the agent has tried in the past from which has gained considerable results and has the possibility to try again. On the other hand, exploration is about the set of actions that the robot hasn’t tried in the past but might be willing to take for a bigger reward (or else).

In another hypothetical situation, imagine that our agent has found the location of the black ball before the white ball’s location. Would it pursue an expedition – explore – for a less big reward, just to satisfy the signals from the environment, or what we know in human terms as “curiosity”?

With this example we have slightly touched in one of the problems of reinforcement learning which is learning by interacting with sufficiently diverse environments and tasks.

### 2.3.2 Value Functions

Value Functions aim to measure the return of a particular action. In other terms they analyze how good an action will be in a particular state. As we can assume they are pair functions, thus we can use the following notation.

\[ V^\pi(s) \]  \hspace{1cm} (1)

Denotes the state \( s \) under policy \( \pi \). Thus, the value \( V \) is the expected return. This notation is supposed to happen before the agent takes any action. Once it does, based the chosen action it will chose the next actions as we present below. *We call this equation state-value function for policy \( \pi \).*

\[ Q^\pi(s,a) \]  \hspace{1cm} (2)

Under policy \( \pi \) and state \( s \), action \( a \) will take place. We will refer to this as the Q-value. *We call this equation action-value function for policy \( \pi \).*
Almost all of the Reinforcement Learning algorithms use Value Function approximation in order to determine the position of the agent in a particular state and measure its best performance that an action in a particular state brings.

### 2.3.3 Markov Decision Processes

In RL the agent makes its decisions based on the signals it receives from the environment’s state (recall the agent-environment conversation below). Therefore, it is important to know what kind of information to expect from the environment and what now. For this reason, we need to define a property of the environment and its state signals that is called the Markov property. A task that has the Markov property is called the Markov Decision Process (MDP).

![Figure 5: A simple graphic representation of an MDP](image)

In the figure above it is presented a simple MDP analysis of an easy problem with 3 states and 2 actions each. Each state can bring 2 possible actions and each action brings other new states. State and action is the terminology we have used in the RL as well. Each action is chosen based on the agent’s decision, at the same time bringing particular rewards. Each of the trajectories (arrays) in the figure represent the probabilities and the rewards the action brings, which are left blank in our example since they are unknown in reinforcement...
learning. The example provided in the figure is rather simple, however when numerous states are possible, reinforcement learning can be combined with function approximation.

In other words, the affects of an action taken in a particular state depend only on the current state and not on the a-priori history of that state. MDPs in essence provide the mathematical background of calculating the decision making process where the results depend partly on the decision maker and partly on the environment or other random factors. In addition, problems with delayed reinforcement are called Markov Decision Processes (MDPs). The agent’s actions determine the next state of the environment thus the state transition function – a notation in MDP – specifies the reward in terms of the current state and the next action. MDPs can be answered using linear programming or dynamic programming (we will talk about the latter in the next sections). When the state and action spaces are finite – as it is the case with the episodic tasks – it is called finite Markov Decision Processes (finite MDP). While the states and actions are continuous, the MDPs can better model the decision making process for a system that has continuous dynamics.

2.3.4 Applications of Reinforcement Learning

The field of Machine learning is suitable to be implemented in complex problems mainly that aim to perform in accordance to human behavior. Below there is a summary of these problems (Grosan & Abraham, 2011).

- Areas of knowledge where there is no human expert. In such fields the unpredictability is high and at the same time the learning rate is unknown. An example of this is the need to predict machine failures before they occur. In this area there are no experts which can a build a computer system. Instead a machine learning system can be build which studies the recorded data and predicts the failure by analyzing sensor readings.

- Areas of knowledge that have human experts but they are not able to explain their expertise. It cannot be explained in detailed steps how speech recognition or natural language processing happen. Instead machine learning agents can be build that are
given the input and output so it learns how to match the two ends – input and output.

- Active problems where events are changing rapidly. Often we need to make assumptions about the future based on a set of predictions such as for the weather forecast or the stock market. These kinds of assumptions require building a program that would analyze the situation on given data and make prediction about the future and also they require maintenance support frequently. Given the set of prediction rules, a machine learning program would automatically update the data and make the necessary predictions.

- Application that are customized or personalized for each computer. The simplest example is the e-mail filtering messages. A machine learning program can “remember” which type of e-mails a particular user rejects thus it repeats this step based on its learning algorithms. The filtering process is different for different users.

However, a variety of problems can be solved using Reinforcement Learning and it is mainly suitable to solve complex problems where there is no human expertise and no obvious solutions seem to be easily programmable. In this category, we find:

- Control problems: The main idea of which is to find an optimal policy. It can be used in the Elevator Scheduling type of algorithms, where the agent has to deal with two options usually, as it is the case with the disk drive options of read and write – the name of the algorithm dubbed from the elevator of a building where the options are going up or going down a building. It is often recommended that the agents spent time in simulated areas where they will find similar situations as in the real environment and thus explore good controlling behavior. It is convenient to use the RL agent for control problems since they can update their learning after each evaluative process.

- Game playing: This is the most widely spread example of how an intelligent program can learn over time by testing its ability to play against certain rules and
opponents. On this note, trying to determine the best move depends on a large number of factors which means that the possible outcomes or states are large. Using Reinforcement Learning the agent learns by playing the game without the necessity to manually insert hard coded rules. The example of the game “Backgammon” is the one usually used, which can “train” agents by playing against humans or even other agents. Moreover, recall “Deep Blue” from chess or “Watson” from Jeopardy! And many examples whose Reinforcement Learning agents skills got incredibly better over time.
3 PROBLEM DEFINITION

Before we get in more detail it should be emphasized that Reinforcement Learning is difficult to achieve due to the following reasons (Poole and Mackworth, 2010):

The agent does not know which action caused the reward or punishment. The action taken by the agent could have happened a long time ago, or a series of actions for this instance. This dilemma happens due to the delayed rewards that the agent receives. Agent isn’t also aware of the fact the it receives “rewards”, it is just briefly informed of the particular environment.

The effect of an action depends on an action. What for instance seems like a bad action now; it may result in a good idea later. Consider a graph where each of the lines is marked values and the agent has to find the shortest way to leave the graph. Assume that the agent makes the first move in the line with the highest value of all which makes it a bad action – if the reward policy determines it might have even been punished – but from this position the agent can move to the next line which has the shortest value of the first and second combined for the three options.

Exploration-Exploitation dilemma is evidently present in every environment. Assuming that the agent has gained knowledge of accomplishing a task, would it find other way to complete it? When there is too much exploration the agent never uses what it has learnt and when there is too much exploitation the agent chooses the same action and repeats them. However, in the next section we provide algorithms that tackle this problem by trying to maintain the balance between the two.

3.1 Intelligent Systems interacting with the Environment

Interaction is the first key mechanism that allows the intelligent systems to learn in Reinforcement Learning. However, there are different types of environments, agents, or
even multi-agent environments. Knowledge application in games has been central to the AI. Indeed, even the first application of Reinforcement Learning algorithms has been in checkers, the game of Arthur Samuel in 1959. Moreover, in 1992 Justin Boyan used the Tic-Tac-Toe example, in the same year Gerald Tesauro used the backgammon (see more in section 5) example which is still studied from Reinforcement learning scholars. Therefore, in this section we present some basic examples of agents interacting with the environment they have been provided and discuss occurrence examples that have been tested and proven successful in their tasks in real life.

3.1.1 Tic-Tac-Toe Example

One of the canonical problems the researchers have exploited for years if the board game Tic-Tac-Toe example. Therefore, it is essential to present the Tic-Tac-Toe example when discussing Reinforcement Learning (Sutton & Barto, 1997). In the 3x3 squared rectangles the opponents play either an “X” or an “O” until one of them wins. In order to win the game one of the players has to have three of its sign in the squares vertically, horizontally or diagonally. In the figure that we have presented below the “X” player has won the game.

In Reinforcement Learning we assume that the agent is playing against a human who is familiar with the rules of the game. On the other hand, the agent is a novice player who has yet to figure the rules. As in any other example, also here we assume that the agent can is only given a brief description of the environment i.e., it knows that the board is a 3x3 rectangular shaped.

![Figure 6: Tic-Tac-Toe Example of Agent-Environment Interaction](image)
3.1.2 Multi-Agent Environment: Rock Paper Scissors Example

A learning environment with more than one agent is thought when trying to increase the velocity of Reinforcement Learning algorithms, which as well represents one of the major problems in RL. This type of environment can be useful also to simulate the life-long learning environment that is the primary aim of RL. Below we present two agents playing the Rock-Paper-Scissors game and the outcome possibilities.

<table>
<thead>
<tr>
<th></th>
<th>Agent 2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rock</td>
<td>Paper</td>
</tr>
<tr>
<td>Rock</td>
<td>0/0</td>
<td>-1/1</td>
</tr>
<tr>
<td>Agent 1</td>
<td>Paper</td>
<td>-1/1</td>
</tr>
<tr>
<td>Scissors</td>
<td>-1/1</td>
<td>1/-1</td>
</tr>
</tbody>
</table>

**Table 1: The possible outcomes of two opponents in the Rock-Paper-Scissors**

The example discussed above is a zero-sum example that indicates that one of the agents has to necessarily either win or lose the game. In this case we have explored one of the extremes of the multi-agent framework which is full competitiveness, as opposed to the full cooperation, where the agents share the same function but can act autonomously. In other cases the outcome depends on the actions of all the agents (Poole and Mackworth, 2010). With respect to this example, the environment is usually more complicated and actions are more numerous in many other examples. The agents in any environment however should not consider other agents as part of the nature of the environment, since each of the agents has its own set of values and action-selection policies that automatically make them negotiate compete or cooperate with each other.

3.1.3 Turing Test

In the paper “Computing Machinery and Intelligence” published in 1951, Alan Turing proposed a test called “The Imitation Game”, which now has been dubbed as the Turing test, to test the intelligence of the intelligent machines, in comparison to that of a human,
which would settle and answer to the question: When is Artificial Intelligence no longer Artificial.

The goal of this test is to carry conversations with the intelligent machines and the test is passed only when one cannot distinguish whether the conversation was carried with a human being or an intelligent machine. We have illustrated the environment in the following figure.

![Turing Test environment scheme](image)

**Figure 7: Turing Test environment scheme**

The following conversation is an excerpt from Turing’s paper.

Q: Please write me a sonnet on the subject of the Forth Bridge.  
A: Count me out on this one. I never could write poetry.

Q: Add 34957 to 70764.  
A: (Pause about 30 seconds and then give as answer) 105621.

Q: Do you play chess?  
A: Yes.

Q: I have K at my K1, and no other pieces. You have only K at K6 and R at R1. It is your move. What do you play?  
A: (After a pause of 15 seconds) R–R8 mate.
If you look closely, there is a mistake in addition in the second question. The right answer is 105721, which leaves the doubt that the machine might be a human. In a second quotation he carries the following conversation concerned with sonnet writing.

Q: In the first line of your sonnet which reads “Shall I compare thee to a summer’s day,” would not “a spring day” do as well or better?
A: It wouldn’t scan.

Q: How about “a winter’s day.” That would scan all right.
A: Yes, but nobody wants to be compared to a winter’s day.

Q: Would you say Mr. Pickwick reminded you of Christmas?
A: In a way.

Q: Yet Christmas is a winter’s day, and I do not think Mr. Pickwick would mind the comparison.
A: I don’t think you’re serious. By a winter’s day one means a typical winter’s day, rather than a special one like Christmas.

### 3.1.4 Chinese Room Experiment

Nevertheless, not everybody agreed with Turing’s idea of testing artificial intelligence. John Searle, a philosopher, proposed the thought experiment known as the “Chinese Room” experiment. It is a reply to Turing’s proposed idea of assessment of intelligence. The core idea of this experiment is to imagine a person who cannot speak Chinese closed a room with Chinese manuscripts and try to answer questions that come from outside, in Chinese. With this example Searle questions the probability of an agent passing the Turing test, but still not showing the intelligence in its essence. Is it intelligence? Maybe yes. Therefore, regardless of machine’s understanding, it cannot prove its “mind”. Is there a higher perspective than intelligence? How is that related to consciousness? By raising these underlying questions, Chinese Room shows that the Turing test is insufficient to prove the existence of the actual understanding in the same way as the human’s brain would.
4 METHODOLOGY

4.1 Reinforcement Learning Algorithms

Typical RL methods based on the Markov Decision processes include two main kinds. Although it is difficult to compare the two types, in the first one, called model-based method, the RL agent first learns the model knowledge and then derives the optimal strategy from the model knowledge (i.e., SARSA algorithm). This type of algorithms requires larger data for learning. However, it makes use of the background data easily. On the other hand, there are the model-free methods which calculate the optimal strategy without model knowledge. This type of algorithms requires less data for learning and often use less computation time. Below we present Dynamic Programming which is used in the both types of algorithms. And then discuss more essentially the algorithms.

4.1.1 Dynamic Programming

Dynamic Programming (DP) is a powerful method of solving problems by breaking them into sub-problems and storing their solutions to help find the overall result of the problem. In comparison to the linear programming, dynamic programming is more flexible since it allows the decision to be taken one after the other, whereas in the linear programming all decisions should be taken upfront.

Dynamic Programming is often called “smart recursion”. Recursion in essence is a program that is part of its own definition, i.e., the acronym GNU which stands for “GNU’s Not Unix”. Another popular recursive example is calculating the n\textsuperscript{th} number in the Fibonacci numbers sequence. In both cases the function calls itself, thus it is recursive. This method finds application in Dynamic Programming, Divide-and-Conquer algorithms (quick-sort and merge-sort algorithms) and many others. Since Recursion is often slow, DP avoids re-computation by converting exponential running times to linear running times.
However, technically there is also recursion in the DP, but that is without repetition. While dealing with complex problems it is in DP’s core to define the sub-problems first. After breaking down the problems it is important to find the recurrences that associate to the sub-problems, this is when the recursion comes in. And at last, it is required to solve the base cases. Dynamic Programming has not much to do with writing code (Dasgupta, Papadimitriou and Vazirani, 2008). The term was coined in 1950s by Richard Bellman by describe a multiple stage process, since “programming” in the mid 20th century simply meant planning.

In terms of Reinforcement Learning, if we used Linear Programming our agent would have to take the decisions forthright and plan its whole path in advance before even starting to explore the environment, which is not feasible with the amount of learning outcomes. Instead, our agent receives feedback after each action due to the usage of Dynamic Programming. It is important to note that the model of the environment decides when the feedback or in RL language, rewards, will be given to the agent. Recall the maze example on whether the environment gives rewards after each move or at the end after the agent finds the exit.

Dynamic Programming methods used to solve RL problem are well developed mathematically but require a complete and accurate model of the environment (Sutton & Barto, 1997). The main drawback of DP is that it requires sweeps of the state set, and if the state set is numerous the whole iteration will be expensive and time consuming. Asynchronous DP methods have been presented to indicate the in-place iteration of DP algorithms. Asynchronous DP allows more flexibility in sweeps since it doesn’t have to go through all the values, in fact it can iterate over particular values several times. However, when it backups particular values it has to continue to back all the values of other states, to converge correctly; also these algorithms allow flexibility in selecting states that have applied the backup policy. Asynchronous DP doesn’t necessarily guarantee less computation by providing less sweeps, but rather than holding the algorithm wait for until any long sweeps have finished, it allows it to make progress by improving a policy.
Below we present other methods that are typically used to solve RL problem which however have the same objective has DP but try to make this attempt more simplified by using less computation and not requiring a whole model of the environment.

### 4.1.2 Monte-Carlo Model

Monte Carlo is most useful when a model is not available. The advantage Monte Carlo has over Dynamic Programming is that it does not need bootstrapping because updating states does not depend on successive states. Monte Carlo also does not need to have full models of the environment or learn all the possible states of an action. These methods are based on learning directly from the experience by using two types of learning: on-line learning where no model is necessary and still it attains optimality and simulated learning which doesn’t require a full environment model.

Monte Carlo methods solve the RL problem based on averaging returns by estimating episodic tasks rather than continuing tasks and we assume that the experience is divided into episodes and that all episodes terminated no matter what actions are selected. However, in Monte Carlo it is crucial to maintain enough exploration, because of its $\epsilon$ soft action selection policies. In essence, Monte Carlo methods are very similar to the Dynamic Programming methods because they carry the most fundamental ideas of Dynamic Programming which is computing the same value functions and interacting to find optimality in essentially the same way.

### 4.1.3 Temporal-Difference Learning

Temporal-Difference Learning (TD-Learning) methods principally can be used to measure the value functions. The value functions need to be estimated in order to revise any state-action pairs to have the rewards simultaneously and not wait until the final reward is received (to update the state-action). This is a characteristic of TD-Learning that comes
from Dynamic programming (DP); whereas the ability to of TD-Learning methods to learn directly from raw experience without a model from the environment is a trait that derives from Monte-Carlo methods. In regards to the DP model similarities, in many literatures, TD-Learning method is called “bootstrapping” because rather than updating the values using the final reward; it partly renews them using an existing estimate. The two qualities of TD-Learning make it a combination between DP and Monte-Carlo methods, and the combined idea stands as the overall core idea of Reinforcement Learning.

In essence, TD-Learning is a prediction-based method that considers the ensuing and their correlation. As suggested Richard Sutton, TD-Learning’s core idea is to adjust predictions to match other predictions for the future. In Reinforcement Learning, TD-Learning is used to predict the measure of the total amount of reward the agent has possibility in gathering.

### 4.1.4 Q-Learning Model

Q-Learning is categorized as a model-free and off-policy RL control algorithm part of the TD-Learning method. This framework was first introduced by Watkins in 1989 (Watkins & Dayan, 1992) and it is used to find the suitable policy to select actions in a finite Markov Decision Process (MDP). Q-learning can handle problems with rewards without requiring any adaptations due to its model free nature of execution. Q-learning as well maximizes the action-state pair Q (s,a) by choosing the action a, at each step s of the function. Q denotes the estimated function which explains how good an action is in a certain state.

According to Watkins and Dayan, Q-Learning despite being a method of TD-learning, it can also be viewed as a method of asynchronous Dynamic Programming due to its “providing of agents to act optimally in Markovian domains by experiencing the consequences of actions, without requiring them to build maps on domains”. In Q-Learning, similar to TD-Learning, an agent tries actions in particular states and evaluates them according to the immediate rewards and its value of the states. Thus, by trying all the possible options at all states and fining the best values the agent can find the best actions.
that bring the best rewards in the long-term. Q-Learning method is especially used when applying offline learning and using that learning – or policy – in an agent that does not explore.

4.2 Action Selection Methods

The agent is provided with the environment and tasks which inevitably enforces the action selection policies in order to take the next action. The goal of action selection policies is to maintain the balance between exploration and exploitation by not always selecting to exploit. This dilemma is dubbed as the exploration-exploitation dilemma, which further notes that it is necessary to maintain the balance between the two because if there is too much exploitation the agent won’t be able to find out about the other potential rewards in the environment, whereas too much exploration will keep the agent away from the short-term rewards. The following Action-Selection Methods have their pros and cons towards selecting the most convenient actions.

4.2.1 $\epsilon$-Greedy

The action with the highest reward is chosen most of the time. This selection is called greedy selection. The agent in such cases chooses the action that it believes is the best in the long term with a probability of 1, given that the probability range is from $0 < \epsilon < 1$ (Tokic and Palm, 2011). However, sometimes the actions are chosen randomly, independent of the value estimation of the actions. This policy ensures that if there are enough trials done, the agent will explore each action infinite times, which in the end brings optimal rewards. The greedy agent is stubborn and doesn’t change the lower route it usually takes thus it does not learn the true utilities or the true optimal policy, which is the upper route. As suggested by Sutton, this is the method that is chosen most of the time. One of the advantages of this method is that is requires no data memorization is required such as counters or confidence bounds (Tokic and Palm, 2011), which makes the methods particularly interesting for very large state-spaces. One disadvantage though, is that in practice it is not clear which setting of $\epsilon$ is used to give good results. However, the method
Value-Difference Based Exploration (VDBE) aims to overcome this by adapting a state-dependent exploration-probability, $\pi(s)$, which instead of requiring tuning a global parameter by hand it is based on the fluctuations of temporal difference error.

In some literatures, the $\pi$-Soft selection method is often proposed. In this method, the action with the highest reward close to 1 is chosen as well, and the rest of the actions are chosen uniformly, which is what makes it very similar to the $\pi$ Greedy.

### 4.2.2 Softmax

Another drawback of the $\pi$ Greedy method is that it chooses actions randomly, which means that it can select the worse possible action just as possibly as the next to best one. Softmax tries to fix this remedy by assigning a weight to each of the actions according to their action-value estimate. In this regard actions are chosen based on their action-value weight, which makes it less likely for the worst actions to be chosen. When the worst case is very unfavorable, this method is usually used.

It is not strictly defined which of these methods works best. It depends on the nature of task and if it is a task that requires interaction with humans, it also depends on the human factor.

### 4.3 Popular examples of Intelligent Systems learning abilities

A lot of scientific organizations, research institutes and universities have made attempts towards designing intelligent programs which would improve their performance due to the learning environment and conditions they were provided. Such examples are briefly discussed below starting with IBM’s Deep Blue and Watson as well as the most ambitious way of robot making in the form of the robot humanoid.

#### 4.3.1 Deep-Blue
The chess-playing computer was developed by IBM with the purpose of beating the world champion. The measuring of computer intelligence in chess games is an old idea. It dates back in 1770 when the Hungarian scientist Wolfgang Von Kempelen designed “The Turk” a chess playing machine that worked only when a human sat inside the machine and decided its next moves.

Later in 1950 Claude Shannon wrote a paper on “Programming a Computer for Playing Chess” in which he proposed algorithms similar to Tree Search and Evaluation Functions, which would predict the efficacy of the next move and used to minimize the maximum possible loss.

Upon below mentioned principles was build the Deep Blue as well. In 1996 Kasparov won the chess match, but a year later, in 1997 in New York, under tournament rules Deep Blue beat chess grandmaster Garry Kasparov to 3.5 to 2.5. Deep Blue applied the brute force strategy and its processing was parallel. It could evaluate 200 million positions per second. Ever since Deep Blue was introduced the technology has flourished and now we have even more advanced chess playing programs in simple desktop or laptop computers.

### 4.3.2 The Humanoid Robots

Humanoid Robots are a wave of robot designation that resemble humans physically as well as with their human level intelligence and behavior. They are usually designed as general purpose intelligent programs for research and experimental purposes and potentially to assist humans in daily tasks.

An example of a humanoid robot is Honda’s robot called ASIMO, short for Advanced Step in Innovative Mobility. ASIMO was designed to perform in real world environments with the capability of walking or running up to 6 kilometers per hour. Created in 2000, ASIMO has the capability of recognizing objects and faces and also making distinctions between different voices, making it one of the most advanced humanoid robots in the world.
Another example of humanoid robots was designed by the Massachusetts Institute of Technology (MIT) with the purpose of studying theories of artificial intelligence and cognitive science. The way this robot, dubbed as COG, was supposed to learn was by being presented in the same environment with humans, the way babies learn to behave as humans when presented in social groups. However, the learning rate of this robot was in turn over long periods, which is one of the main problems the overall field of Reinforcement Learning faces today.

4.3.3 Watson of Jeopardy!

Watson is another example of a computer beating humans at intelligent games. Developed by IBM’s DeepQA project with the research team led by David Ferrucci, Watson is a question answering computer developed specifically to answer the questions of the Jeopardy! quiz show. In 2011 Watson, named so in regards to IBM’s first CEO Thomas J. Watson, beat two former winner of this game.

Watson while playing had access to 200 million pages, including the full text of Wikipedia, with a content of four terabytes in disk storage but during the game it was not connected to the internet. It continuously outperformed its human opponents but had trouble in questions with only few keywords.

A QA (Question Answer) searching computer, Watson applies natural language processing, knowledge engineering, automated reasoning and machine learning. The key difference between QA searching technology and the traditional document searching (search engine provided search) is that while document search takes keywords and returns web pages based on page rank, QA technology tries to understand the question in detail while using natural language and give a more precise answer.
5 RESULTS

As we discussed in the beginning of the thesis, Intelligent Systems employ fuzzy logic, neural networks, evolutionary computation or intelligent agents; as such find applicability in manufacturing, intelligent control or business application for data mining in large scale corporations. The common accepted definition of intelligent systems that can be applied is described as a system of methods and infrastructure that enhance human intelligence by learning and discovering new patterns, relationships, and structures in complex dynamic environments for solving practical problems (Kordon, 2012).

A definition of what is considered to be “industrial”, in terms of the adequate dataset that we were trying to find to perform the application of RL in order to extract data patterns could not be defined but we tried to set common ground such as considering the format of the dataset, the number of samples it has and the states it can support.

5.1 Reinforcement Learning application domain

One of the well known cases of performing data mining – extracting patterns from data – is by the usage of the agent, in which case the agent is responsible for the data collection. Agent-based computing, such as the one used in Reinforcement learning has been dubbed as ‘the next significant breakthrough in software development’ (Sargent, 1992), and ‘the new revolution in software’ (Ovum, 1994).

The application domain of RL is still blurry. The RL algorithms keep evolving over time and new ones continue to be proposed. RL has been known to researchers since the 1950s and a lot of experimental research has taken place – enough to prove that RL is an incredibly powerful learning technique that can potentially be applied to industrial applications. These half century studies have also shown that the application domain can be broader, which is what will present in the next section, by mentioning some of the most successful applications of RL in different spheres.
1. **Game playing.** The first field where most of the AI algorithms have been tested has been in gaming since it involves the combinatorial search problems. The same tribute holds true for RL as well. In the 1950s, Arthur Samuel, probably even before the name RL was coined, observed that computers were no exception to the laws of nature and that could learn by experience. He is famously known for the usage of RL in the game of Checkers. The training scheme he used is very similar to the one used in Temporal Difference and Q-Learning while the value function was represented by a linear function approximation.

As Michael Littman points out, “two-player games do not fit into the established reinforcement-learning framework since the optimality criterion for games is not one of maximizing reward in the face of a fixed environment, but one of maximizing reward against an optimal adversary (minimax)”. However, RL algorithms can be adapted for a much broader class of games as it has been done so by A. Samuel and other researchers.

Another successful application by Tesauro of the RL has been in the game of Backgammon which employs the TD-Learning technique. Backgammon has nearly $10^{20}$ states which make RL impossible to be applied due to its scalability problems with large data; however Tesauro used a three-layer neural network based on back-propagation function approximation as a value function. Tesauro applied a no exploration strategy, which means it employed a greedy action selection policy by choosing the one action with the highest reward. After a couple of months of constant self-play, the learning algorithms proved to be efficient, and was considered one of the best players in the world; for instance, in 1,500,000 training games it had 80 Hidden Units and it lost by 1 point in 40 games.

An honorable mention in this application domain should deservingly go to the game of Solitaire (Solitaire: Man versus Machine, 2005), Chess (The KnightCap program)
which alters its heuristic evaluation function using TD-Lambda, and Checkers (TD-Learning applied to high-performance game-playing).

2. **Robot Control.** The one fascinating manifestation of AI that grabs social attention and it continuously keep exploring new application possibilities is the robotics. Being that the aim of the AI researchers is to make machines similar to humans, the RL algorithms are widely used, and some of these applications are presented below.

Fieldman and Stone [26] demonstrate the ability of a robot to learn a high-lever goal-oriented task which is fully implemented in a Sony Aibo ERS-7 robot. The robot learns ball acquisition in three hours machine time and the behavior described is said to be significantly better than that of using a hand-coded behavior. To continue with another example of object acquisition, Schaal and Atkeson [27] constructed a two armed robot that learns to juggle a device. The human precision cannot be compared with that of the robot, since after 40 first attempts the robot learns to keep juggling for hundreds of hits, in contrast to a human which would take more practice to achieve merely tens of hits. The robot uses dynamic programming to learn the world model from experience.

Another example in the field of robotics and control is the convoluted example of box-pushing for extended periods of time; which are regarded by Mahadevan and Connell at IBM T.J Watson Research Center [32]. In their paper they describe the usage of RL towards learning individual modules in a behavior based robot, concretizing with Finding a box, Pushing a box and Getting unwaged. Also they conclude that by using a subsumption architecture is superior to using any one-part controller architecture in RL. The agent used in this experiment learned the box-pushing task more efficiently by subsumption approach rather than having it decomposed. The box-pushing robot, who learned to perform comparatively with the performance of a hand-coded computer, was also used to conclude that the subsumption approach was more effective than the monolithic one by a factor of two.
3. **Elevator Scheduling.** As part of a simulation, Q-Learning has been applied to improving the efficacy of four elevators servicing four floors [28]. In which case the number of states was $10^{22}$ which makes RL Markov Processes difficult to implement, thus a function approximation using neural networks was used. The averaged squared wait time for passenger of the designed controller was less than half in comparison to the controller used in real elevator systems. Thus the results of Crites and Barto in the difficult real world problem of elevator dispatching, demonstrate the power of RL on large scale stochastic dynamic optimization problem of practical utility.

4. **Decision making in unpredicted rapid environments.** RL is being optimized for the usage in decision making theory, by being able to predict the rapidly changing environment models, and to achieve results in predicting the most suitable choices [29].

   **Stock investment decisions.** There have been a couple of trials in applying RL to predicting the stock price movements, which is considered as Markov process which can be optimized by an RL based algorithm. Among the most adopted techniques is the TD-Learning and the function approximation is achieved by the use of neural networks; in which case the learned values of states represent the stock price at a given time.

   **Chemotherapy treatment decisions.** Zhao, Kosorok and Zeng [30] describe how to use RL to make individualized decisions and personalized treatments on treating life threatening diseases such as cancer. By using Q-Learning algorithm it is shown that RL can have tremendous potential in clinical research because it selects actions that improve outcomes by taking into account delayed effects even when the relationship between actions and outcomes is not fully known. Even though a simulation so far, the proposed strategy for describing diseases treatment is highly productive in selecting actions that yield results in treatment.

5. **Helicopter Maneuvering.** In the expanding field of Autonomous Control, the example derives from the Helicopter Control which is described to be a difficult control problem. In 2004 a research of Stanford University was described as successful in using the RL...
to designing a controller for sustained inverted flight on an autonomous helicopter, specifically, with the means of MDP set of RL tools.

However there are two reasons that make action-reward RL difficult to implement. First of all, actions have non-deterministic (stochastic) effects, which mean that they occur by chance and are initially unknown and must be learned. And secondly, rewards or punishments can be infrequent, which is then difficult to determine what actions are responsible for a particular reward or punishment. Furthermore, the rewards are usually assigned usually at the end of a sequence of action which increases their infrequency and does not allow to easily making a pattern between actions and rewards.

Furthermore, even though RL is sometimes difficult to implement, quite often it is not the right solution due to the scalability problems which have prevented its successful use in many complex real-world domains. The more complex the tasks, the longer it takes a reinforcement learning algorithm to converge to a good solution. At this era, given the breakthroughs of data and consequently information, RL questions the importance of big data since the main issue remains making RL techniques efficient in working with small data as well in both cases the function approximation should be utilized.

It is often argued that Reinforcement Learning receives more academic research, rather than industrial research and application. According to the respective researchers the practical RL applications that we will get to use in our daily life not just in simulations or not even only in industry, are still to be seen since a great deal work is in progress and with that various RL algorithms that will shape the future are being designed. As we have seen using RL to control physical problems that might take extended periods of time to arrive at a desired outcome – such as walking or flying – stands at the core of the RL research. Solving logistical problems, such as box-pushing robot we discussed or the elevator dispatch control, scheduling and power optimization.
5.1.1 Deep Reinforcement Learning

Among the other techniques for exploiting data in ML is the new big trend so called Deep Learning. Deep Learning refers to the artificial neural networks that work in many layers. This is the reason it is dubbed as “deep” since the input surpasses several non-linearities before producing an output. Many researchers agree that Deep Learning is another name for Artificial Neural Networks (ANN) which has existed for 40 years. Nevertheless, the ANN of the 40 years ago has contained two or three layers, whereas now they contain 10 to 100 or more layers. Thus due to the powerful capabilities of exploiting data, Deep Learning needs to mainly be used for big data exploitation.

The reason we briefly introduce Deep Learning is due to the powerful method it provides when it uses Reinforcement Learning altogether known as Deep Reinforcement Learning (DRL). The first renowned use of this technique was at a small company in London called DeepMind (later bought by Google), which used DRL to demonstrate the effective play of a computer in the video game Atari 2600. The learning process was rather simple and was made through sensory input (observing the pixels on the screen) as the computer receives a reward when the game score increased, which it was ordered for – maximize the score on the screen [31]. After the first 120 minutes of training it plays like an expert and after some time it even realizes the trick to winning the game and collecting the maximized score in a short amount of time (which is by digging a tunnel though the wall of bricks). This experiment used the Q-Learning model-free RL algorithm.

5.1.2 Knowledge-Based Reinforcement Learning

RL has taken different forms of applications which are suited to the uniqueness of the situation. We mentioned that studies show that RL suffers from scalability problems, thus it cannot perform well in real world complex examples. However, Kudenko and Grzes (2009) have proposed two new techniques in the field of the so called Knowledge-Based RL (KBRL). These techniques make advantage of the fact that there is human-expert knowledge in many areas which if applied alone to automated expert systems provide sub-
optimal solutions due to the uncertainty of the environment and incompleteness of the knowledge. On the other hand this domain knowledge uses RL it can overcome the weakness of the heuristic domain knowledge and produce optimal solutions [23]. Therefore, in many RL applications the agent is considered tabula rasa, which indicates that the agent has no prior knowledge but when combined with the human knowledge that already exists, it produces faster and more efficient results.

The first technique is called STRIP based KBRL with incorporates the STRIP operator knowledge in reward shaping, which in other words is plan-based search, to focus the search for the optimal policy. On the other hand when the STRIP operator is not available, it is proposed a second technique which supports the reward with hierarchical tile coding. This method suggests that along with the Q-function which is represented with low-level tile coding, a V-function with coarser tile-coding can be applied in parallel, which as result can be used to approximate the potential for ground states. The applications of the KBRL methods include Network Security [33] – where groups of Multi-Agent RL fight groups of distributed denial-of-service (DDoS) – and Distributed Mobile Sensor Management [34] which by using the points-of-interest (POI) represents a stochastic simulation in sensor domain and it also involves the impact and dynamics of the sensor control by the use of RL.

5.2 Case study: Quadrotors Autonomous Flight Control using RL

The one industrial area of application of the RL, which we will discuss more thoroughly, is employed in the field of robotics and control. The field of the Unmanned Aerial Vehicles (UAV), which is a common name for the aircrafts piloted remotely from a board (without the presence of a human pilot onboard), is one of the most adequate examples for the RL application. One of the most used UAVs for research are the Quadrotors (also called quadrotor helicopter or quadcopter) which use four motors with four fixed pitch blades to life the aircraft. Two of the motors rotate counter clockwise and the other two rotate
clockwise. Each of the four motors (rotors) is adjusted to the speed required to generate a total thrust or a turning force that will enable the uniform movement of the quadrotor [35].

### 5.2.1 Architecture and Organization of quadrotors

In addition, because the quadrotor is a nonlinear coupling dynamics system, it is difficult to be controlled. In the past there used to be four pilots controlling the each of the motors simultaneously. However, due to advancements in the micro controller technology, the nowadays quadrotors can be controlled even by the novice pilots. The argument is that quadrotor helicopters are a much more stable platform for artificial intelligence research than traditional helicopters and planes due to their vertical takeoff and landing (VTOL) capability and the simple construction and design. Furthermore, they are expandable and more inexpensive in comparison to the helicopters. In extension to the VTOL a quadrotor has the Inertial Measurement Unit (IMU), Electronic Speed Controllers (ESCs) and a microcontroller. The usage of this technology overlaps two of the main goals of the RL algorithms: the control theory and decision making in rapid changing environments. The figure below describes the system architecture of the quadrotor and the communication between its components [41].

![Figure 8: System Architecture of the components of the quadrotor](image)

The objective, in the case of the quadrotors, is to reduce (or replace) the extensive time it takes to manually tuning them to learn new maneuvers, with a continuous state-action space RL algorithm. Quadrotors, in comparison to the helicopters, are much cheaper and they
come in variety of sizes. Moreover, due to being an easy-to-maintain device, quadrotors can even be controlled by the amateurs and hobbyists.

The pilot that controls the quadrotor is placed outside the vehicle and this adds challenges in regards to the situational awareness. Although many quadrotors are equipped with on board cameras to give the pilot first person perspective while flying the quadrotors and to make it easier to make decisions about the pre-planned routes and related issues; many precautions have to still be taken into account such as the obstacle maps in order to be within the line of sight to ensure a safe flight [35].

For the reason that no direct human control is required, quadrotors are used for more complicated and sometimes dangerous tasks. Therefore, the Machine Learning approach is most appropriate in order to control the movement of the rotors and make sure that the arm turn is most suitable in regards to the situation.

5.2.2 Usage of quadrotors

It is necessary to point out that quadrotors are used in research to test out ideas especially on control theory, robotics and navigation. By using the mentioned field, often they are seen to fly in groups and make artistic movements in the air by applying the principles of navigation. Moreover, the use of the quadrotors is vast in many fields due to their easy to maintain mechanical structure and simple design. Furthermore, quadrotors are a growing area of interest for research due to their use in expandable technologies such as Global Positioning System (GPS), Gyros, Magnetometers, Barometers, Vision and Accelerometers etc. To name a few of these fields of use:

- Search and Rescue
- Logistics and Freight
- Transportation
- Surveillance
• Crowd Control
• Aerial Imagery/Film
• Wild Fire Monitoring
• Military and Defense

The main research regarding the quadrotors is based on how to design autonomously controlled quadrotors. Thus, Reinforcement Learning is one of the techniques that enable autonomous control. As we have seen the standard RL environment is defined as a MDP. Thus, when considering the quadrotor as an agent, the actions defined for each of the rotors are: pitch, roll, yaw and thrust, as shown in the figure below.

**Figure 9: Identified actions of the quadrotor and their possible behaviors**

5.2.3 Uses of Reinforcement Learning in Quadrotor control

As it can be seen from the figure, controlled by the throttle stick, yaw action is the rotation of the head of the quadrotor either to left or right. Pitch action is the movement the quadrotor makes which is backward or forward. This movement is achieved by moving the
aileron stick backward or forward, respectively. Roll action is also controlled with the aileron stick whose movement to the left will make the quadrotor fly to the left, and the movement to the right will make the quadrotor fly to the right. Thus, it can be concluded that Roll makes the quadrotor fly sideways. Whereas Thrust is the amount of power it takes for the quadrotor controller to achieve one of the three actions, i.e., when applying equal thrust to the four rotors, the quadrotor adjusts its altitude or when applying more thrust to the rotors rotating in the same direction it adjusts its yaw; whereas to adjust the pitch or roll, the quadrotor applies more thrust to one of the rotors and less thrust to the other rotors that are placed diametrically opposite. Understanding the mechanics of the architecture of the quadrotors, it makes much easier to define the states and actions of the agent which need to be used in RL algorithms. The figure below gives an understanding on the dimensions on which the actions take place.

![Diagram showing the dimensions of quadrotor's actions](image)

**Figure 10: The dimensions of quadrotor's actions**

On the other hand, the states of the quadrotor depend on the type of problem that we seek to solve, however the position of the quadrotor in the environment is detected via the sensors such as the IMU which is responsible to maneuver the aircraft. IMU uses the accelerometers, magnetometers and gyroscopes to measure and report on the orientation and velocity of the quadrotors. The collected data is then loaded into its computer software. Since the IMU is prone to accumulated errors it is often replaced by the barometers, which instead of using the dead reckoning data transmission method, it processes the collected
data in real-time. Among the other sensors they also use the ultrasonic sensors, infrared rays or laser range finders, at certain heights for obstacle detection and avoidance. Accelerometer measures the position of the quadrotor with respect to Earth; however since it assumes that the quadrotor is stationary it cannot detect rotation, whereas the Gyroscope estimates the rate of rotation around an axis.

5.2.3.1 Case 1: RL in stochastic MDP model of Quadrotor control design

Having given a short description of the architecture including the RL components, we are allowed to speak about the RL algorithms mainly used in research. The paper on ‘Multi-Agent Quadrotor Test-bed Control Design’ (Waslander, et al., 2005) employs the model of the aircraft dynamics as a stochastic Markov Process, in which case the components in the equation are the battery level, total motor power, altitude and the distribution of the output error as determined by maximum estimate of the Gaussian noise in the Locally Weighted Linear Regression (LWLR) which is used to map the current state of the agent into update on the subsequent state estimates. In which case, the algorithm used, is a model-based one which defines the policies of the overall RL case. The paper illustrates the results of the integrated sliding mode control and the RL algorithms.

In this section we will discuss the setting of the RL continuous state-action space control algorithms. First of all, a nonlinear, nonparametric model of the system is first constructed using flight data, approximating the system as a stochastic MDP. Secondly, based on the gained model of the environment, a model-based RL algorithm performs policy-iteration search, to find an optimal control policy. In mapping the state of the environment, as introduced above, a Locally Weighted Linear Regression (LWLR) is used; which is suitable in this case due to its fitting to a non-parametric curve to the local structure of the data.

The control policy has been chosen after having taken into account the model of the stochastic MDP – which considers the entry values and gives output accordingly – and a
quadratic reward function – which gives reward for accurate tracking and good damping keeping a reference state desired for the system. The need for policy iteration is given beforehand with an RL algorithm in the figure below. In the testing phase it is necessary to keep the convergences by giving the same set of values for the initial parameters and reference trajectories at each iteration during the simulation.

Algorithm 1 Model-Based Reinforcement Learning
1: Generate set $S_0$ of random initial states
2: Generate set $T$ of random reference trajectories
3: Initialize $w$ to reasonable values
4: $R_{best} \leftarrow -\infty$, $w_{best} \leftarrow w$
5: repeat
6: $R_{total} \leftarrow 0$
7: for $s_0 \in S_0$, $t \in T$ do
8: $S(0) \leftarrow s_0$
9: for $t = 0$ to $t_{max} - 1$ do
10: $u(t) \leftarrow \pi(S(t), w)$
11: $S(t + 1) \leftarrow LWLR(S(t), u(t)) + v$
12: $R_{total} \leftarrow R_{total} + R(S(t + 1))$
13: end for
14: end for
15: if $R_{total} > R_{best}$ then
16: $R_{best} \leftarrow R_{total}$, $w_{best} \leftarrow w$
17: end if
18: Add Gaussian random vector to $w_{best}$, store as $w$
19: until $w_{best}$ converges

Figure 11: RL algorithm used for policy iteration

The result of this simulation is noteworthy; using RL in an outdoor flight test, the policy iteration algorithm arrived at the implemented control law after only 3 hours on Pentium IV computer (2005). The data has also been filtered using a Kalman filter [38].

5.2.3.2 Case 2: AR.Drone control via Robot Operating System based on a model-based RL algorithm

A research at Brown University on Autonomous Quadrotor Control using RL uses the Robot Operating System (ROS) in a type of UAV called AR.Drone. The ROS is an open-source framework for finding, building, and seamlessly running robot control code across
multiple computers. In essence ROS provides a structured communications layer above the host operating systems of a heterogeneous compute cluster [44]. Whereas the Parrot AR.Drone is a remotely controlled quadrotor that can be controlled by Smartphone or Tablet in Android or iOS Operating Systems [45]. Its Version 2.0 has the following characteristics:

- Interfaces: USB and Wi-Fi 802.11n
- Front camera: 720p sensor with 93° lens, recording up to 30fps
- Vertical camera: QVGA sensor with 64° lens, recording up to 60fps

**Figure 12: Parrot AR.Drone system and control**

Interfacing directly with AR.Drone over 802.11 Wi-Fi, a modified version of Brown University’s AR.Drone driver is responsible for converting ROS messages to and from Parrot’s custom packet format. Through this node the quadrotor controls its angular and linear velocities. In addition, it also converts the readings of the sensors of the quadrotor into understandable messages to be understood from the other nodes of ROS. Whereas to measure the location of the quadrotor is a global positioning scale the Vicon Tracking System was selected due to its 125 Hz sampling rate. To obtain the exact position of the drone via the Vicon measurements Tracking System, a Kalman filter is proposed which via
various measurements observed over time estimates the joint probability distribution over the variables for each timeframe [37].

Using the information on the MDPs and the RL setting, as the drone’s state and its motor outputs as actions, the problem of training the quadrotor to perform a high-level action is in the same form as the MDPs while using a model-based RL algorithm, is what is to be done as future work.

5.2.3.3 Case 3: Quadrotor’s simulated control using a MATLAB Quadcopter Control Toolbox

The research on the flight control of the quadrotors using RL, has led to designing a MATLAB Quadcopter Control Toolbox [36] based on the Linear Quadratic Regulator (LQR) which allows parameter change such as the arm length or mass weight. As it can be seen from the figure below, this toolbox allows rapid visualization of the system response in 3D rotation. In designing the toolbox the LQR tracking controller has been taken into account as well as the trajectory controller. The latter allows a number of waypoints adding to the 3D trajectory which enables the quadrotor to follow complex maneuvers.
Figure 13: The GUI of the MATLAB quadcopter control toolbox

Except for presenting the toolbox, this paper [36] also investigates control using RL. The Quadrotor is formulated as a MDP in the 5-tuple \( (S, A, \{P_{sa}\}, R, \gamma) \). The \( S \) represents the set of the system states, which is a vector with the position and the velocity of each state in the \( r_z \) direction (We took a look at the architecture of the quadrotor in the sections above). The second parameter, \( A \), is the set of actions the system can take from any state, where each of the actions is an input thrust. The \( \{P_{sa}\} \) describes the unknown state transition probabilities, which captures the changes of the system. It represents the variations in the state 2 given state 1 applied action 1. The fourth parameter represents the reward function and the fifth parameter is the discount factor on rewards, set to be 0.995.

The reward function in regards to the reference state, is designed to encourage convergence of states to its vicinity, where a discretized pseudo-Gaussian is placed on the reference state, which gives positive rewards to desirable states and negative rewards to undesirable states. In order to learn the dynamical model, it is important to sample the action space. If we assume that the system starts from a random state \( s \), and an action is applied, a state \( s_1 \) is observed. Actions are randomly sampled from the parameter \( A \), to encourage the exploration of the state space. As expected, when a set of actions achieve an optimal policy, a bias towards the discovered optimal actions is applied. The result of this application can be seen in the figure below.
The black-box nature of MDPs allows it to be used and give results in non-linear environments such as this one [36].

5.2.4 Case study on Policy Gradient via Signed Derivative RL algorithm in quadrotor control

Using MATLAB, RL algorithms have been applied differently in simulated test cases such as using the algorithm Policy Gradient via Signed Derivative (PGSD) in movement stabilization [39]. For difference from the presented examples above this case study uses a model-free algorithm. The model-free algorithm was implemented in the online controller tuning procedure and furthermore requiring no details on the dynamical model of the vehicle. After having specified the parameters of the controllers, the RL algorithms are used to auto-tune the parameters derived from the four channels – yaw, roll, pitch – which improve the performance during flight. A more detailed overview of this experiment is given below.
As a first step, it is necessary to identify the channels through which the quadrotor is controlled; and they can be roughly summarized as yaw, roll and pitch, which were described in the sections above. To conclude the parameterization of controller by having taken into account the linear controller with coupling terms as well as the nonlinear controller, it has been generated a PID control scheme of nine control parameters in total.

As a second step, it is required to tune these parameters, which in practice would require the generation of steps in excruciating details. Instead, this is when the RL is used and after a couple of iteration the parameters will be autonomously tuned. As we gave a brief introduction above, the algorithm employed is the PGSD. The paper that proposes the PGSD, considers the learning policy parameters that improve performance of a system. On the other hand, the approximation of the policy gradient method, called Signed Derivative, is based on the intuition that it is often very easy to guess the direction in which control inputs affect future state variables, even if the model of the environment is not provided. In the cases presented in the paper, this algorithm performs as well as the model-based policy gradient [39].

---

**Algorithm 1 Policy Gradient w/ Signed Derivative (PGSD)**

**Input:**
- $S \in \mathbb{R}^{m \times n}$: signed derivative matrix
- $H \in \mathbb{Z}_+$: horizon
- $Q_t \in \mathbb{R}^{n \times n}, R_t \in \mathbb{R}^{m \times m}$: diagonal cost function matrices
- $\alpha \in \mathbb{R}_+$: learning rate
- $\phi : \mathbb{R}^n \times \mathbb{R} \rightarrow \mathbb{R}^k$: feature vector function
- $W \in \mathbb{R}^{k \times m}$: initial policy parameters

**Repeat:**
1. Execute policy for $H$ steps to obtain $u_0, s_1, \ldots, u_{H-1}, u_H$.
2. Compute approximate gradients w.r.t. controls:
   \[ \nabla_{u_t} J(s_0, \Theta) \leftarrow \sum_{t'=t+1}^{H} S^{T} Q_{t'}(s_{t'} - s_{t'}) + R_t u_t \]
3. Update parameters:
   \[ w \leftarrow w - \frac{\alpha}{H} \sum_{t=0}^{H-1} \phi(s_t, t)(\nabla_{u_t} J(s_0, \Theta))^T \]
As a third step, the credulity of this algorithm and software experiment has been tested on hardware also. The used machinery is called NI myRIO-1900, which provides analog input (AI), analog output (AO), digital input and output (DIO), audio, and power output in a compact embedded device. The NI myRIO-1900 connects to a host computer over USB and wireless 802.11b, g,n. [42]. In this experiment, due to the large number of features that NI myRIO-1900 provides, it has been used as a flight control processor, sensor data acquisition, algorithm computation, data fusion and rotor control.
During the testing phase, the control loop frequency employed is 100 Hz and the necessary amount of iterations, for each iteration being 10 seconds of aviation time; it has been estimated to be 10 such iterations in order to receive the expected results.

The results can be found in the figures below (the figure in the left describes the initial performance, whereas the figure in the right describes the ultimate performance) [40].

![Figure 17: The result of a model-free RL algorithm in controlling the attitude of the quadrotor](image)

### 5.2.5 Conclusions on the foreseen future work of RL in Quadrotor control

This field of research has been launched not long ago. The examples presented still require continuous research to yield results. The industry of UAVs, in particular, the area of quadrotors In the case of RL, in many of the tested applications so far, algorithms have been applied to a black-box quadcopter, by using the MDP approach, thus allowing the learned control to be performed [36]. However in the case study with ROS and AR.Drone, the upcoming work is expected to focus on using a continuous state-action reinforcement learning algorithm to learn complex behavior more quickly than would be possible through manual programming [37].
6 DISCUSSION AND CONCLUSION

6.1 Summary

In this thesis we have covered a spectrum of information regarding the use of a Machine Learning technique, expectedly well known by now, Reinforcement Learning. After a discussion on Intelligent System’s architecture and characteristics; we took a general look on the overall field of Artificial Intelligence and introduced the field of Machine Learning most notably its three types of learning; supervised learning, unsupervised learning and reinforcement learning. After discussing their differences and underlining their uses in industrial applications, we introduced in depth the main concepts of the RL model – agent, environment, action, task, state, reward, exploration and exploitation – and value functions. The following chapter included the IS interacting with environment and an example of multi-agent interaction. The next section contained explanations on algorithms such as: Q-Learning, – the most used one in simulation cases – Dynamic Programming, Monte-Carlo Model and Temporal-Difference learning. Beside the algorithms, we introduced the two action selection algorithms – greedy and softmax – and a couple of popular examples of IS learning abilities. In the end, in the section of Results (5) we observed more closely the RL application domain along with the two most recent forms of RL – Deep RL and Knowledge-based RL – and concentrated on the industrial test case of automated control through different RL in Quadrotors. After having introduced a short description of their architecture and usage, we focused on the RL algorithms used in Quadrotor control such as the RL in stochastic MDP model, ROS based on a model-based algorithm and the simulated control using the MATLAB quadrotor control toolbox. In more depth we viewed a case study on autonomous control of quadrotors via a model-free RL algorithm.

As such, we are confident to have covered the majority of the concepts regarding the RL and its application domain. The most practical approach has been described in the case studies using MATLAB for simulations in the field of quadrotors for learned autonomous movement using RL.
6.2 Limitations

Among the first identified needs, has been the one for hardware equipment in which we would test the MATLAB simulations containing the RL algorithms.

6.3 Future Work

The field of RL, as we discussed above, is obtaining a lot of attention due to its simple objective of learning by doing. It is a field that has existed for almost half a century, and it keeps expanding. As the conclusion on the survey on RL [10] articulates, there are a range of RL techniques that work effectively in a wide range of small problems. However, a few of these techniques scale well to larger problems and that is because it is difficult to solve subjective problems from objective (general) cases. The paper further recommends that in order to solve highly complex problems, we should incorporate bias in the varieties of these techniques that as a result will give leverage to the learning process.
REFERENCES


