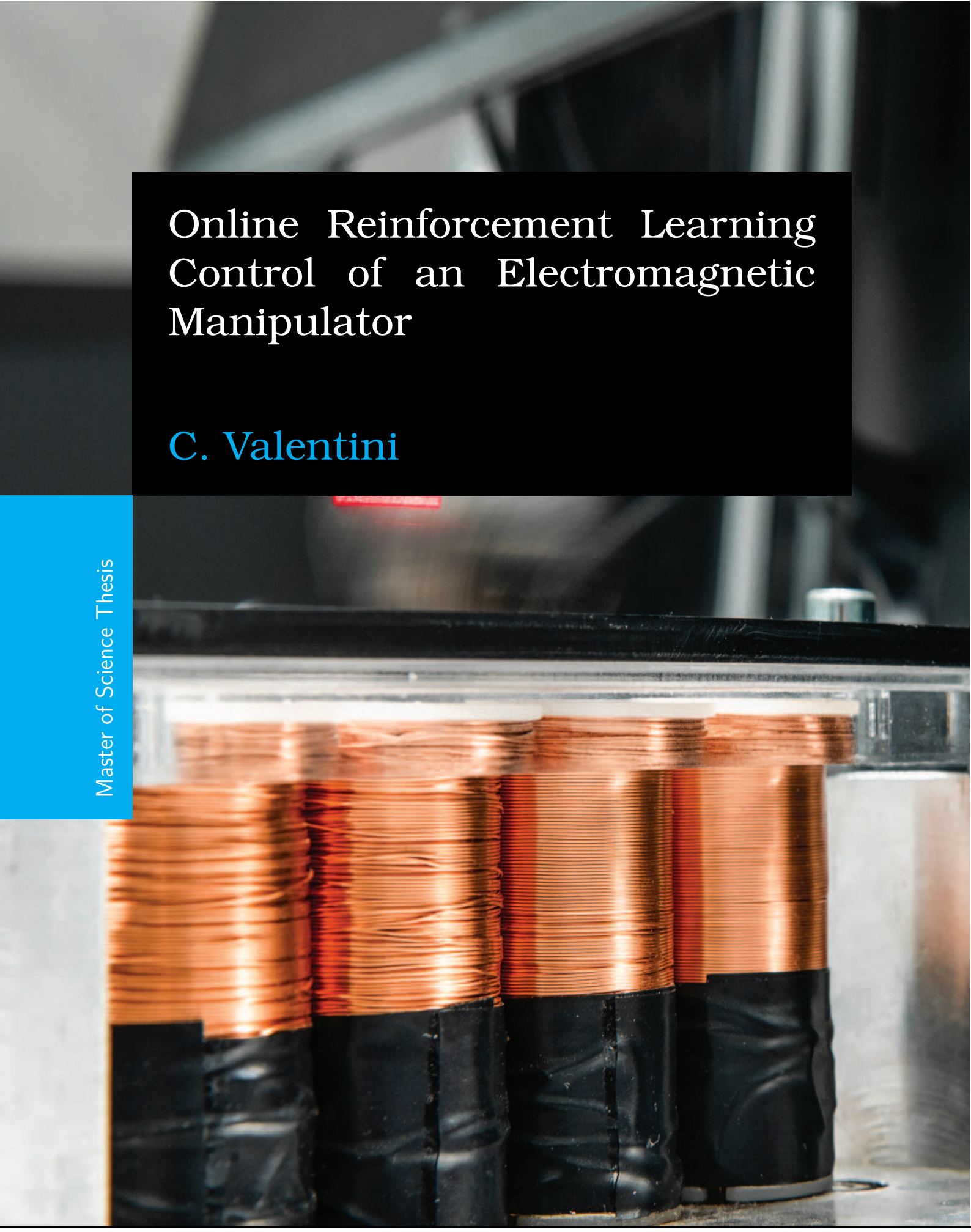
# Online Reinforcement Learning Control of an Electromagnetic Manipulator

C. Valentini

Master of Science Thesis



TU Delft Delft University of Technology

Delft Center for Systems and Control

# Online Reinforcement Learning Control of an Electromagnetic Manipulator

Master of Science Thesis

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

C. Valentini

August, 27, 2019

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


Delft University of Technology
Department of
Delft Center for Systems and Control (dcsc)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

Online Reinforcement Learning Control of an Electromagnetic Manipulator

by

C. Valentini

in partial fulfillment of the requirements for the degree of

Master of Science Systems and Control

Dated: August, 27, 2019

Supervisor(s):

Prof. dr. R. Babuška Supervisor

T.D. de Bruin Second Supervisor

Reader(s):

Dr. ing. J. Kober

Dr. ir. A.J.J. van den Boom

# Abstract

Machine Learning Control is a control paradigm that applies Artificial Intelligence methods to control problems. Within this domain, the field of Reinforcement Learning (RL) is particularly promising, since it provides a framework in which a control policy does not have to be programmed explicitly, but can be learned by an *intelligent* controller directly from real-world data, allowing to control systems that are either arduous or even impossible to model analytically. However, in spite of such considerable potential, the RL paradigm poses a number of challenges that effectively hinder its applications in the real-world and in industry. It is therefore critical that research in this field is advanced until RL-based controllers can be practically demonstrated to be real-world feasible and reliable. This thesis report presents the attempts made at applying control strategies based on Reinforcement Learning to solve a precise positioning task with a physical experimental setup. The setup at hand is a magnetic manipulator (magman) characterized by a high degree of nonlinearity. The controller uses the spatially continuous magnetic field generated by four actuators to displace a steel ball, constrained to move in one dimension, towards a reference position. Two different implementations of the Q-learning algorithm (Sutton, Barto, et al., 1998) were deployed. In spite of the good results obtained in a simplified simulated environment, both implementations failed on the experimental setup. The negative outcome of these experiments is mainly due to the fact that, since the task at hand is an accurate positioning task, the reward obtained by the learner while interacting with the environment is too sparse for it to be able to learn a stabilizing control policy. Other factors have presumably contributed to the controllers' failure, such as the circumstance that the agent does not have access to the full system state information and a sub-optimal tuning of the algorithms' hyper-parameters. Besides model-free RL, the Value Iteration model-based method was successfully applied both in simulations and with the experimental setup. The present findings suggest that, in order to solve the magman task with model-free RL, more sophisticated algorithms need to be deployed, such as for example an agent that can naturally deal with continuous state and action spaces, as the DDPG algorithm (Lillicrap et al., 2015), with exploration carried out in the parameter-space rather than in the control action space (Plappert et al., 2017), in addition to a more optimal exploitation of the information extracted from the environment, for example using Hindsight Experience Replay (Andrychowicz et al., 2017).

# Table of Contents

**Preface** v

**1 Introduction and motivation** 1
1-1 Research goal . . . 3
1-2 Thesis structure . . . 4

**2 The *Magman* control benchmark** 5
2-1 Experimental setup . . . 5
2-2 Dynamics model . . . 7
2-2-1 Validation of the legacy model . . . 8
2-2-2 New model and validation . . . 10

**3 Reinforcement learning** 13
3-1 Theoretical framework . . . 15
3-2 Model-based reinforcement learning . . . 17
3-3 Model-free reinforcement learning . . . 18
3-4 Value iteration methods . . . 18
3-5 Policy-based methods . . . 18
3-6 On-policy algorithms . . . 19
3-7 Off-policy algorithms . . . 19
3-8 Summary . . . 19

**4 Q-Learning** 21
4-1 The algorithm . . . 21
4-1-1 Exploration strategy . . . 22
4-1-2 The training algorithm . . . 22
4-1-3 Experience replay . . . 22
4-1-4 The target network . . . 23
4-1-5 Algorithm summary . . . 23
4-2 Model evaluation and tuning . . . 24
4-3 The grid-search method . . . 26
4-3-1 Grid-search for the ANN topology . . . 27
4-3-2 Grid-search for the remaining algorithm hyper-parameters . . . 28
4-4 On the choice of the reward function . . . 29
4-5 Performance in simulations . . . 36
4-6 Performance comparison to other controllers . . . 41
4-7 Performance on the experimental setup . . . 43
4-8 Q-learning - Second implementation . . . 48
4-9 Discussion of the results . . . 50
4-9-1 Control input transportation delay . . . 50

4-9-2 Markov Property . . . . . 51

**5 Conclusions and recommendation for future work** **57**
5-1 Conclusions . . . . . 57
5-2 Recommendations . . . . . 58

**A Value Iteration** **61**

**B Hardware** **65**
B-1 Position measurement . . . . . 65
B-2 Actuation . . . . . 68
B-3 Connection between control boards and computer . . . . . 70

**References** **71**

# Preface

This document is a report of the work carried out during my Master of Science graduation project. When I accepted this thesis assignment, my knowledge of the field of Machine Learning was close to non-existing, but my curiosity was great.

I would like to thank my supervisor, Professor R.Babuška for giving me the opportunity of working on this challenging project and providing guidance and valuable insight throughout, and I would like to thank T. De Bruin for co-supervising the thesis, providing valuable advice based on his experience in the field, besides the baseline implementations of the Machine Learning methods deployed.

I would also like to thank my family, for the unwavering support they gave me not only during the writing of this thesis, but throughout the years of my academic formation.

Finally, I would like to thank Kallia, for the wholehearted support, and my friends Andrea, Antonio, Arnau, Ben, Folkert, Francesco, Gianluca, Joris, Martijn and Tori, who contributed to making the last three years more enjoyable than I could possibly have hoped.

"In theory, there is no difference between theory and practice. But, in practice, there is."

— *Jan L. A. van de Snepscheut*

# Chapter 1

# Introduction and motivation

Magnetic manipulation consists of accurately controlling the movement (position and velocity) of a ferromagnetic object in an environment, by creating and shaping dynamically a magnetic field generated by the electric current running through one or more electromagnets. This invisible force is essential to a large part of modern technology, from every-day use loudspeakers to the leading-edge physics research carried out in the CERN Large Hadron Collider experiments. The contact-less nature of the electromagnetic force in fact allows interactions that would simply not be possible using traditional, contact force-based actuators.

Achieving accurate control of the electromagnetic force, however, is not a trivial undertaking. The high degree of nonlinearity that characterizes the electromagnetic actuators intrinsically calls for advanced control strategies that usually must rely on an accurate mathematical model of the system dynamics. Nonetheless, there are many control tasks for which it can be impractical or unfeasible to obtain such model. Magnetic manipulation is an example of such a task. Other examples include robotic skills such as dexterous manipulation or navigation in unknown dynamic environments, but also management of complex networks such as power grids or road networks. Modeling explicitly similar systems may either require very specialized domain expertise (and therefore be rather costly), or simply be impossible.

For such a system for which *first principles* modeling is not feasible, it is still possible to measure and collect Input/Output data that hold valuable information about the system itself. Data that can turn out to be even more valuable than an accurate system model. Reinforcement learning control tries to use this data to derive control strategies (policies) for complex systems (such as the electromagnetic manipulator used throughout this thesis) without using an explicit system model.

There are many practical advantages to such an approach.

- Reinforcement-learning control allows to tackle problems that would be impractical or unfeasible to tackle using traditional controllers.

- Theoretically, an intelligent agent is able to learn a policy directly from real-world data, with no need for a detailed system model, which would in turn require domain expertise, which is often expensive in terms of time and effort.
- In most cases, the performance of a model-based controller designed by a human will only be as good as the model of the system itself, which reflects the human understanding of it. On the other hand, an artificial agent strictly speaking *understands* nothing of the task it is dealing with. In this sense, it is not *biased* by prior knowledge, and this can result in the development of behaviors that, although they might not seem immediately intuitive to a human, would eventually lead to a better performance on the task. A remarkable example is the *AlphaGo* AI developed by Deepmind to play the boardgame *Go*, which eventually achieved super-human performance by pure reinforcement learning, and even learned some game strategies unfamiliar to the best human players (Silver et al., 2017).
- A well-tuned and stabilizing reinforcement learning controllers is intrinsically robust to changes in the system. Usually, when a controller is to be deployed on a physical system, its parameters are optimized to achieve the best performance possible. Real, physical systems, however, are changing continuously. For example, if the system of interest is a mechatronic system deployed in an industrial environment, the wear and tear of the mechanical components will affect the system dynamics in the long run. The original controller that was tuned for the original dynamics will no longer be able to get the most out of the system. A learning controller, on the other hand, will naturally adjust its policy without any human intervention, changing over time *together with* the physical system, and always getting the full benefit of an optimally tuned controller.

Having made these considerations, reinforcement learning control clearly seems to be a very appealing and promising paradigm, and to no surprise this field has drawn massive attention from the research community worldwide in the recent years. However, judging from the lack of real-world and industrial applications of this technology, one could be under the impression that we have barely dipped our feet into the ocean of potential held by RL. In truth, this control paradigm poses a number of challenges with respect to the application of classic control theory, and until the day these challenges are overcome, traditional controllers will be preferred.

- Most reinforcement learning algorithms and agents have a relatively large number of hyper-parameters (high-level design choices and/or tunable parameters of the agent) that affect the learning process. Finding an adequate model architecture/parameter configuration is a critical factor to achieve satisfactory performance, and such configuration largely depends on the characteristics of the particular task that the learner is confronted with. Attaining a suitable configuration, however, largely remains an empirical science (given the lack of theoretical foundations to guide most of the choices) and usually a tremendously time-consuming process (in an inversely proportional manner to the experience and skills of the machine learning practitioner).
- A reinforcement learning-based controller has always some intrinsic degree of stochasticity. Where a traditional controller essentially reads the system state, performs some predefined operations and computes an action in a deterministic way, the action that is selected by an intelligent agent is the result of a learning process studded with stochastic

elements, such as for example the random initialization of the policy at the beginning of the process, or the random sequence of control actions that allow the agent to explore its environment. Therefore, most of the process that leads the agent towards a policy remains largely unpredictable, and clearly an intelligent controller is not as consistent in delivering the same results in the same reliable fashion of a deterministic controller.

- The performance learned by an intelligent agent might not be better than the performance of a traditional, state-of-the-art controller. In some other cases, a marginal performance improvement simply would not justify the overhead of designing a reinforcement-learning based controller instead of a traditional controller, usually simpler to implement.
- Writing a machine learning algorithm is not a simple task. It is indeed possible to find an abundance of standard algorithms implementations and ML code in many public online repositories to get started with, but unless one is interested in a few hackneyed toy problems such as *CartPole* or *Pendulum swing-up*, actually implementing from scratches an algorithm for a custom application is an endeavor that requires a fair amount of expertise in the field.

Precisely *because* of the aforementioned drawbacks, control problems such as electromagnetic manipulation constitute a particularly interesting challenge for academic research on machine learning methods. A previous attempt at applying reinforcement learning to the *magman* manipulator is reported in (J. Damsteeg, 2015). In spite of the fact that a reasonable performance level was achieved using an actor-critic algorithm in the simulated environment, the agent, proved unable to converge to a stabilizing policy on the experimental setup. Furthermore, it is noteworthy that the performance (in simulation) of the intelligent agent compared negatively (in terms of settling time, overshoot and control effort) with respect to other nonlinear controllers deployed.

Because of the interesting properties of this peculiar magnetic manipulator setup, it has been used to benchmark both traditional, nonlinear controllers (J.-W. Damsteeg, Nageshrao, & Babuska, 2017) and cutting-edge reinforcement learning controllers (Alibekov, Kubalík, & Babuška, 2018), (De Bruin, Kober, Tuyls, & Babuška, 2018). The theoretical results achieved by the aforementioned research in the RL field, however, were not validated by real-world experimental results.

In the remainder of this thesis, reinforcement learning algorithms will be applied to solve the problem of achieving linear electromagnetic regulation, in an attempt to bridge the gap between what should theoretically be possible and the real-world.

## 1-1 Research goal

The first goal of the Master thesis presented in this report is to implement and deploy a state-of-the-art controller based on Machine Learning methods to solve a regulation problem using a linear magnetic manipulator.

The controller will at first be deployed in a physics simulation environment and then on the experimental setup.

Finally, the controller performance will be compared to the performance of traditional non-linear controllers.

## 1-2 Thesis structure

The remainder of this thesis is structured as follows:

- **Chapter 2** describes the linear magnetic manipulator experimental setup and its dynamics model used throughout this thesis work.
- **Chapter 3** briefly introduces Reinforcement Learning (RL) and its main paradigms.
- **Chapter 4** presents the Q-learning algorithm and the results obtained both in simulations and on the experimental setup.
- **Chapter 5** contains the conclusions drawn from this thesis project and some recommendations for future work.
- **Appendix A** presents Value Iteration, a model-based RL method that was also applied to the control task at hand, both in simulations and on the experimental setup.
- **Appendix B** gives some details on the hardware used and on the I/O interface.

# Chapter 2

# The *Magman* control benchmark

The *magman* (short for magnetic manipulator) control benchmark shown in Fig. 2-1 was used throughout the thesis work presented in this report.

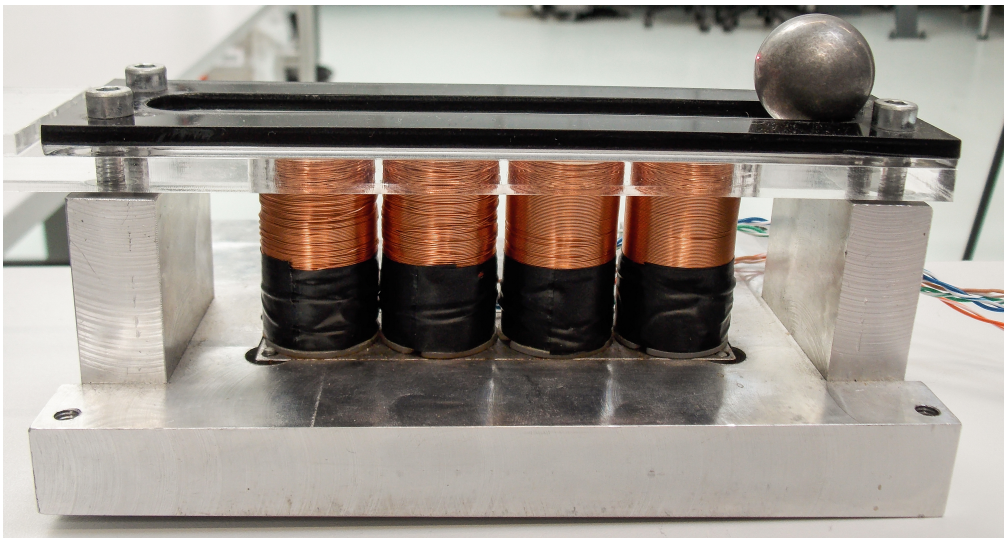


**Figure 2-1:** The *magman* experimental setup.

## 2-1 Experimental setup

The magman experimental setup consists of four linearly arranged electromagnets and a steel ball that can move along a rail-like track positioned above them. The track constrains the ball movement to a single dimension.

This setup has been widely used within the TU Delft DCSC (Delft Center for Systems and Control) and CoR (Cognitive Robotics) departments for didactic purposes, as well as to serve as nonlinear control test-bench and technology demonstrator.

The two state variables are the ball position (measured by a laser sensor) and velocity (that is not measured directly but needs to be reconstructed from the position measurements). The four coils are used to generate a magnetic field that can be shaped to put the steel ball in motion and steer it towards a reference position or to track a reference state. Therefore, the setup is treated as a MISO (multiple-input, single-output) nonlinear dynamic system.

Appendix B can be consulted for some details on the system I/O interface.

Fig. 2-2 shows a schematic representation of the setup. The laser sensor is positioned in such a way that its measuring range coincides with the mono-dimensional rail along which the ball can move. The four coils are positioned at approximately 0.025 m, 0.050 m, 0.075 m, and 0.100 m from the beginning of the rail (point A in Fig. 2-2), where the laser sensor measures 0 m.

Summarizing, the state and action spaces of the magman system are the following:

- The position is limited in the interval $[0.0, 0.125]$ m.
- The length of the rail limits the velocity in the interval $[-0.4, 0.4]$ m/s.
- The control action for each coil is a continuous value in the interval $[0.0, 0.55]$ A$^2$.

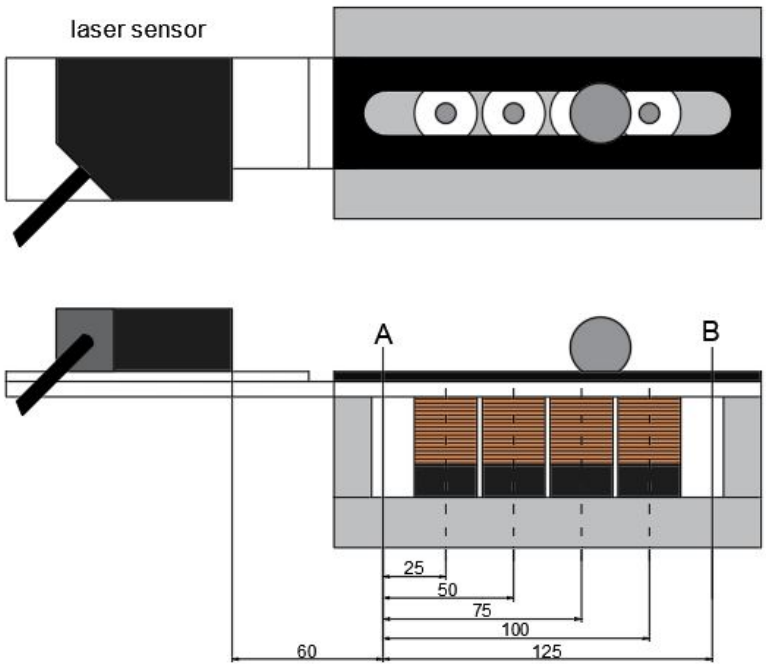



**Figure 2-2:** A schematic representation of the magman setup. The two points $A$ and $B$ indicate the ball position range.

On a final note on the experimental setup, it is worth mentioning that it is essentially a simplified version of the planar electromagnetic manipulator developed by Zemánek and Hurák at the Czech Technical University shown in Fig. 2-3. See (Hurák & Zemánek, 2012) and (Zemánek, Čelikovskỳ, & Hurák, 2017) for additional details.

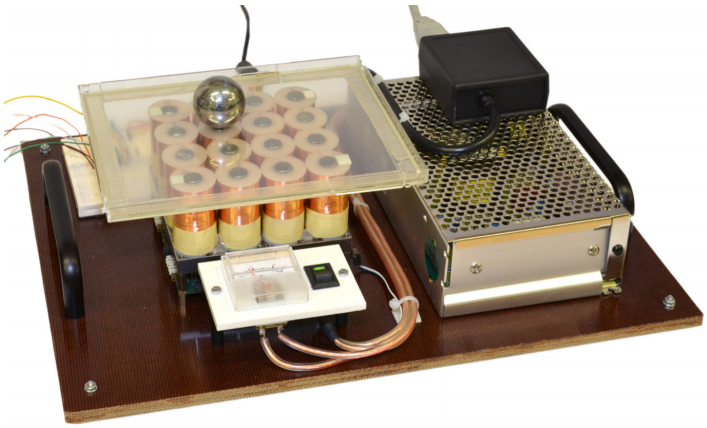


**Figure 2-3:** The planar magnetic manipulator setup. The ball rolls over a flat surface under the influence of the electromagnetic field generated by the combined action of 16 coils.
**Image source:** (Simonian, 2014).

## 2-2 Dynamics model

As stated in the introduction chapter, the control strategies that will be deployed do not require an accurate system model, strictly speaking. However, a mathematical model of the system is of critical importance during the design and verification phases of the learning algorithm, and is also instrumental to reducing the time necessary to perform model optimization.

The local interactions between an electromagnetic field and a ferromagnetic object have a highly nonlinear nature. To further complicate the dynamics of the system, the dependency of the intensity of the magnetic field generated by an electromagnet on the magnitude of the current running trough its coils is also non-linear. Modeling these interactions from first principles is a non-trivial task that would require advanced vector calculus analysis.

Studies on a setup using the same kind of actuator (Simonian, 2014) used an empirical approach to identify the following force model for a single coil:

$$F(d, I) = g(d)I = \frac{-\alpha d}{(d^2 + \beta)^3} I \tag{2-1}$$

With $d$ the distance of the center of the steel ball from the center of the coil, $I$ the intensity of the current running through the coil and $\alpha$ and $\beta$ two dimensionless parameters depending on the physical properties of the ball and of the electromagnet. This model makes a simplification assuming linear dependency of the exerted force on the current intensity, but higher order dependencies are commonly used to describe such systems more accurately (Hammond, 2013).

Fig. 2-4 is a visual interpretation of the relation in Eq. 2-1.

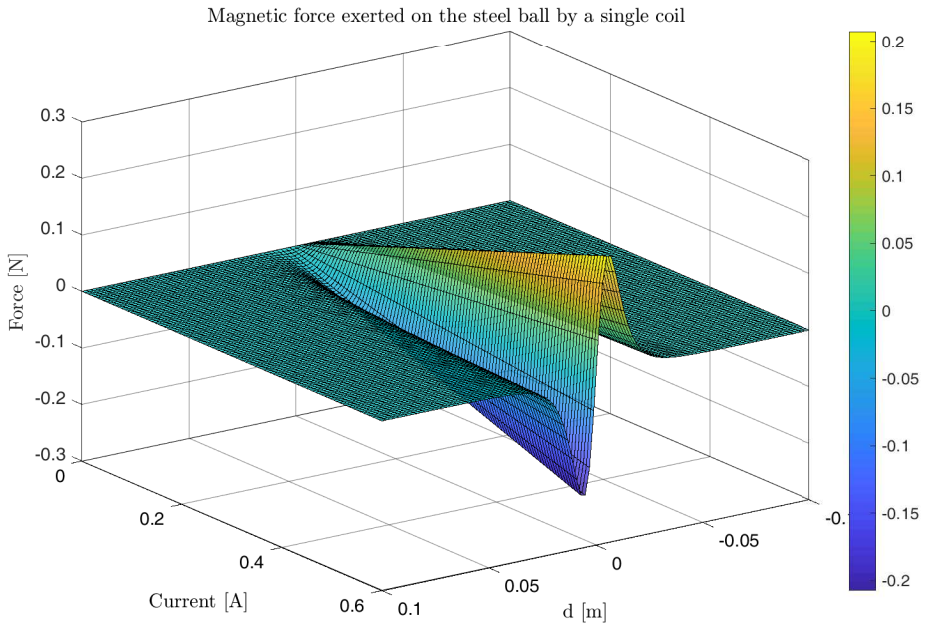



**Figure 2-4:** 3D visualization of the nonlinear function linking the magnetic force exerted on the steel ball to the current running through a single coil and on distance of the ball from the center of the coil, according to the empirical model identified in (Simonian, 2014).

Since the coils have a similar construction, it is reasonable to assume that this model, with the same parameters, can be used to represent the behavior of all the coils.

A further simplification that can be made is to assume that the superposition principle holds for the force exerted on the ball by the spatially continuous magnetic field generated by the four currents running through the four coils. In other words, we are assuming that the total magnetic force exerted on the steel ball is equal to the sum of the four individual forces that would be caused by the action of each of the four coils *independently* from each other.

It needs to be kept in mind, however, that the superposition principle is a property of linear systems, but the effects of a dynamic magnetic field such as the one generated by the four coils of the magman on a rigid body subject to physical constrains are highly nonlinear. Such force is in fact a nonlinear function of the magnetic field, which is in turn a nonlinear function of the electric currents flowing through the electromagnet. The empirical model therefore makes quite a relevant simplification, and this is reflected in the results of an open loop model validation.

### 2-2-1 Validation of the legacy model

With the assumptions and simplifications discussed, and assuming a simple viscous friction model, the dynamics of the magman system are described by the following set of differential equations:

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ -\frac{b}{m}\dot{x} + \frac{1}{m}\sum_{j=1}^{4} g(x,j) I_j \end{bmatrix} \tag{2-2}$$

The following parameters were used by a legacy model of the system:

- $\alpha = 5.52e - 10$
- $\beta = 1.74e - 4$
- $b = 0.0161$
- $m = 0.0563$ (Please note that the legacy model actually used 0.032 kg for the ball mass, since the ball used was different)

This system model was used in the past for the design of model-based nonlinear controllers, see (J.-W. Damsteeg et al., 2017) for the details.

A Python implementation of the magman and other control benchmarks, developed by T. De Bruin, can be found at https://github.com/timdebruin/CoR-control-benchmarks.

Fig. 2-5 shows a comparison between the system step response of the experimental setup and the response of the legacy model. The Variance Accounted For (VAF) (Verhaegen & Verdult, 2007) is used to assess model accuracy, defined as:

$$VAF = 100 \cdot (1 - \frac{\text{var}(y - \hat{y})}{\text{var}(y)}) \tag{2-3}$$

Where $y$ is the output of the real system and $\hat{y}$ is the output of the system model. The closer this value is to 100%, the more similar the identified model is to the actual system.

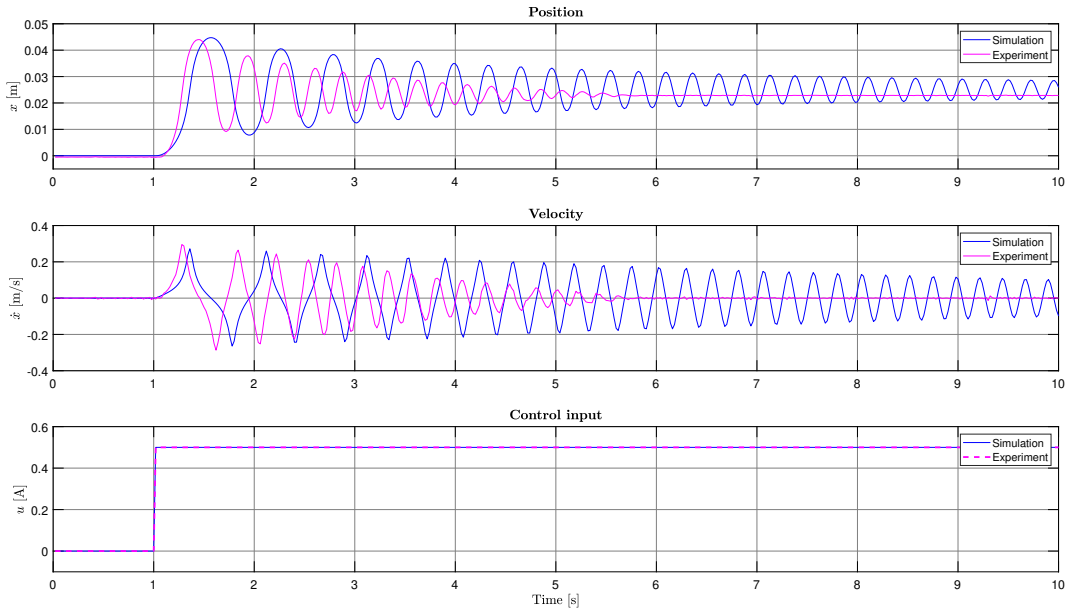



**Figure 2-5:** Step response open-loop validation of the legacy magman model - One coil. **VAF** = 43.4124%

Fig. 2-6 shows a comparison of the step response when two coils are actuated. As it can be seen, there is a more accentuated nonlinearity that is not captured by the legacy model. The mismatch between the model output and the output of the experimental setup is significant, which is reflected in the low VAF of 4.1480 obtained for the step response.

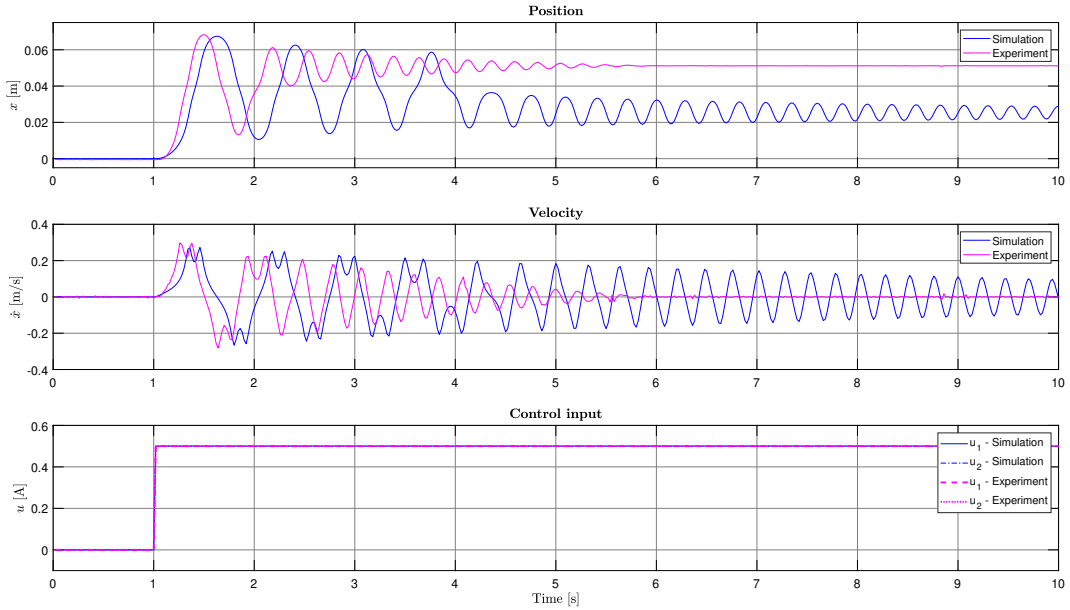



**Figure 2-6:** Step response open-loop validation of the legacy magman model - Two coils. **VAF** = 4.1480%

As it can be seen from Fig. 2-5 and Fig. 2-6, this model is not quite representative of the actual system. Eventually, however, it will only be used to provide a quick prototyping environment to aid the implementation and testing of the learning algorithms, and not by the controller itself. Therefore, having a mathematical model that is not completely accurate but can still be considered illustrative of the system under investigation should not be a hinder to the success of a learning agent.

Nevertheless, it was considered desirable to identify a new system model, closer to the actual system.

### 2-2-2 New model and validation

The model in Eq. 2-2 was improved with a Coulomb friction model and a force model that takes into account the *true* position of the coils, rather than the nominal positions. The new model is the following:

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \frac{1}{m}(-f_b\dot{x} - f_c\text{sgn}(\dot{x}) + u_1(-\alpha\frac{x-0.0233}{((x-0.0233)^2+\beta)^3} + u_2(-\alpha\frac{x-0.0505}{((x-0.0505)^2+\beta)^3} + \\ u_3(-\alpha\frac{x-0.0770}{((x-0.0770)^2+\beta)^3} + u_4(-\alpha\frac{x-0.1040}{((x-0.1040)^2+\beta)^3}) \end{bmatrix} \tag{2-4}$$

With the following parameters:

- $\alpha = 2.0821 \cdot 10^{-9}$
- $\beta = 3.0289 \cdot 10^{-4}$
- Viscous friction coefficient $f_b = 0.0087$
- Coulomb friction coefficient $f_C = 0.0034$

- Steel ball mass $m = 0.0563\,\text{kg}$

Fig. 2-7 and Fig. 2-8 show that this model is much more representative of the actual system with respect to the legacy model, specifically on the step response with two different coils, in which the VAF value increases from 4.1480% to 80.5508%.

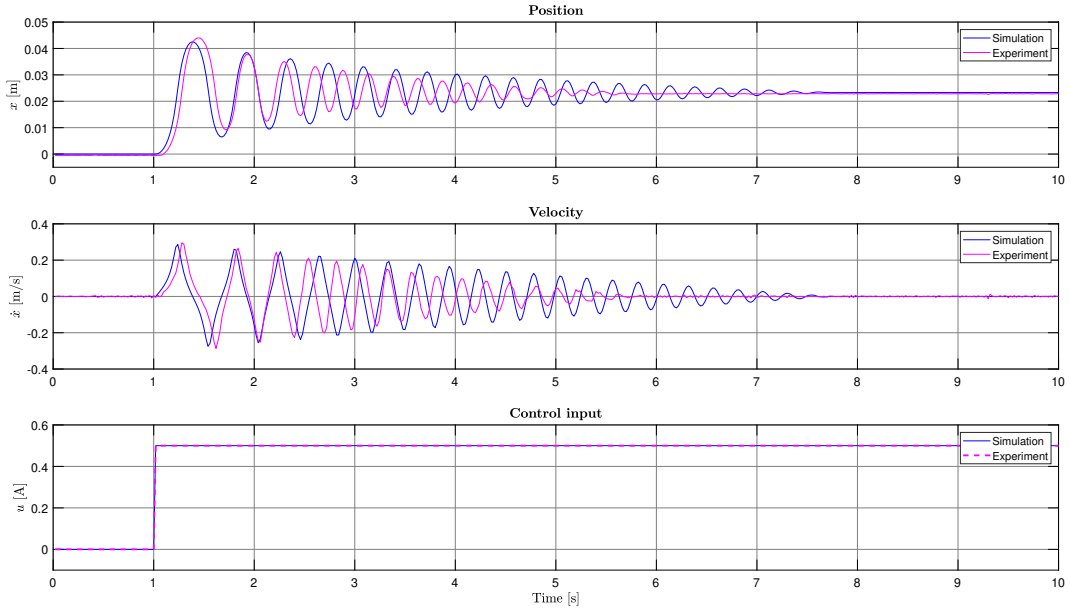



**Figure 2-7:** Step response open-loop validation of the new magman model - Single coil. **VAF** = 70.8411%

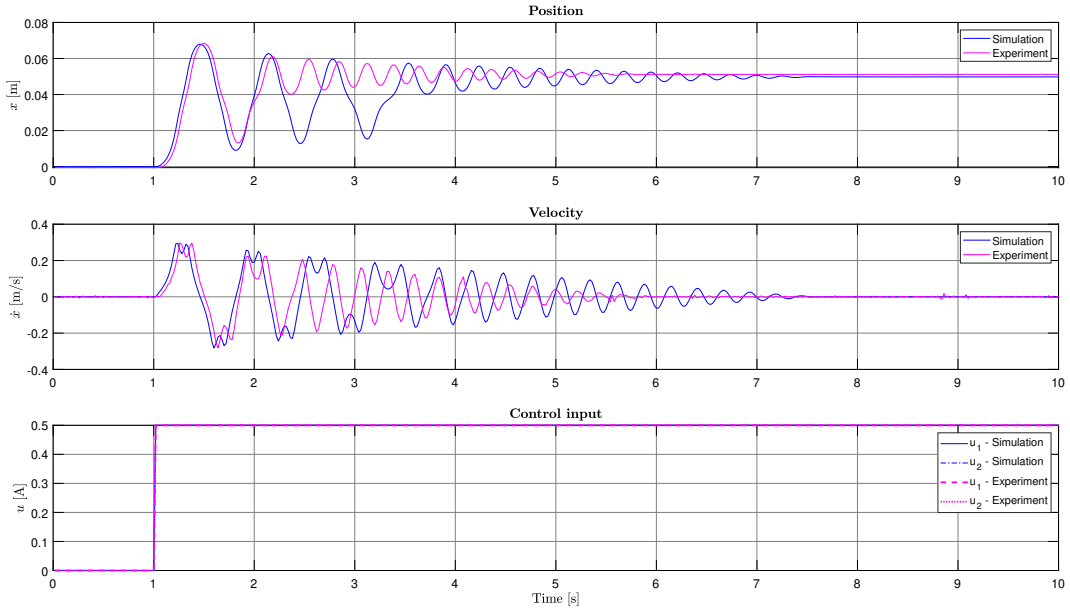



**Figure 2-8:** Step response open-loop validation of the new magman model - Two coils. **VAF** = 80.5508%

In spite of the fact that the new model is indeed a closer match to the actual system with

respect to the legacy model, the system dynamics are still not captured entirely.

However, getting a truly accurate model of the magman is unpractical and would require a considerable effort. This is one of the reasons why it is desirable to deploy a model-free reinforcement algorithm as an alternative to traditional control methods.

# Chapter 3

# Reinforcement learning

Reinforcement learning is a field of machine learning, part of the larger domain of Artificial Intelligence. It can be defined as the art (and the science) of *teaching* a machine how to perform complex tasks with no explicit programming. The machine shall be able to learn from data extracted from the real world, improve its performance by interacting with its environment and eventually become able to generalize the learned behavior to novel situations.

Research in this field has lately known a significant growth, both in the academia and in the industry, with an ever-increasing number of researches being published and investments being made. The enthusiasm for research in this domain is fueled by results that showed in an unprecedented way the potential in machine learning to equal and exceed human-level performance on a number of specific tasks. Among many other important accomplishments, it is worth mentioning the following three:

- The *Watson* Artificial Intelligence developed by IBM, that won the popular television-streamed game *Jeopardy!* playing against human players (Ferrucci, 2012).
- The *DQN* (Deep Q-Network) agent developed by Deepmind, that bested the top human video-game players on most of the Atari 2600 console games (Mnih et al., 2015).
- The *AplhaGo* AI, also developed by Deepmind, that repeatedly beat the best human players on the board game *Go* (Silver et al., 2016).

Conceivably, machine learning has also penetrated the realm of control engineering, leading to the development of the Machine Learning Control (MLC) field. MLC is a rather vast concept, with research branching in different directions and towards different paradigms. The three main paradigms are the following:

- Supervised Learning, where the agent learns an input/output mapping from *labeled* training data. Supervised learning provides an excellent tool to solve a broad range of classification problems such as medical diagnostics, handwriting recognition, face recognition etc.. However, supervised learning *alone* is not really useful for control applications.

If it was in fact possible to make an *a priori* distinction between good and less good control actions or, in other words, if labeled data was available, a good control strategy would be already known, and machine learning could only lead to some marginal improvements. This was the case of the agent based on imitation learning (a branch of supervised learning) that is described in (J. Damsteeg, 2015). This controller successfully learned a control policy imitating a tuned-down model-based controller. However, the performance of the imitation learning controller was worse with respect to the performance of a version of the model-based controller before it was down-tuned.

- Unsupervised Learning, where the agent is trained on *unlabeled* data. The goal of unsupervised learning is to extrapolate the properties of an input dataset (distribution), without any sort of feedback from a teacher/supervisor. The unsupervised learning approach (*by itself*) is also not suitable to deal with a dynamic control problem, since it is crucial that the behavior of the agent is reactive to the feedback received. However, unsupervised learning could be used for example to pre-cluster the system state space in a number of sub-spaces, and the global control policy could be the one resulting from the integration of local policies defined by means of different strategies over the state space segmentation.

  Furthermore, unsupervised learning could be used to train Kohonen self-organizing maps (Kohonen, 1982), that are able to represent the training sample in a low-dimensionality space while maintaining the topological properties of the input space. Once the system model is identified in this way, it could then be used to carry out model-based controller design. Techniques such as the vector-quantized temporal associative memory (VQTAM) algorithm developed in (Barreto & Araujo, 2004) are also based on self-organizing maps, but in the literature there is no report of the application of this particular strategy to a real control problem.

- Reinforcement Learning is a machine learning paradigm that was originally inspired by behavioral psychology and ethology (animal behavior). The neurological evidence produced in these fields inspired computational models that seem to be able to process high dimensional information in a similar way to the human brain. Chapters 14 and 15 of (Sutton & Barto, 2018) make an interesting parallel between psychology concepts and engineering.

  Reinforcement learning agents do not require any explicit programming (in the traditional sense) to carry out a task, but learn directly from interaction with an environment (or exposition to datasets). Besides control tasks, reinforcement learning is also increasingly being used to solve optimization problems (Sra, Nowozin, & Wright, 2012). If the RL algorithm uses deep artificial neural networks (ANNs) as function approximators, it is referred to as deep reinforcement learning.

Out of the three, reinforcement learning is conceivably the most fascinating approach, and definitely the one that seemed to hold the greatest potential to solve the control problem at hand, considering its characteristics. The next section will briefly describe the Reinforcement Learning process and its framework.

## 3-1 Theoretical framework

The idea at the base of RL is the following: the RL agent observes the state of the environment that it needs to solve and takes an action according to its internal control policy. Then, it observes the consequences of its actions on the state of the system and adapts its behavior accordingly, based on its programming. The effects of the learner's actions are quantified in a *reward* that implicitly shapes the agent's behavior.

In this, the reinforcement learning *agent* is a state-feedback controller, and its *environment* is the magman experimental setup. The reinforcement learning goal is for the agent to learn a control strategy, also referred to as *policy*, that allows it to perform a regulation task.

The system state, observed at discrete time intervals, is denoted as $s_k$, and the control action determined by the deterministic policy $\pi$ is denoted by $a_k = \pi_k(s_k)$. During a training episode, the agent interacts with the environment as follows:

- Observe the system **state**, $s_k$
- Take an **action**, $a_k$
- Observe the system **new state**, $s_{k+1}$
- Get an immediate **reward** for the interaction, $r_{k+1}$

The *reward* is the feedback that agent receives from the environment for its *immediate* performance. It is a scalar function of the system state and control action. The sum of all the immediate rewards over the course of a training episode is referred to as *return*. The agent learns a control policy by optimizing the reward gained from the system during the interactions.

Fig. 3-1 illustrates the interaction between agent and environment in the context of reinforcement learning.

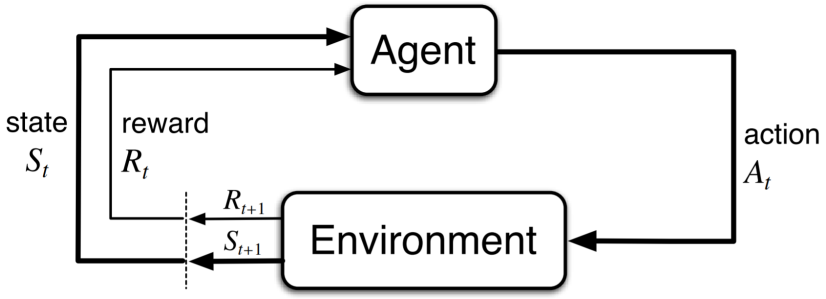



**Figure 3-1:** A scheme of the RL framework. **Image source:** (Sutton et al., 1998).

An important assumption of RL is that the task at hand is a so-called *Markov Decision Process* (satisfying the Markov Property (Puterman, 2014)), which means that the environment next state $s_{k+1}$ only depends on its current state $s_k$ and the action taken by agent, $a_k$, and it is therefore *independent* of previous system states and previous decisions of the agent. In other words, the RL task needs to be causal and deterministic.

### Drawbacks of the approach

There are, however, a number of concerns about the applicability and actual *usefulness* of Reinforcement Learning. A number of issues are known to be affecting the RL in general.

- Machine Learning is sample inefficient. In many cases, a reinforcement learning agent might need a very large amount of data.

  For many difficult problems, the number of samples necessary is well beyond a practically acceptable level. In the case of the AlphaGo Zero agent (Silver et al., 2017), 29 million games and 40 days of continued training on 64 GPU workers were necessary to learn super-human performance. Such resources are well beyond what is normally available to researchers.

- As a consequence of the sample inefficiency, the fact that many interactions with the environment are needed substantially hinders many real-world applications, as in the case of a robotic manipulator, for example, that would naturally suffer from wear and tear and which maintenance costs would not justify the benefits of a learning controller (Kober, Bagnell, & Peters, 2013).

- While RL is rightfully considered a fascinating research topic, the performance on many control tasks of a ML-based controller is in most cases unlikely to be better than the performance of a traditional, fine-tuned controller based on a domain-specific algorithm. As an example, a Monte Carlo tree-search-based method (Guo, Singh, Lee, Lewis, & Wang, 2014) has been shown to out-perform significantly the already mentioned DQN agent (Mnih et al., 2015) on the Atari Arcade Learning Environment. When domain-specific knowledge is available, model-predictive control can be applied to solve complex tasks such as locomotion. In (Tassa, Erez, & Todorov, 2012), the model-predictive paradigm is used to control the motion of a humanoid puppet in the MuJoCo physics engine (Todorov, Erez, & Tassa, 2012). Other papers such as (Heess et al., 2017) deployed Reinforcement Learning on a similar locomotion task, but most of the learned gaits look rather *unnatural*, although the puppet is still able to navigate through complex environments.

- ML algorithms are known to be very sensitive to changes in hyper-parameters and other high-level design choices such as the topology of the ANN (if one is used), etc.. Tuning such an algorithm is not a trivial task, requires considerable amount of expertise and is a time-consuming process.

- The reward function is a critical factor that determines success at correctly solving the environment. A misspecified reward function might in fact bias learning and is bound to make the agent fail to learn the desired behavior. In many practical cases, designing a reward function is not an easy task, as it is widely reported in the literature. For example, a reward function that is defined too loosely might lead an optimizing RL agent to learn quite different behaviors with respect to the desired ones, as reported in (Popov et al., 2017), where the task at hand is robotic dexterous manipulation. The field of evolutionary reinforcement learning (Lehman et al., 2018) provides plenty of other examples in which the agent learns to *trick* the specified reward (*reward hacking*). In other cases, the agent might simply fail at its task as a result of a poor trade-off between exploration and exploitation, therefore learning to exploit local optima generated in turn

by a poorly specified reward function.

- Another drawback affecting Reinforcement Learning specifically is what is often referred to as the *exploration vs exploitation dilemma* (Sutton et al., 1998). In control applications, the RL algorithm essentially boils down to solving a nonlinear optimization problem by finding the policy that maximizes the *return* (the discounted sum of rewards expected from the system). As with all nonlinear optimization problems, there are often many different local optima, and the RL agent might get irredeemably *stuck* in one of those points. Therefore, the agent needs to be able to *explore* its action space, while still *exploiting* what it has learned so far. On the other hand, if the scales tips too much in favor of exploration, the agent will never converge to a stable policy.

Academic research is continuously striving to alleviate these and other obstacles to a widespread use of RL in a number of different ways, leading to the development of various classes of algorithms.

The first major distinction within the reinforcement learning paradigm can be made between *model-based* and *model-free* algorithms, depending on weather a model of the environment is used or not.

## 3-2 Model-based reinforcement learning

Model-based reinforcement learning (MBRL) is a popular research branch in RL. A model-based agent explicitly uses a model of the interaction with the environment. In other words, it predicts or approximates the expected environment response to its possible actions. The optimal action or policy is derived from this model with an optimization over the action space. The environment response is then observed, and the policy is updated accordingly, as summarized in Fig. 3-2.

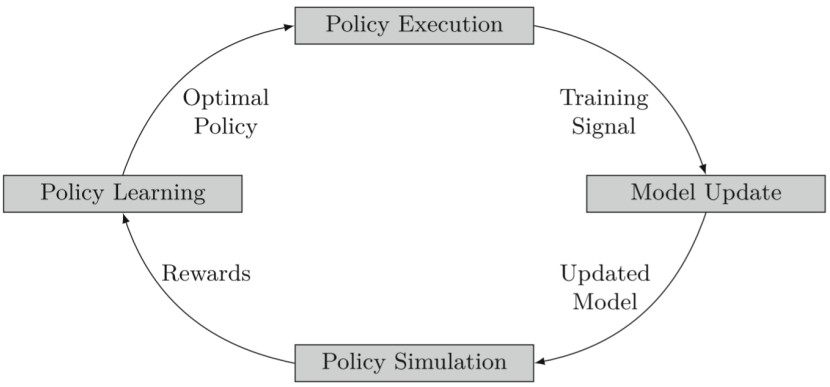



**Figure 3-2:** Pipeline of a model-based RL algorithm.
**Image source:** (Polydoros & Nalpantidis, 2017).

The literature concerning model-based reinforcement learning is extensive. This paradigm has been successfully applied to several control tasks, with many real-world experiments ranging from simple *toy* problems such as pendulum swing-up (Atkeson & Santamaria, 1997) to more complex tasks such as agents deployed to learn learn hovering and acrobatic maneuvers (Kim, Jordan, Sastry, & Ng, 2004) of a small helicopter. Numerous applications of Model-Based

Reinforcement Learning are also found in the field of robotics (Kober et al., 2013), (Polydoros & Nalpantidis, 2017), where the number of environment interactions necessary for the agent to learn a good policy constitutes a significant cost factor.

The main advantage of model-based RL over direct RL lies in the fact that the number of physical interactions necessary for the agent to learn a policy is reduced, since the agent can use its internal model for learning. This feature makes MBRL particularly appealing for those applications in which exploratory behavior might be not desirable and/or expensive. This advantage comes at the cost of a strong dependence on the environment transition dynamics model maintained by the agent, which is not easily obtained for systems such as the magman, as aforementioned. Furthermore, the policy simulation and optimization process might be computationally expensive for controllers with large action spaces, which might be a limiting factor for a multiple-input, real-time, continuous system such as the magnetic manipulator.

## 3-3 Model-free reinforcement learning

In order to break the dependency of the agent on a transition model and also in order to overcome the difficulties in exploiting the available problem-specific information, it was decided to deploy model-free reinforcement learning to solve the magman problem.

A model-free agent does not rely on predictions made by an internal model of the system to define a control policy. Q-learning methods such as DQN (Mnih et al., 2015) and policy gradient methods such as REINFORCE (Williams, 1992) are two examples of model-free algorithms.

An important distinction within the Model-free paradigm can be made between value iteration (VI) and Policy gradient-based methods. Yet another distinction can be made between on-policy and off-policy algorithms. In the following these definitions will be briefly explained.

## 3-4 Value iteration methods

Value iteration methods iteratively approximate a value function, while learning the control policy at the same time. A value function essentially tells the agent how good it is to be in a certain state, or alternatively how good it is to be in a certain state and take a certain action. A state value function is denoted as V(s) and a state-action value function is denoted as Q(s,a). The value function of the MDP is of course not known *a priori*, and needs to be

When the MDP has a limited number of states and actions, the estimate of Q(s,a) is usually kept in a table, for example in the case of grid-world toy problems, or otherwise it can be parametrized by an ANN. The value function is updated as the agent interacts with its environment.

## 3-5 Policy-based methods

Alternatively, the policy can be searched directly. Usually, the policy is parametrized by a DNN.

Policy-based methods are clearly more suited for continuous control applications with respect to state-value and state-action value functions, which are more appropriate for inherently discrete problems. The concept of deterministic policy gradient (DPG) was introduced in (Sutton & Barto, 2011), (Silver et al., 2014). The proposed method maximizes the expected total return by estimating the policy gradient and moving their parameters in the direction of gradient ascent (Schulman, Moritz, Levine, Jordan, & Abbeel, 2015). In other words, a DPG agent does not try to find an approximation of the optimal Q value function, but parametrizes directly the control policy $\pi$ as $\pi_\theta(s,a) = \mathcal{P}[a|s,\theta]$ as a probability distribution over the actions by manipulating the parameters $\theta$. The parameters are updated in such a way that the policy moves in the direction that maximizes the return (gradient ascent strategy) (Williams, 1992) (Baxter, Bartlett, et al., 2000). Since the DPG agent follows the positive direction of the gradient, it usually has very good convergence-properties, but on the other hand it is bound to remain stuck in a local optimum in case the exploratory behavior is not handled correctly. This agent is also known for its good learning stability properties, due to the fact that the policy is updated smoothly and incrementally. Furthermore, since the policy is directly parametrized, it is not necessary to carry out the computationally expensive optimization step over the whole action space necessary for a Q-learning agent. A policy-based method can therefore be implemented effectively for a continuous action space such as the magnetic manipulation setup.

## 3-6 On-policy algorithms

These algorithms usually attempt to improve the policy that is currently used for selecting the control actions. Example of on-policy algorithms are REINFORCE, SARSA and actor-critic schemes (Konda & Tsitsiklis, 2000). As already mentioned, an actor-critic algorithm was deployed on a simulated environment of the magman in (J. Damsteeg, 2015), but failed when deployed on the experimental setup.

## 3-7 Off-policy algorithms

The agents based on off-policy algorithms implement a different policy with respect to the one that is being optimized. The main advantage of these methods over the on-policy algorithms is that they are generally more sample-efficient.

Furthermore, since the experiences are collected following a policy that can be completely arbitrary with respect to the one that is being optimized, it is possible to explore the action space in ways that could not be possible if an on-policy strategy was followed.

## 3-8 Summary

Eventually, the algorithms deployed for control on the magman setup were:

- The offline, off-policy, model-based **Value Iteration** (VI) method, described in Appendix. A
- Two different online implementations of the model-free, off-policy **Q-learning** method, described in Chapter 4

Eventually, some experiments were also made with baseline implementations of the **Deep Deterministic Policy Gradient** (DDPG) and the **Normalized Advantage Functions** (NAF), but are not described in detail this paper. The main conclusions drawn from these attempts is that Deep Learning proved to be less suitable with respect to simple Q-learning to solve the task at hand.

# Chapter 4

# Q-Learning

The first machine learning algorithm deployed in an attempt to solve the magman problem was an instance of the DQN (Deep Q-Networks) algorithm (Mnih et al., 2015). The DQN agent is widely considered to be one of the most remarkable contributions to the field of ML, since it demonstrated for the first time that (deep) reinforcement learning could be used to successfully learn control policies for a complex environment, such as the Atari arcade games, directly from high-dimensional sensory input.

## 4-1 The algorithm

This method belongs to the class of value-iteration methods. The $Q$-function (value function) is a function of the environment state $s$ and of the control actions $a$. The function output is the expected return obtained from the environment if the agent takes action $a$ when the system is in state $s$ and it follows the policy greedily from that moment on.

The optimal $Q$-function, denoted as $Q^*$ is of course not known a priori, but the DQN agent is able to improve its approximation of it trough the rewards obtained while interacting with the environment. The $Q$-values are updated iteratively using the Bellman optimality equation (E. 4-1), until the approximation of the $Q$-function that is kept by the agent converges to the optimal $Q^*$-function.

$$q(s,a) = E\left[R_{k+1} + \gamma \max_{a'} q(s',a')\right] \tag{4-1}$$

Once the $Q$-function is a good approximation of the optimal $Q^*$-function, a good control policy follows as a consequence, as it is simply the action that maximizes the $Q$-function in every state:

$$\pi = \max_{a} Q(s,a) \tag{4-2}$$

In the DQN algorihtm, this value function is parametrized by a (deep) neural network.

A few expedients that will be described in the following subsections are necessary to make the DQN algorithm work in practice.

### 4-1-1 Exploration strategy

For the sake of simplicity, an $\epsilon$-greedy exploration strategy was used. This basic form of undirected exploration selects a random action (instead of the *greedy* action, associated to the highest Q-value in a certain state) with a certain probability $\epsilon \in (0, 1)$. As the value function is being learned by the agent over time, the probability of selecting a random action is annealed (linearly or asymptotically) in order to balance exploration of the environment and exploitation of the learned policy.

### 4-1-2 The training algorithm

The training algorithm used, also referred to as the optimizer, is the adaptive moment estimation (*adam*) algorithm (Kingma & Ba, 2014). It is an extension of the stochastic gradient method which maintains separate learning rates for each weight of the value function parametrization. It has been empirically shown that the adam optimizer performs better with respect to other training algorithms such as AdaGrad, RMSProp, and is often suggested as the standard choice for solving deep learning problems.

### 4-1-3 Experience replay

Machine learning training algorithms based on stochastic gradient descent as the *adam* optimizer used in this case require to be trained on i.i.d. data samples. However, the sequence of states and actions visited by a reinforcement learning agent as it is interacting with its environment is naturally highly correlated.

Experience replay (Lin, 1992) was introduced as a strategy to break the correlation in the training data, increasing the stability of the learning process and improving the chances of convergence to a stabilizing policy.

An experience replay buffer is essentially a database that holds the agent's past *experiences*, where each experience is defined as a tuple that describes an interaction with the environment. At time-step $t_k$, an agent in state $s_k$ performing an action $a_k$ reaches state $s_{k+1}$ and receives a reward $r_{k+1}$. The tuple $e = \{s_k, a_k, s_{k+1}, r_{k+1}\}$ constitutes one experience. Over time, the agent accumulates experiences and uses them to learn the behavior that leads it to maximize the return over a training episode. In other words, the value function is not updated based on the latest interaction, but it is updated based on past interactions.

Research has shown (De Bruin et al., 2018) that the definition of the experience replay strategy (i.e., how big the buffer is, which experiences are added to it and how they are *replayed*) can have a significant impact on the outcome of a training run and on the convergence time.

For what concerns the buffer size, the literature suggests that retaining all of the past experiences is often a good baseline although for some problems, such as the linear magnetic manipulator used in this thesis, a limited buffer leads to better performances. For what concerns the composition and sampling of the buffer, some utility proxies, such as how *surprising*

is one experience with respect to the others already in the database or the element of exploration associated to it, can be used to give a score to each experience depending on the characteristics of the data samples that are considered to be most important.

However, in order to avoid adding elements of complexity to the algorithm design, it was chosen to keep a limited ER buffer updated with a FIFO (First-In-First-Out) rule, and to sample experience mini-batches from it uniformly at random. The results obtained in the simulated environment indicate that, at least for the mathematical model of the magman, a good policy can be obtained within a reasonable time without the need for a more sophisticated strategy.

### 4-1-4 The target network

An important part of the DQN training process is the so called *target network*.

Computing the loss between the output Q-values and the target Q-values would require two *passes* through the network parametrizing the policy, and then the function parameters would be updated accordingly. However, in this way the loss is computed with respect to a constantly moving target, which makes the learning unstable.

Therefore, the target Q-values needs to be obtained from a different network with respect to the Q network. This new network, called *target network* used to calculate the max Q-value for the system's next state, is initialized at the beginning of the learning with a copy of the weights of the Q network, and updated every once in a while. For the experiments, the target network was always updated every 10000 steps, but the frequency of update is usually considered a tunable hyperparameter of the RL model.

### 4-1-5 Algorithm summary

The following Table summarizes the algorithm used.

**Algorithm 1** DQN algorithm

Initialize ER memory buffer $\mathcal{D}$ to capacity N
Initialize the Q network (approximating the Q-value function $\mathcal{Q}$) with random weights
Initialize the target network, cloning the weights of the Q networks
**for** episode = 1,M **do**
  Initialize the system state
  **for** timestep = 1,N **do**
    Select and execute an action, depending on the exploration strategy or exploitation of the current policy
    Observe the system's next state and compute the reward associated to the transition (and add the experience to the ER database)
    Sample a mini-batch of experiences from the replay memory
    Pass the experiences batch to the Q network and calculate the loss between output Q-values and target Q-values
    Update the weights of the Q network (according to the training algorithm) to minimize loss
    Every x timesteps, clone the weights of the Q network to the target network
  **end for**
**end for**

In the magman case, empirical tests suggested that the learning process did not benefit from a deep NN parametrization of the policy, but it was instead better to parametrize the policy using only the weights of the input and output layers of the neural network, which have linear activation functions. Therefore, it would be more appropriate to call the strategy deployed a *Flat* Q-network (FQN), since DNNs are not used.

## 4-2 Model evaluation and tuning

Throughout the thesis work here presented, a considerable amount of time and effort was spent on testing different hyper-parameters configurations and exploring diverse high-level design choices. Optimizing the parameters of a learning agent is usually a computationally intensive task given the high dimensionality of the parameter vector and the time it takes to evaluate the agent's performance with every set. Finding an adequate tuning, however, is just as crucial for the success of the agent as the selection of an adequate learning algorithm itself.

For the DQN algorithm that was used during this thesis, the main hyper-parameters and high-level design choices were the following ones:

- The system control frequency, at which the learner records the system state and interacts with the environment by means of a control action.
- The sensor sampling frequency, at which the laser sensor is sampled by the data acquisition board. Normally, control frequency and sampling frequency are the same, but in the magman case, since part of the state vector is not measured directly but needs to be reconstructed, the sampling frequency was also considered a design parameter.

- The definition of a training run (i.e., initial state, target state, terminating condition)
- The number of episodes in each training run
- Definition of the system state (i.e., continuous or discretized state, resolution of the discretization, etc.)
- Definition of the control actions (i.e., continuous or discrete actions, etc.)
- Definition of the reward function
- Training algorithm (and its settings)
- Parametrization of the value function approximator (e.g., ANN and its topology, etc.)
- Training strategy (e.g. online, quasi-online, at the end of an episode, number of updates after each steps, etc.)
- Learning rate
- Discount factor
- Exploration strategy
- Definition of the experience replay buffer (size, update rule, sampling rule, mini-batch size, etc.)

The machine learning practitioner can occasionally find (scattered throughout relevant literature and online journals or forums) some directions and lessons learned.

For example, when it comes to the choice of the neural network optimizer, the literature (Kingma & Ba, 2014), (Bahar, Alkhouli, Peter, Brix, & Ney, 2017), (Basu, De, Mukherjee, & Ullah, 2018) agrees that the *adam* algorithm outperforms all of the other first-order, gradient-based optimization methods that are commonly used to train artificial neural networks, such as AdaGrad, AdaDelta, Momentum, SGT, etc.. For most of the other parameters, however, there is no *silver bullet* (a choice that has been proved to be working in the majority of applications), but the optimal value depends on the control task at hand.

Therefore, tuning and optimizing a machine learning algorithm, largely remains an empirical science. This is especially true when confronted with a problem that has not been solved with RL before, as it is the case for the magman setup.

Although a machine learning specialist can usually find a valid hyper-parameter set by trial-and-error, following the intuition developed over years of experience in the field, *manually* looking for a configuration that yields solid performances has many drawbacks. It is not a systematic approach and can be incredibly time-consuming, depending on the experience of the experimenter. The optimal values of some parameters might not be intuitive, as the performance of the learner is determined by the combination of all the parameters, whose contributions to the learning process cannot be considered and assessed independently from each other. Furthermore, keeping tracks of the many configurations tested, as the model evolves and functionalities are added, can be challenging.

For these reasons, model optimization is usually carried out in a systematic fashion deploying strategies such as grid-search, random-search or genetic algorithm-based optimization strate-

gies, see (Bergstra, Bardenet, Bengio, & Kégl, 2011), (Friedrichs & Igel, 2005) and (Sehgal, La, Louis, & Nguyen, 2019).

During this thesis, a grid-search-assisted manual optimization technique (simply referred to as grid-search) was used, chosen over more complex evolution-based optimization algorithm because of its ease of implementation. The following Section 4-3 describes the technique and its application to the magman case in some detail.

## 4-3 The grid-search method

For an algorithm with $N$ tuning parameters, a grid is defined over the N-dimensional parameter space. Every intersection corresponds to one of the configurations that will be tested during the search and for which the learning process will be evaluated, according to some pre-defined metric. Once all of the configurations have been tested, the results of each training run can be interpreted to gain more insight into how every parameter affects the learning process and on the quality of the final policy.

As an example, let us consider the case of two parameters. The discount factor $\gamma$ and the learning rate $\alpha$ of the training algorithm are two critical parameters that determine how the value function will be constructed, and updated.

The two parameter ranges are $\gamma \in [0, 1)$ and $\alpha > 0$ (the range $\alpha$ might be different for different algorithms). It is common practice to construct the grid with parameter values that are often encountered in literature and/or have been empirically found to be yielding good results on tasks that are similar to the one of interest. Following this logic, a grid was defined for values of $\gamma$ in the set $\{0.8, 0.9, 0.95, 0.97, 0.99, 0.999\}$ and values of $\alpha$ in the set $\{0.00005, 0.0001, 0.0005, 0.001, 0.005\}$.

As an evaluation metric, the average score of the final policy over a number of tests in which the system state is initialized in different initial positions evenly distributed throughout the system state space was used. Additionally, in order to account for some degree of stochasticity that is inherent to RL methods, such as the random initialization of the parameterization of the value function and the randomness of the exploration process, the scores are averaged over multiple training runs with the same hyper-parameter configuration.

The grid-search was performed to find good parameters for the Q-learning based algorithm used. Fig. 4-1 shows the grid and the scores obtained by each configuration tested.

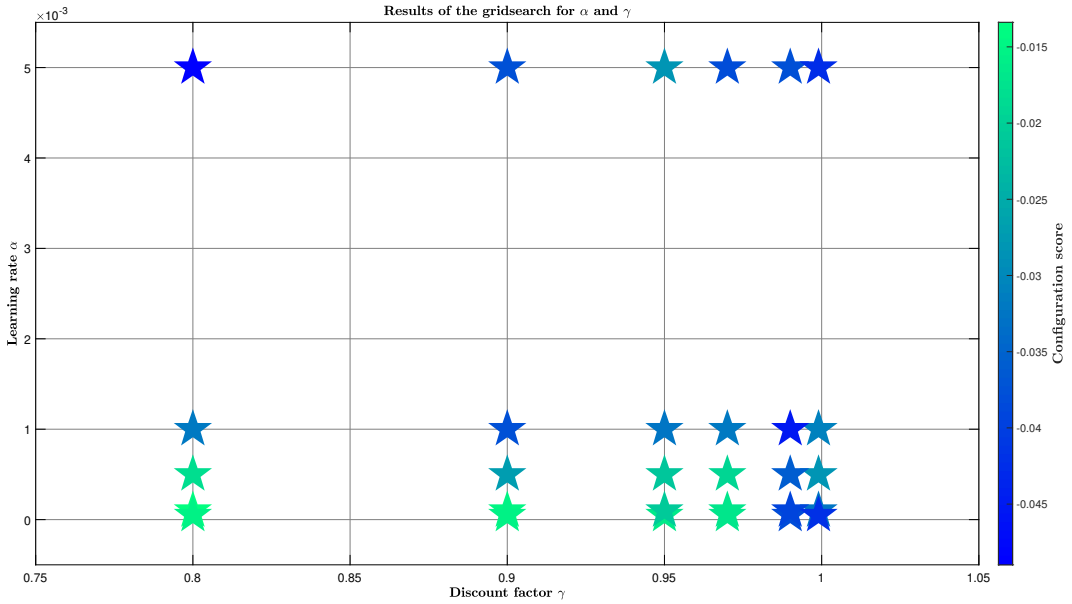



**Figure 4-1:** Scores obtained for every configuration tested by the grid-search for $\alpha$ and $\gamma$.

As it can be seen, the configuration $\gamma = 0.8$, $\alpha = 0.0001$ eventually yielded the best policy.

The same concept can be extended to the remaining algorithm hyper-parameters, but the computational time increases exponentially as more dimensions are added to the grid, and clearly such method is not feasible for use with a real-world setup. More extensive grid-searches were deployed first to determine the optimal topology of the Artificial Neural Network used to parametrize the value function, and then to optimize the remaining hyper-parameters.

### 4-3-1 Grid-search for the ANN topology

The first grid-search was carried out to determine the most suitable topology for the neural network approximating the value function in the DQN algorithm. The only hyper-parameters that were changing were the number of hidden layers and the number of neurons in each hidden layer, as illustrated in Table 4-1.

**Table 4-1:** Hyper-parameters grid search for the best ANN topology.

| Parameter | Number of hidden layers | Size of the hidden layers |
|---|---|---|
| Values | 0 | [-] |
| | 1 | [64] |
| | | [256] |
| | 2 | [64, 32] |
| | | [128, 64] |
| | 3 | [64, 64, 32] |
| | | [256, 128, 64] |
| | 4 | [64, 64, 64, 32] |

The *relu* activation function (Nair & Hinton, 2010) was used in each case. The other fixed design parameters were the following:

- Initial position picked uniformly at random in the system state space
- All four actuators in use
- Sampling rate = 50 Hz
- Number of episodes = 2000
- Episodes duration = 5 s
- Target position 0.0625 m, target velocity 0.0 m/s
- Number of RBFs used to represent the system state = 11
- Number of discrete actions for each coil = 5
- Discount factor $\gamma = 0.99$
- Learning rate $\alpha = 1 \cdot 10^{-4}$
- Initial probability of selecting a random action $\epsilon_0 = 0.45$
- Final probability of selecting a random action $\epsilon_f = \epsilon_0/100$
- ER buffer size = $1 \cdot 10^5$
- Sampled experiences mini-batch size = 128
- Linear reward function $r_{k+1} = w_x|x_e| + w_{\dot{x}}|\dot{x}_e| + \sum_{i=1}^{2} w_i|u_i|$, with weights $w_x = 1$, $w_{\dot{x}} = 0.01$, $w_1 = w_2 = 0.01$

The score assigned to each different configuration was averaged over three training runs. The outcome of this gridsearch indicated that the magman task does not benefit from a complex ANN topology, but better policies were instead found for shallow (a single hidden layer) or even flat (no hidden layer) networks. For this reason, it was decided not to pursue further deep learning, and use a flat network topology, where the network parameters are simply the weights on the linear activation functions of the input and output layers, with no hidden layer in between.

### 4-3-2 Grid-search for the remaining algorithm hyper-parameters

Following the grid-search for the optimization fo the ANN topology, other searches followed to optimize the remaining hyper-parameters. Tab. 4-2 shows the grid in which one of such searches was performed.

The following other parameters were instead constant in all of the runs:

- Initial position 0.0025 m, initial velocity 0.0 m/s
- Target position 0.0625 m, target velocity 0.0 m/s
- Sampling rate = 50 Hz
- Number of episodes = 500
- Episodes duration = 2 s
- Final probability of selecting a random action, $\epsilon_f = \epsilon_0/100$

**Table 4-2:** Grid of hyper-parameters selected to perform the grid-search.

| Parameter | Values | | |
|---|---|---|---|
| State space resolution (# of RBFs per state) | 5 | 11 | 17 |
| Number of discrete actions (per coil) | 3 | 7 | 11 |
| Discount factor, $\gamma$ | 0.9 | 0.99 | 0.999 |
| Learning rate, $\alpha$ | $1 \cdot 10^{-5}$ | $1 \cdot 10^{-4}$ | $1 \cdot 10^{-3}$ |
| Initial probability of selecting a random action, $\epsilon_0$ | 0.8 | 0.4 | 0.1 |
| Size of the experience replay buffer | $1 \cdot 10^4$ | $1 \cdot 10^9$ | |
| Mini-batch size of the replayed experiences | 64 | 128 | 256 |

- Linear reward function $r_{k+1} = w_x|x_e| + w_{\dot{x}}|\dot{x}_e| + \sum_{i=1}^{2} w_i|u_i|$, with weights $w_x = 1$, $w_{\dot{x}} = 0.01$, $w_1 = w_2 = 0.01$

The total number of configurations tested was equal to 1458, which required approximately 120 hours (wall-clock time). The following configuration proved to be the best one:

- State space resolution = 17
- Number of discrete actions per coil = 3
- Discount factor, $\gamma = 0.999$
- Learning rate, $\alpha = 1 \cdot 10^{-4}$
- Initial probability of selecting a random action, $\epsilon_0 = 0.1$
- Size of the ER buffer = $1 \cdot 10^4$
- Mini-batch size of the replayed experiences = 64

Let it be noted that this configuration was not used throughout all the tests that followed, but rather it was considered as a starting point for further fine-tuning.

The conclusions drawn from the grid-search were the following:

- Less than 70% of the 1458 combinations tested eventually led to a policy performing better with respect to a random one.
- The resolution of the state space and action space proved to be respectively positively and negatively correlated to the configuration score, in such a way that the best scores were obtained with the largest number of RBFs representing the system state and the lowest resolution of the control space.
- The characteristics of the ER buffer (i.e., mini-batch size and buffer size) did not have a significant impact on the configuration score.

## 4-4 On the choice of the reward function

The reward function is another critical design parameter that deserves some special attention.

As already mentioned in Chapter 3, the reward function is essentially a mapping from the system state and control action to a scalar value that measures how close the system is to the

target state. Throughout every episode, a reward is given to the agent after every interaction. The cumulative reward obtained at the end of every episode is an indication of the level of performance achieved by the policy.

Therefore, the reward function implicitly describes the desired agent behavior, and is a design choice of paramount importance. It essentially determines *if*, *what*, and *how* the agent eventually learns.

Reward functions can be classified in two main categories: sparse and shaped.

If the reward function is sparse, the learner is only rewarded when it achieves its target. In the magman case, the sparse reward would be received when the system state is within a box around the target state. Because the system state is continuous, the control task is accurate positioning, and the probability of the agent casually stumbling upon a sequence of actions that leads to the reward can be considered fairly low, using a *box* reward function was not considered feasible in practice. Instead, a shaped reward function was used, that continuously gives increasing feedback to the learner in an inversely proportional manner to the distance from the system desired state. In this way, the agent learns to steer the system towards the target state gradually, *following the gradient of the reward function.*

Fig. 4-2 and Fig. 4-3 show the learning curves of two identical DQN agents trained on 2000 3-second episodes (sampling frequency 50 Hz, state represented with 18RBFs, 5 discrete actions, FIFO ER buffer composed of 100000 experiences, trained in mini-batches of 128 experiences by an adam optimizer with discount factor 0.99 and learning rate 0.0001, with linearly annealed $\epsilon$-greedy exploration strategy from $\epsilon_0 = 0.6$ to $\epsilon_f = 0.0001$) with two different reward functions. As it can be seen, there is a significant difference in the two learning curves. While the agent trained with the shaped reward function successfully learns the control task, the agent trained with the sparse reward function fails to improve the agent's policy in the same number of interactions with the environment.

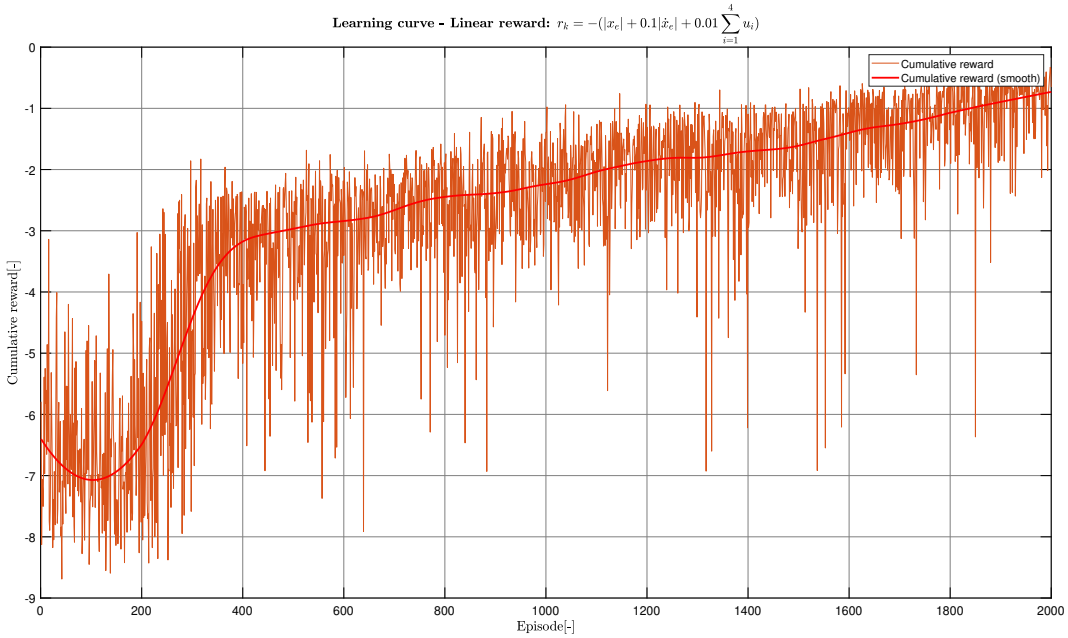



**Figure 4-2:** Learning curve of a DQN agent receiving linear reward from the environment.

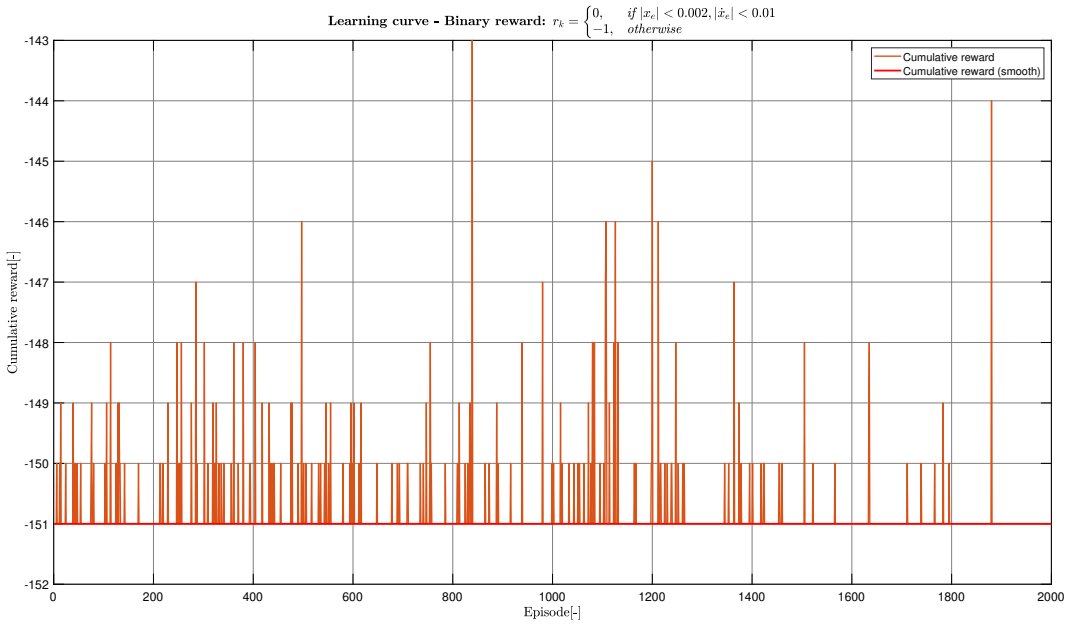



**Figure 4-3:** Learning curve of a DQN agent receiving a sparse (binary) reward from the environment.

During this thesis, different reward functions were used in an attempt to facilitate the learning task for the DQN agent agent deployed. Fig. 4-4 shows two such functions that are commonly used in reinforcement learning.

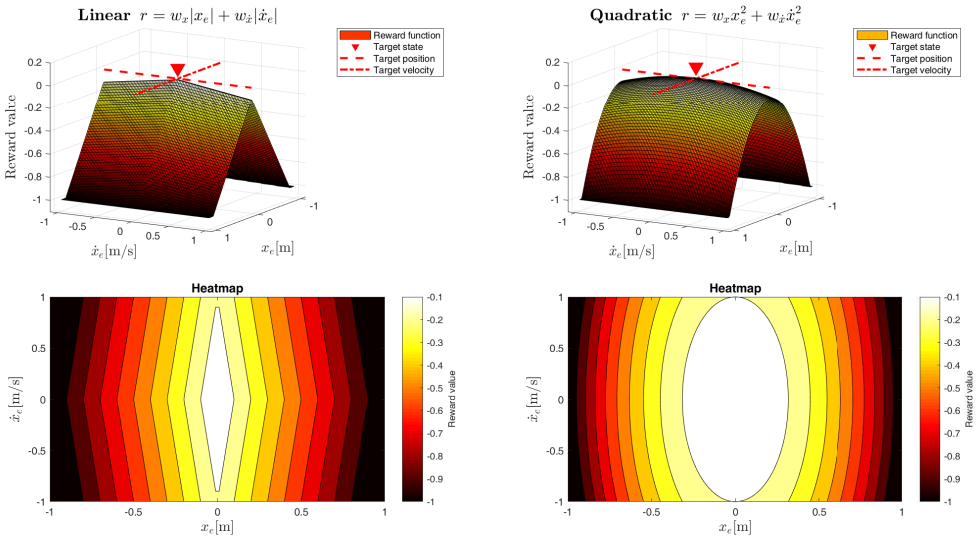



**Figure 4-4:** Two commonly used reward functions for reinforcement learning

Fig. 4-5 shows three somewhat more complex reward functions that were designed to reduce training time on the magman control task. With respect to the standard linear and quadratic reward functions, the gradients of these more complex functions get sharper as the state error gets smaller. The steepness of the gradient shall ensure that the learning agent focuses on achieving accurate positioning, and disregards any local optima that it might find in the proximity of the target state.

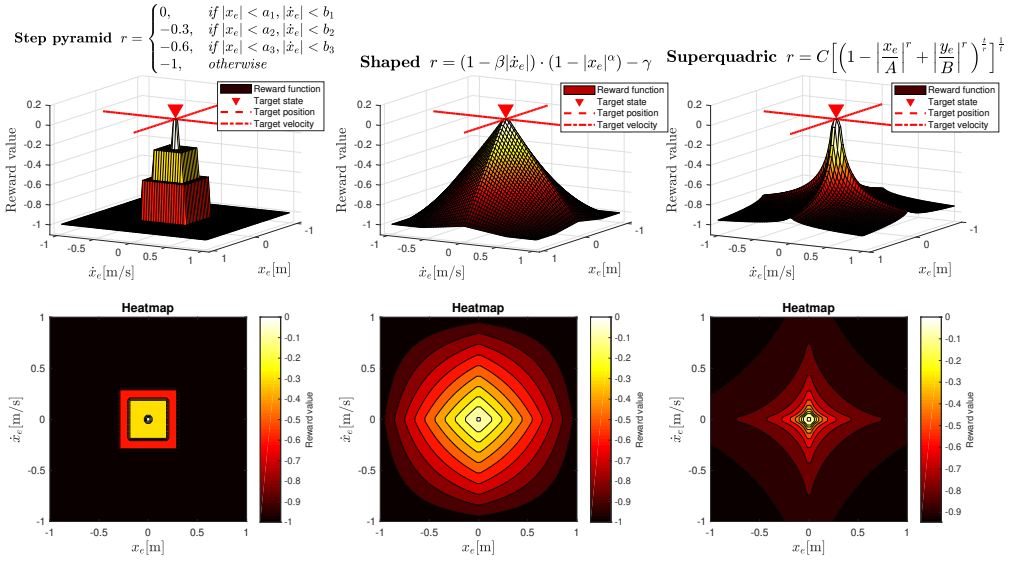



**Figure 4-5:** Some more complex reward functions designed for the magman task

The following Figures (4-6 to 4-10) show the effect of using these reward functions on the

learning process and on the performance of the final policy with different system initial conditions (only the position trajectory is shown for convenience). The agent is the same used to compare binary and linear reward.

As it can be seen, the linear and quadratic reward functions are the ones that are more efficient in allowing the agent to gradually learn a good policy within the same number of iterations. The agents trained with the other reward functions perform significantly worse.

The task at hand is accurate positioning the steel ball in an inter-coils position. Positioning the ball on top of the coils would be an easier task for the agent, since the magnetic field generated by the coil is very strong and naturally attracts the ball towards it. If the target is defined on any other point of the state space, the value associated to the coil-top locations intuitively will not be good, since there is a great chance that the steel ball gets stuck there and is eventually not able to reach the reference state. Therefore, it seemed intuitive that a suitable reward function for the magman task would give a big reward only in the close proximity of the target state, and would therefore have a relatively steep gradient to avoid giving the agent excessive reward when the ball is in a coil-top position. Therefore, the *step-pyramid*, *shaped* and *super-quadric* reward functions were designed. Although intuitively seemed to be more suitable to learn the accurate positioning task described, they actually proved to be unlearnable by the agent already at simulation stage, because the reward obtained is too sparse.

Therefore, it was decided to compute the reward for the agent according to the linear reward function (Fig. 4-6), which in simulation led to the fastest training and to the best policy. Another factor that led to this choice is that the linear reward function only has two tuning parameters. Let it be noted, however, that the other cost functions shall not be considered bad in an absolute sense, but they could still lead to meaningful results if tuned more carefully and/or if the agent is allowed for more interactions with the environment, maybe under a different exploration strategy.

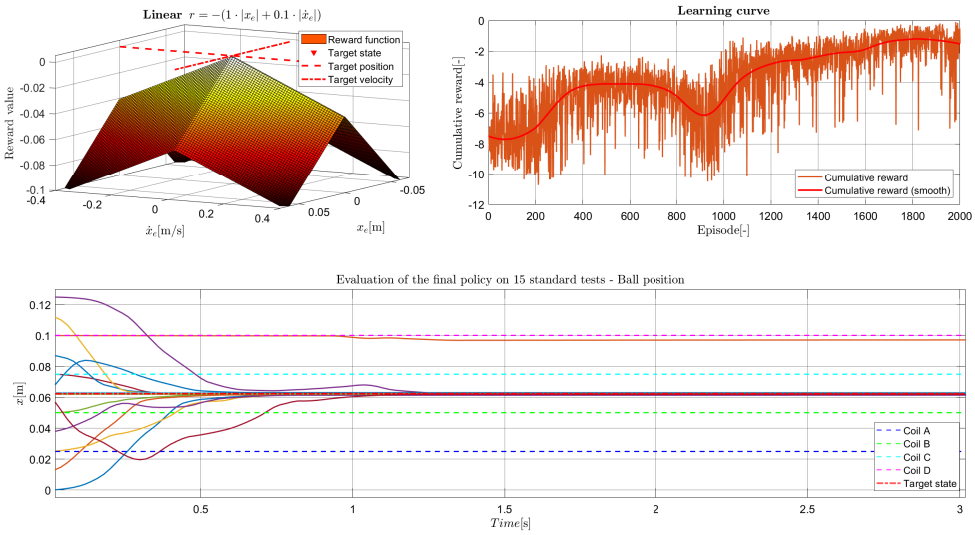



**Figure 4-6:** Effects of training an agent with a linear reward function.

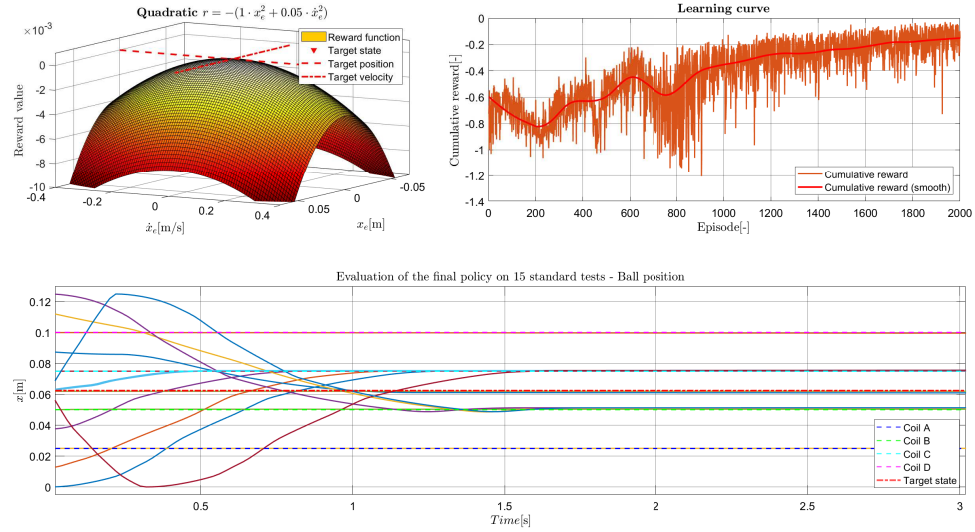



**Figure 4-7:** Effects of training an agent with a quadratic reward function.

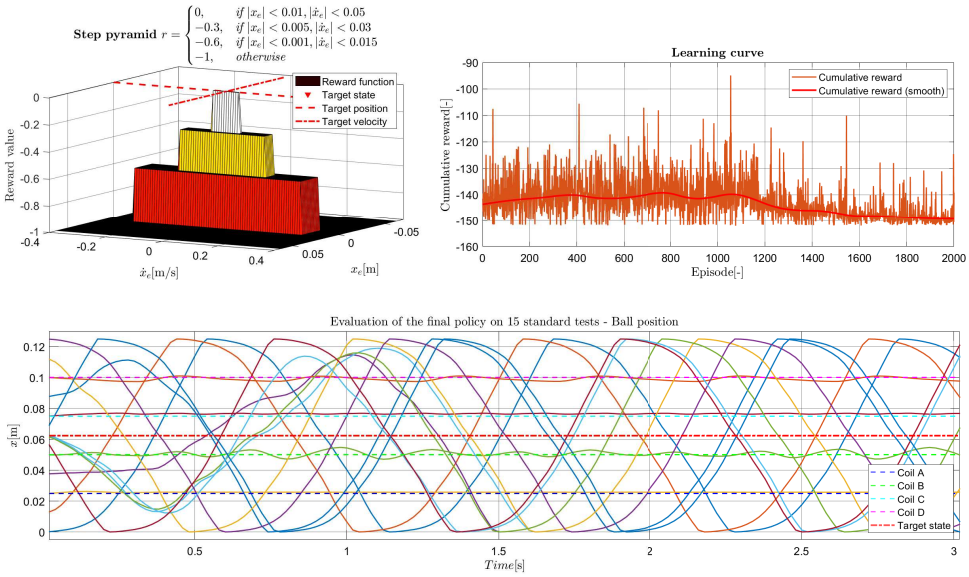



**Figure 4-8:** Effects of training an agent with a step-pyramid reward function.

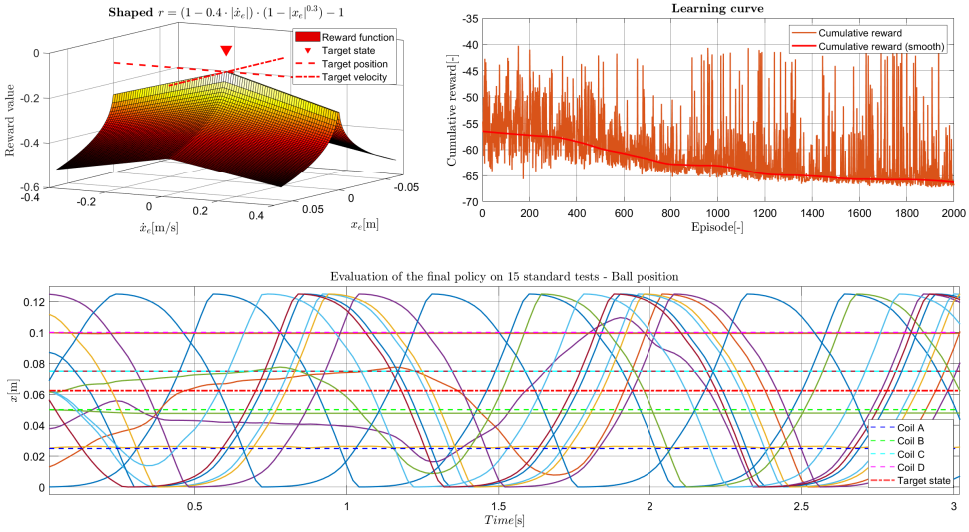



**Figure 4-9:** Effects of training an agent with another shaped reward function.

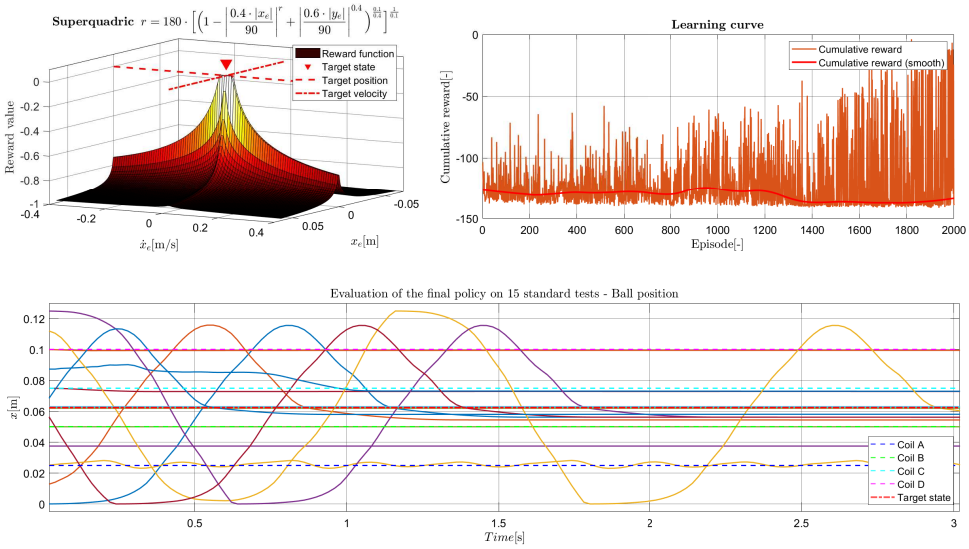



**Figure 4-10:** Effects of training an agent with a superquadric reward function.

## 4-5 Performance in simulations

The algorithm described in the previous section was optimized with the aforementioned grid-search method.

The following experiment was defined:

- All four coils (labeled as A, B, C and D) in use, positioned in $\{0.025, 0.050, 0.075, 0.100\}$ m
- Initial states picked uniformly at random in the state space ($x_0 \in [0.0, 0.125]$ m, $\dot{x}_0 \in [-0.4, 0.4]$ m/s)
- Target state $x_f = 0.0625$ m, $\dot{x}_f = 0.0$ m/s
- Sampling frequency $= 50$ Hz
- Number of episodes in the training run $= 20000$
- Training starting at episode 50 (to initialize the ER buffer with some experiences)
- Episodes duration $= 3$ s
- $\epsilon-$greedy exploration strategy. $\epsilon$ linearly annealed from 0.6 to 0.00001
- DQN algorithm trained with the *adam* optimizer (learning rate $= 0.0001$)
- Discount factor $= 0.95$
- System state represented with 11 RBFs per state
- 3 discrete control actions
- Agent trained (Q network updated) once at every time-step

- Linear reward function ($r_k = w_x|x_e| + w_{\dot{x}}|\dot{x}_e| + w_u \sum_{i=1}^{4} |u_{e_i}|$), with $w_x = 1.0$, $w_{\dot{x}} = 0.03$ and $w_u = 0.015$

As shown in Fig. 4-11, the cumulative reward obtained by the agent in every episode increases steadily throughout the training run, getting from an average starting value of $-7$ to an average final value of $-1$. As it can be seen, however, the learning curve has not plateaued around a constant value, and is still increasing. Since the final probability of choosing a random action ($\epsilon$) is relatively low, this indicates that the Q-function has not yet converged to the optimal value function, in spite of the large amount of training that the agent has received.

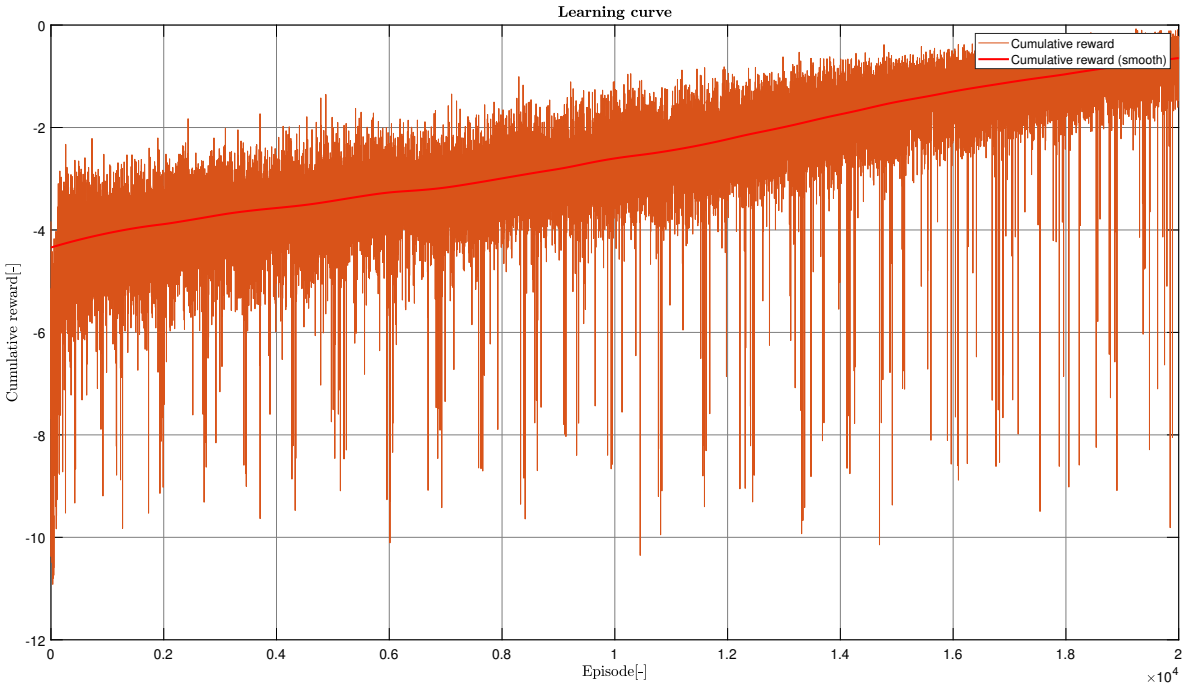



**Figure 4-11:** Cumulative reward over the 20000 training episodes.

At the end of the training, the final policy is evaluated on a number of tests where the environment is initialized in different states. As it can be seen in Fig. 4-12, the policy achieves position regulation in a consistent way across the state space, from all the initial states considered.

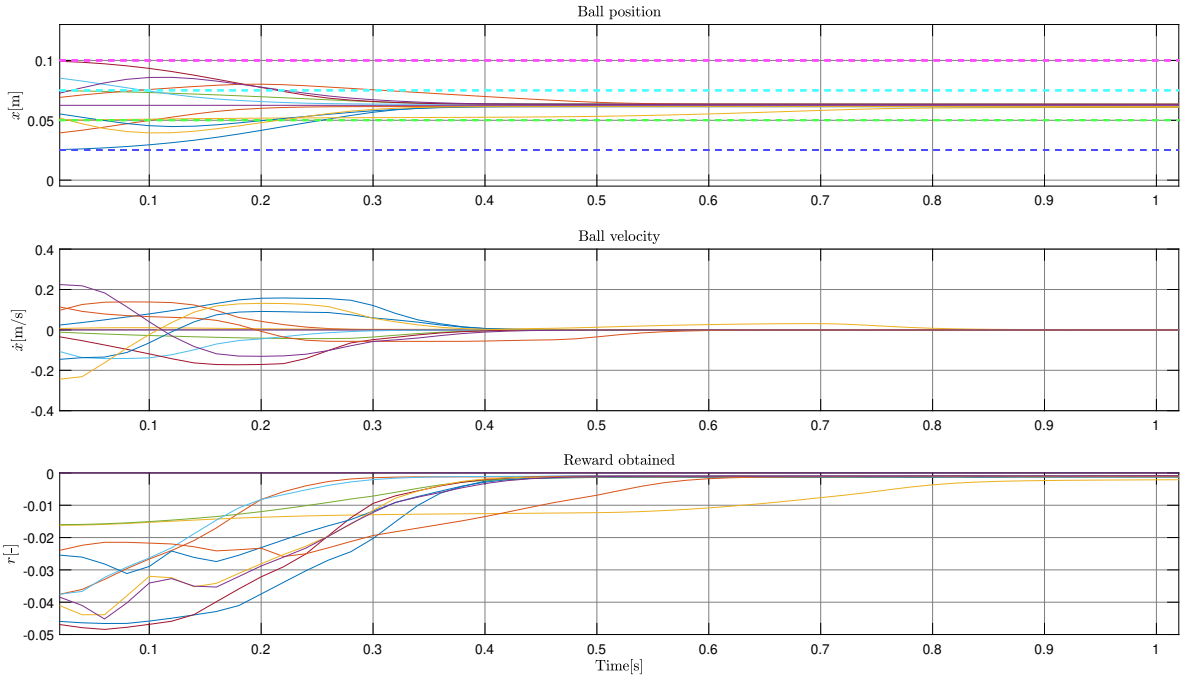



**Figure 4-12:** Performance of the final policy with the environment initialized in 15 different points chosen across the state space.

Fig.4-13 and Fig.4-14 show the state, control and reward trajectories of the first two of these tests individually.

In the first test (Fig.4-13), the system is initialized in $x = 0.0\,\text{m}$, $\dot{x} = 0.0\,\text{m/s}$. The response shows a small steady state error of $0.00127\,\text{m}$, a percentage overshoot of $0.037\%$, a rise time of $0.22\,\text{s}$ and a settling time of $0.54\,\text{s}$. As it can be seen, the agent learns what intuitively seems to be a reasonable strategy. First, it powers the two coils that are in between the initial position and the target position to move the ball towards the target. Then, it uses the second coil to decelerate the ball until it stops in the close proximity of the target, therefore switching off all of the coils.

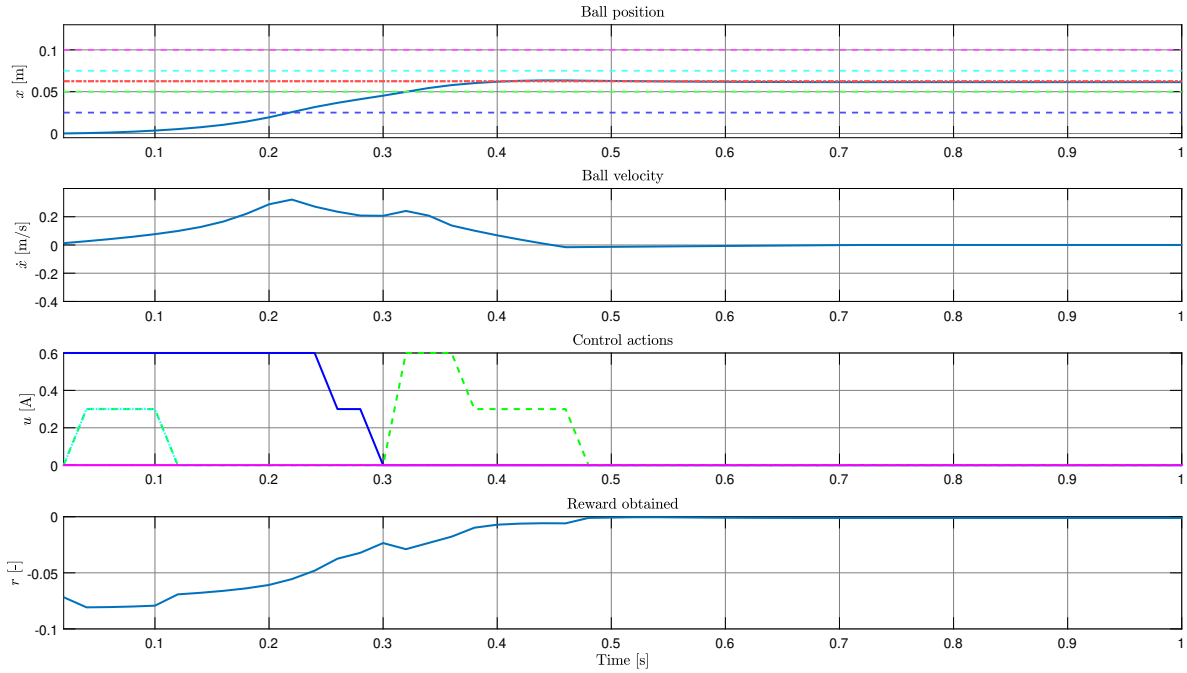



**Figure 4-13:** Performance of the final policy during an episode where the system state is initialized in $x = 0.0\,\text{m}$, $\dot{x} = 0.0\,\text{m/s}$.

The second test (Fig.4-13) is perhaps more interesting to look at, since the behavior learned by the agent is not intuitive and clearly suboptimal. The system is initialized in $x = 0.0125\,\text{m}$, $\dot{x} = 0.0\,\text{m/s}$. The response shows again a steady state error of approximately $0.00125\,\text{m}$, a negligible percentage overshoot of $0.0033\%$, a rise time of $0.2\,\text{s}$ and a settling time of $0.58\,\text{s}$. What is interesting is that the agent does not power on the first coil immediately, but instead powers on the second coil with half of the maximum available current, and then uses the second and the fourth coils to stop the ball on the reference position, before turning off all of the coils.

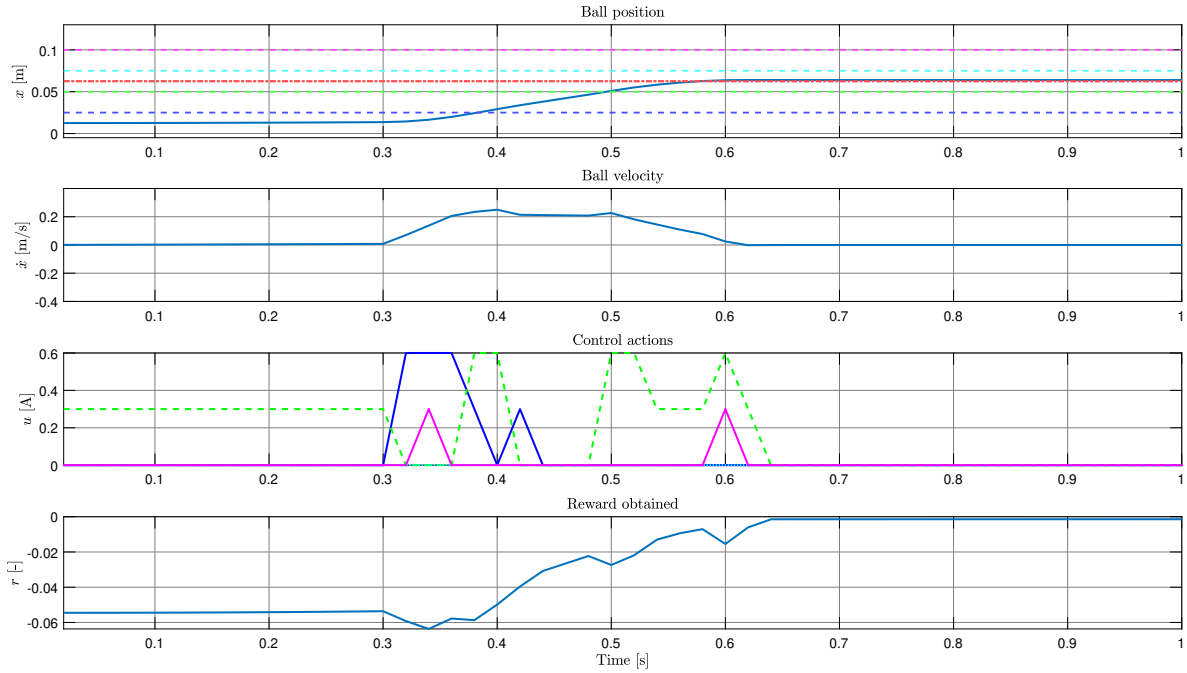



**Figure 4-14:** Performance of the final policy during an episode where the system state is initialized in $x = 0.0125\,\text{m}$ (on top of the first coil), $\dot{x} = 0.0\,\text{m/s}$.

Fig. 4-15 shows the final policy for each of the four coils. As it can be seen, the policy is not exactly coherent, in spite of the fact that the agent seems to be doing what it is supposed to, its approximation of the value function is clearly not optimal. Such shape of the control policy is due to the choice to represent the system state as RBFs.

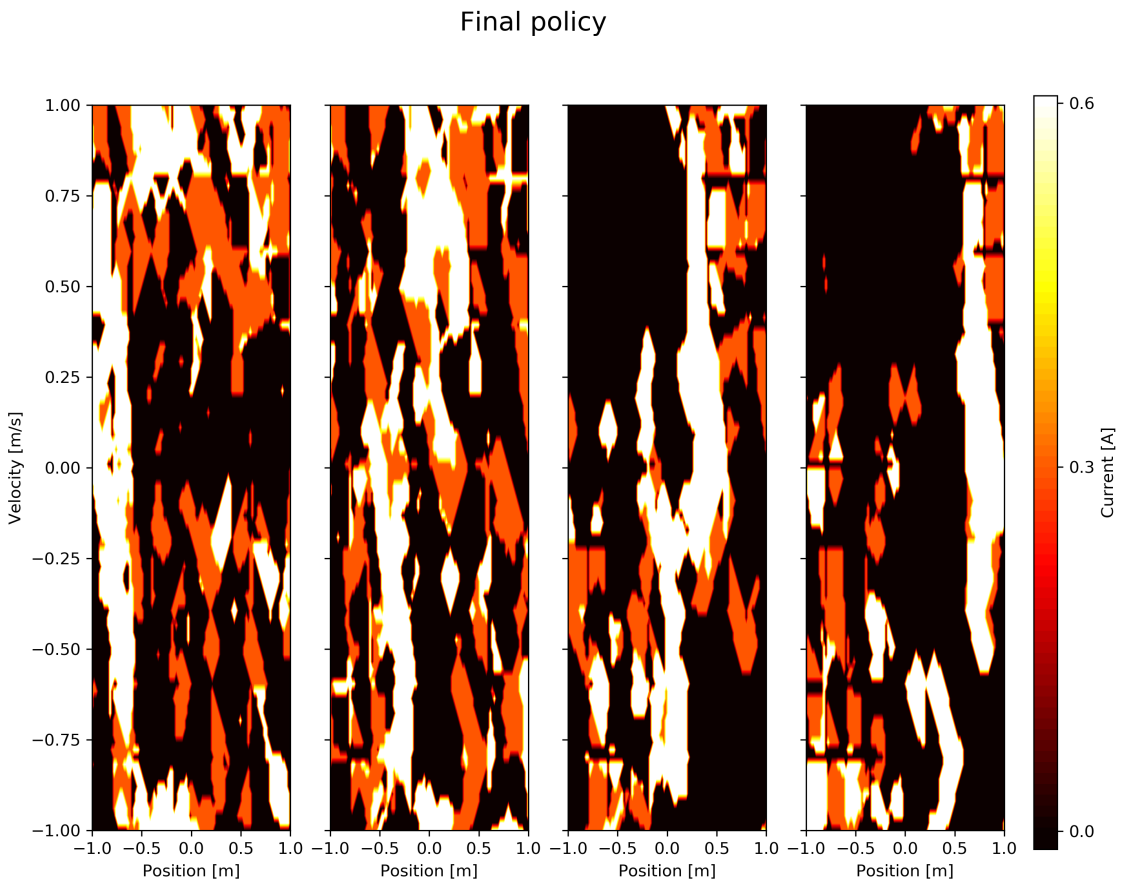



**Figure 4-15:** Final policy learned by the agent for the four actuators (from coils A to D from right to left).

## 4-6 Performance comparison to other controllers

At this point it was considered interesting to compare the performance of the controller achieved so far to the performance of two other controllers designed for the magman setup and described in (J. Damsteeg, 2015).

In order to allow for a comparison, the experiment was redefined and the agent presented in the previous section was retrained from scratches with:

- Initial state $x_0 = 0.075\,\text{m}$, $\dot{x}_0 = 0\,\text{m/s}$ (steel ball at rest top of the third coil) in all of the episodes
- Target state $x_f = 0.025\,\text{m}$, $\dot{x}_f = 0.0\,\text{m/s}$ (steel ball on top of the first coil with zero velocity)

The system model on which the agent was trained was also changed back to the legacy model used by Damsteeg (2-2).

The RL controller is compared to:

- **A nonlinear controller based on Constrained State-Dependent Riccati Equations (C-SDRE)**
- **An imitation learning controller based on Local Linear Regression (LLR)**

The comparison was carried out on the basis of the following characteristics of the closed loop response:

- **Rise time**, defined as the time necessary for the ball position to raise from 10% to 90% of the steady state value.
- **Settling time**, defined as the time when the position error becomes less than 2% of the steady state value (the target state).
- **Overshoot**, defined as the percentage overshoot relative to the steady state ball position.

The agent was trained for 10000 episodes of the duration of $50Hz$ each. Fig. 4-16 shows the learning curve of the training run.

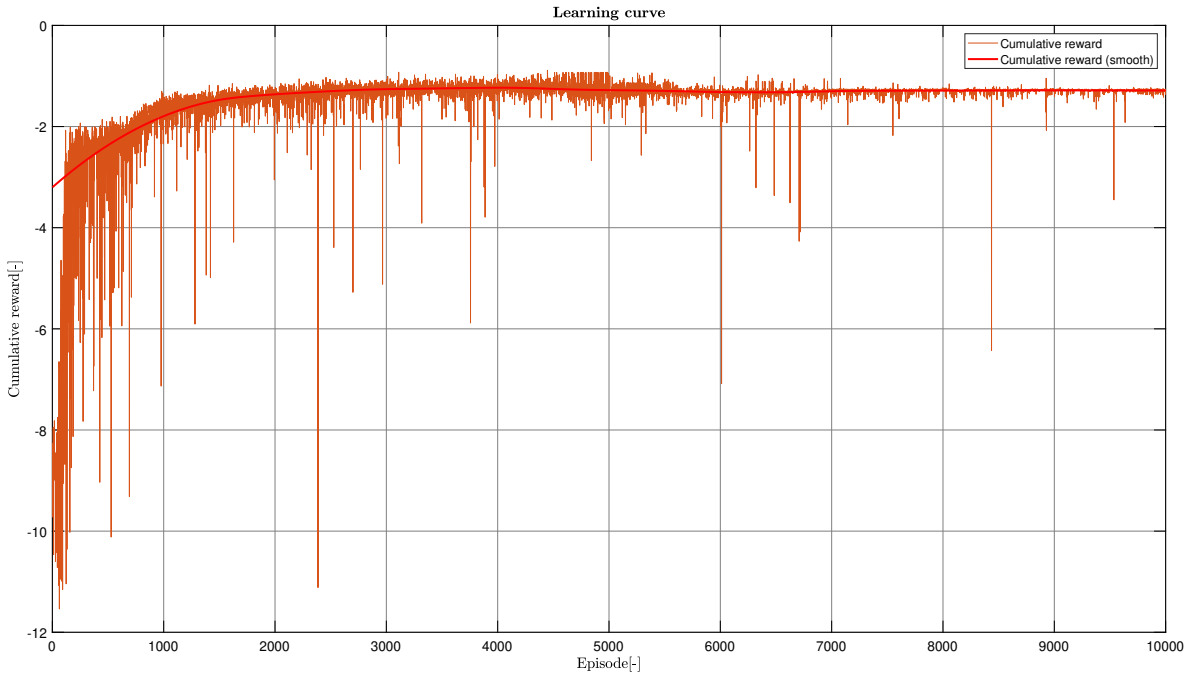



**Figure 4-16:** Learning curve of the agent trained with reference state for the steel ball position of 0.025 m.

Fig. 4-17 shows the step-response in the time domain of the policy learned by the agent.

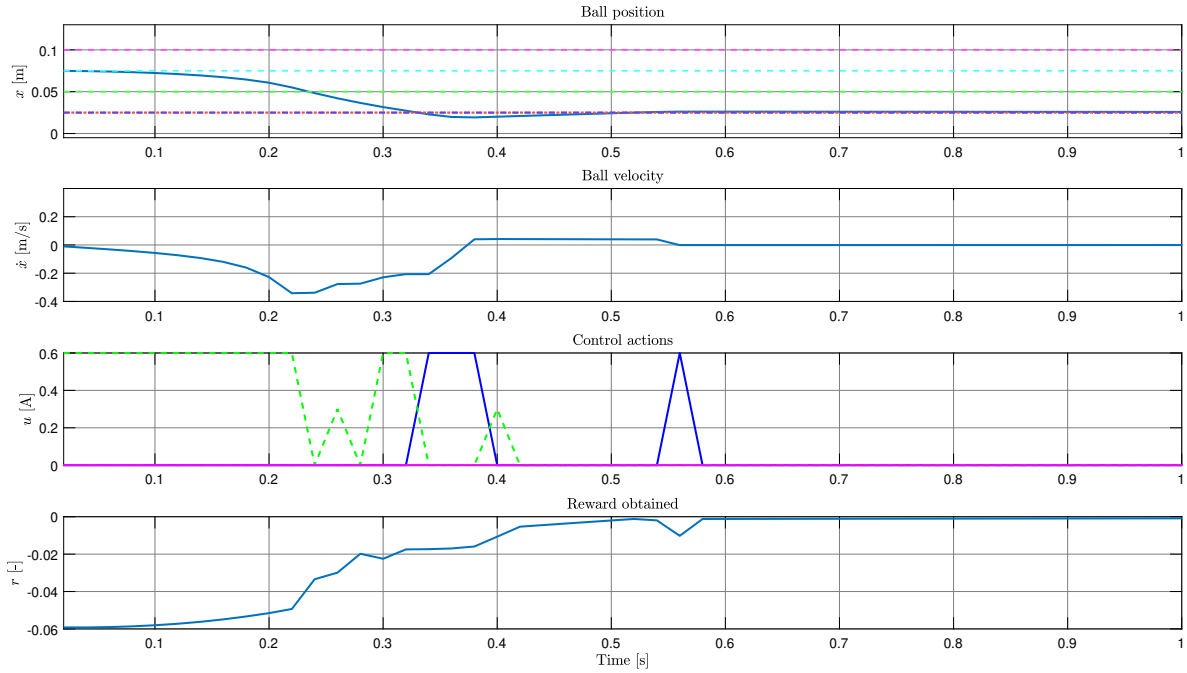



**Figure 4-17:** Policy learned by the agent for a target state of $0.025\,\text{m}$ and an initial state of $0.075$ m

Table 4-3 compares the performance (in the simulated environment) of the RL agent with the performance of the two controllers designed by Damsteeg. As it can be seen, the RL agent

**Table 4-3:** Performance comparision of the RL controller with two traditional controllers.

| **Controller:** | **Q-learning** | **C-SDRE** | **LLR** |
|---|---|---|---|
| **Rise time [s]** | 0.28 | 0.32 | 0.33 |
| **Settling time [s]** | 1.66 | 0.98 | 0.65 |
| **Overshoot [%]** | 10.6 | 0 | 3.2 |

performs worse compared to the two other controllers, with a significant overshoot of around 10% and a longer settling time.

## 4-7 Performance on the experimental setup

The same agent described in Sec. 4-5 was deployed on the experimental setup, but the observed behavior was quite different with respect to what had been observed in the simulation. In fact, the agent deployed, even after a fairly large number of interactions with the environment, would not show any improvement in its behavior.

Therefore, the reinforcement learning task was scaled down to a simpler problem in order to try to understand better where the bad performance of the agent were stemming from.

- The ball position was constrained to a subset of the original space, from $[0.0, 0.125]$ m to $[0.025, 0.100]$ m.

- Only the two middle actuators (coil B and coil C) were used for control.
- At the beginning of each episode, the steel ball was initialized at the beginning of the reduced-length rail, on top of the first coil with zero velocity, rather than in a random position of the state space, as in the case of the agent trained in the simulated environment.

Initially, it was assumed that the poor agent's performance could be due to a poor tuning of the algorithm hyper-parameters. The literature on the topic indicates that RL algorithms in general are quite sensitive to changes in the hyper-parameters. Therefore, it was considered that the parameters setting that had been found for the simulated environment could not be optimal also for the experimental setup, because of the model-plant mismatches.

Therefore, after a conspicuous time spent on fine-tuning the hyper-parameters, the following experiment was defined:

- Only the two middle coils (labeled as B and C) in use, positioned in $\{0.050, 0.075\}\,\mathrm{m}$
- System state initialized on top of the first coil with zero velocity ($x_0 = 0.025\,\mathrm{m}$, $\dot{x}_0 = 0.0\,\mathrm{m/s}$).
- Target state $x_f = 0.0625\,\mathrm{m}$, $\dot{x}_f = 0.0\,\mathrm{m/s}$
- Sampling frequency $= 50\,\mathrm{Hz}$
- Number of episodes in the training run $= 3000$
- Episode duration $= 3\,\mathrm{s}$
- $\epsilon-$greedy exploration strategy. $\epsilon$ asymptotically annealed from $\epsilon_0 = 0.6$ to $\epsilon_f = 0.0005$
- DQN algorithm trained with the *adam* optimizer (learning rate $= 0.0001$)
- Discount factor $= 0.95$
- System state represented with 5 RBFs per state
- 13 discrete control actions
- Agent trained (Q-network updated) four times at every time-step
- Linear reward function ($r_{k+1} = w_x|x_e k+1| + w_{\dot{x}}|\dot{x}_e k+1| + w_u \sum_{i=1}^{2} |u_{e_i}|$), with $w_x = 1.0$, $w_{\dot{x}} = 0.1$ and $w_u = 0.01$

With this definition of the high-level experiment and with these hyper-parameters, the agent showed an interesting behavior. As it can be seen from Fig. 4-18, the cumulative reward obtained by the agent during the training run increases steadily from an average of $-5$ to an average of $-2$.

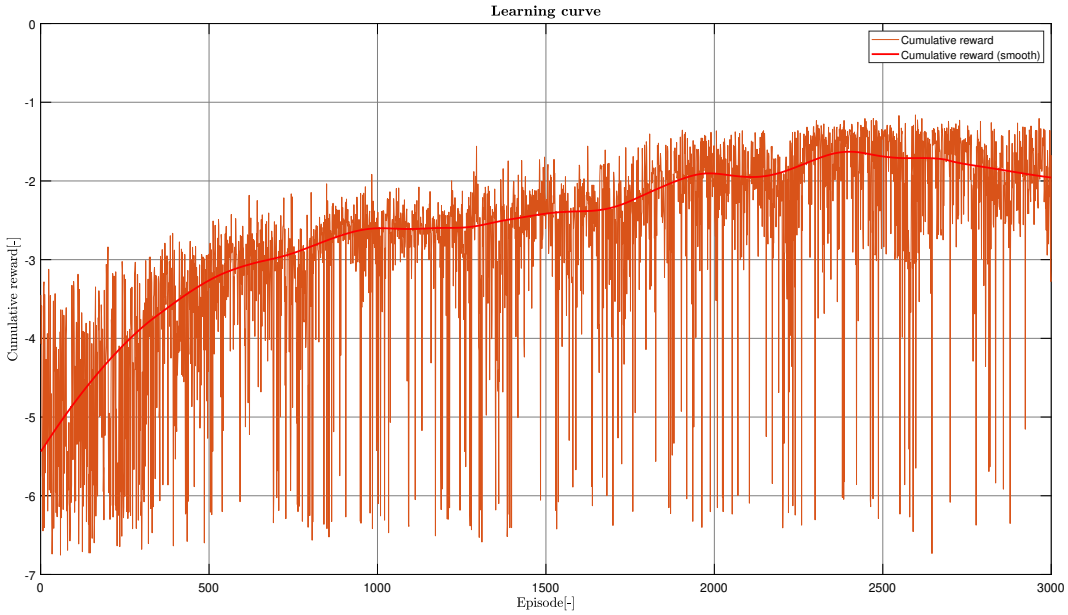



**Figure 4-18:** Learning curve of an agent trained online on data extracted from the experimental setup in a scaled-down version of the experiment.

Fig. 4-19 shows the distribution of the final states in the system state space. As it can be seen, they are essentially distributed in the close proximity of the target state, but still not close enough.

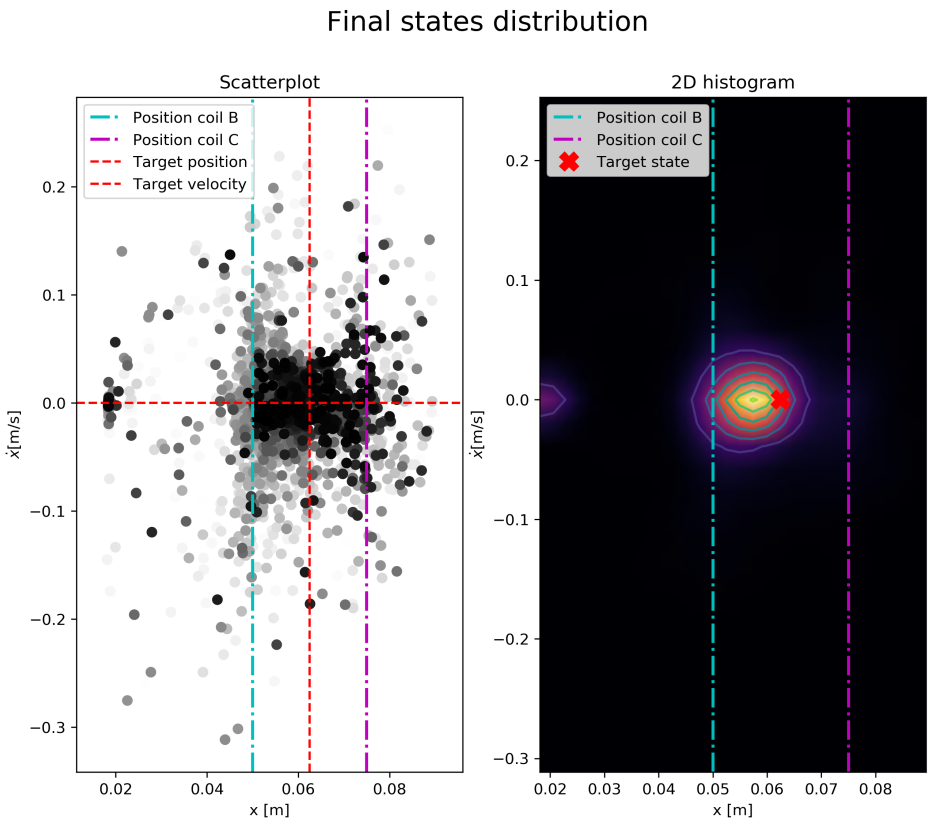



**Figure 4-19:** Distribution of the final states at the end of each episode in the training run. The darker dots indicate the final state towards the end of the training run.

Fig. 4-20 shows the performance of the policy in the last 10 episodes, when the probability of selecting a random action is the smallest and the agent is essentially following the policy learned by the agent. As it can be seen, the agent successfully actuates the two coils to bring the ball towards the reference position and does not allow it to overshoot beyond the position of the second coil, slowing it down to zero velocity. Hovewer, it does not really manage to achieve accurate positioning.

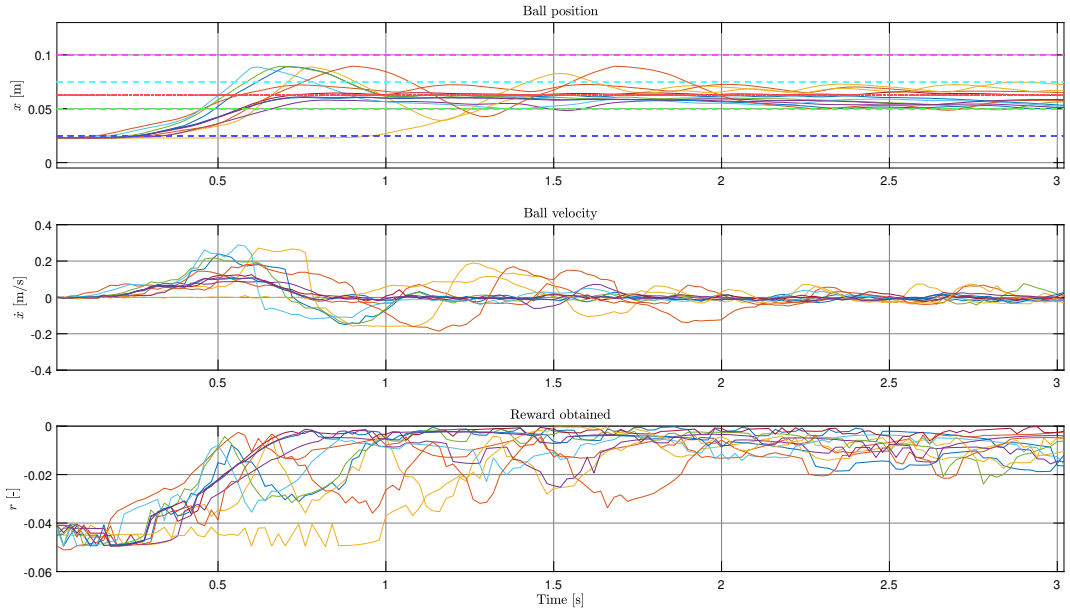



**Figure 4-20:** Time-domain plot of the last 10 episodes of the training run, when the exploration probability has died out.

Fig. 4-21 shows the policy learned by the agent at the end of the training.

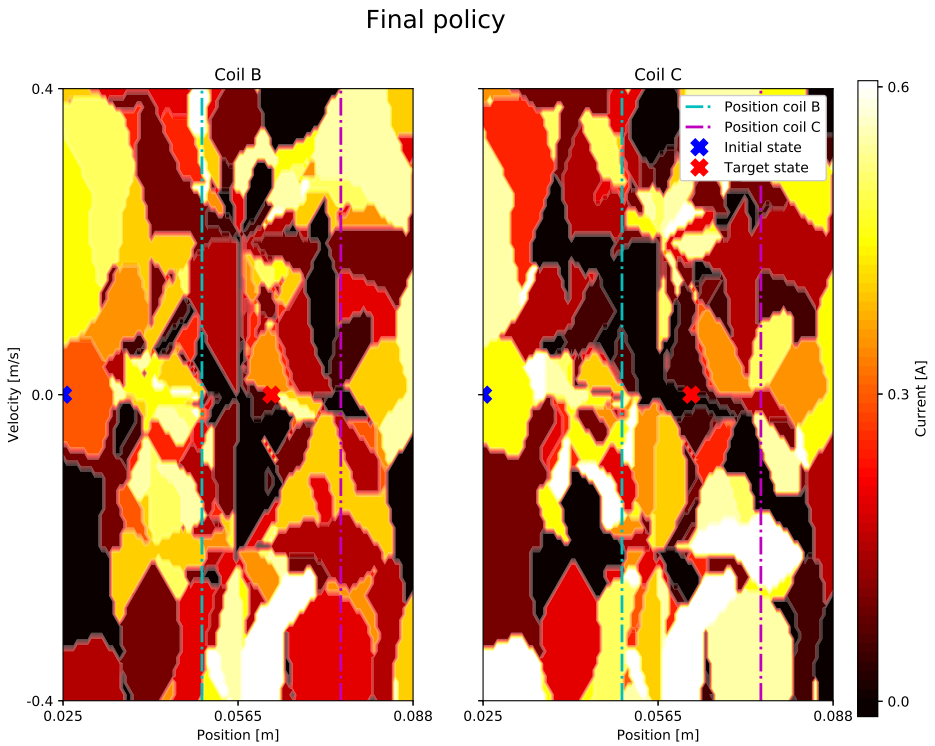



**Figure 4-21:** Final policy learned by the agent.

More effort and time was therefore put into fine-tuning the hyper-parameters in an attempt

to improve the performance of the agent, and particularly the reward function was modified in such a way to have a steeper gradient towards the target state, in such a way to encourage the desired behavior, but the performance of the agent got worse rather than improving.

## 4-8 Q-learning - Second implementation

Another controller based on Q-learning was implemented from scratch with only the essential components of the algorithm, without relying on *grey-box* models based on the Tensorflow and Keras libraries. The algorithm is the following.

**Algorithm 2** DQN algorithm - second implementation

1: Initialize ER memory buffer $\mathcal{D}$ to capacity D
2: Initialize the parameters of the Q-network, $\Theta(s, a)$ pessimistically
3: **for** episode = 1, $N_e$ **do**
4: Initialize the system state
5: **for** timestep = 1,K **do**
6: Select and execute an action, depending on the exploration strategy or exploitation of the current policy
7: Observe the system's next state and compute the reward associated to the transition (and add the experience to the ER database)
8: Sample a single experience from the replay memory
9: Use the sampled experience to update the Q-network parameters according to the Bellman equation, with $a$ the action that led to the state transition, $s$ the initial state, $s'$ the new state, $r(s', a)$ the reward associated to the transition, $\gamma$ the discount factor and $\alpha$ the learning rate
$Q(s, a) \longleftarrow Q(s, a) + \alpha\phi(s)\left[r(s', a) + \gamma \max_{a'}(Q(s', a')) - Q(s, a)\right]$
10: **end for**
11: **end for**

The system continuous state is represented by radial basis functions fuzzy approximators (Buşoniu, Ernst, De Schutter, & Babuška, 2005) with $N$ cores along each state-dimension, in such a way that the 2-dimensional state will be represented by a $2Nx1$ vector containing the membership degrees.

The Q-network is therefore parametrized by a matrix $\Theta$ having dimension $2NxM$, where $M$ is the number of possible discrete actions.

The following experiment was therefore defined:

- Only the two middle coils (labeled as B and C) in use, positioned in $\{0.050, 0.075\}$ m
- System state initialized on top of the first coil with zero velocity ($x_0 = 0.025$ m, $\dot{x}_0 = 0.0$ m/s).
- Target state $x_f = 0.0625$ m, $\dot{x}_f = 0.0$ m/s
- Sampling frequency = 50 Hz
- Number of episodes in the training run = 3000

- Episodes duration = 3 s
- $\epsilon-$greedy exploration strategy. $\epsilon$ linearly annealed from $\epsilon_0 = 0.9$ to $\epsilon_f = 0.1$
- Learning rate $\alpha = 0.1$
- Discount factor $\gamma = 0.95$
- System state represented with 17 RBFs per state
- 3 discrete control actions for each coil (9 in total)
- Linear reward function ($r_{k+1} = w_x|x_e k+1| + w_{\dot{x}}|\dot{x}_e k+1| + w_u \sum_{i=1}^{2} |u_{e_i}|$), with $w_x = 100$, $w_{\dot{x}} = 0.0$ and $w_u = 0.1$

Fig. 4-22 shows the learning curve of the training run. Clearly, the agent does not show signs of improvement and does not converge to a stabilizing policy. It is however more interesting to analyze the evolution of the values of the Q-network parametrization throughout the training run, shown in Fig. 4-23. Approximately 2400 out of the 3600 parameters, in fact, do not change much from their initial values, in spite of the relatively large number of training episodes. This suggests that this algorithm implementation is very sample-inefficient, and a much longer training time is necessary for the value function and the policy to converge.

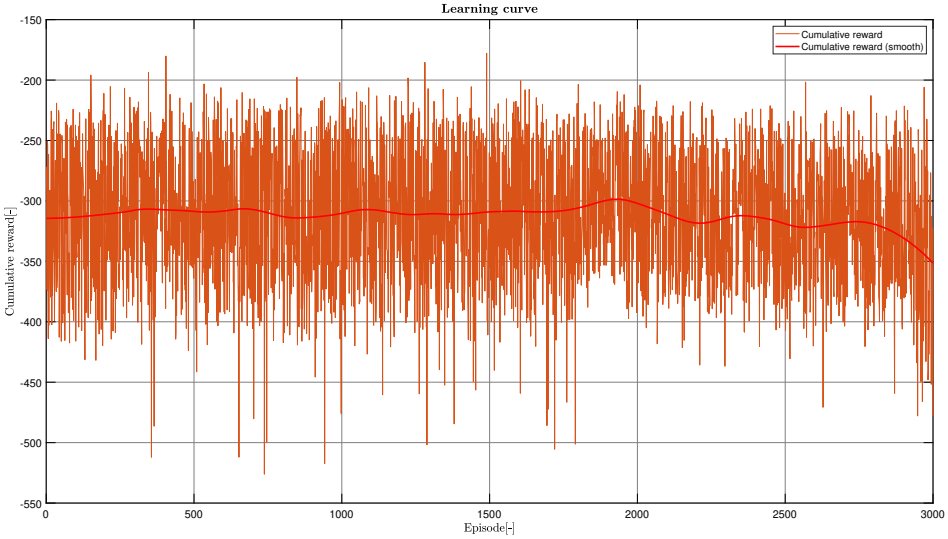



**Figure 4-22:** Learning curve of an agent based on a simplified implementation of the Q-learning algorithm.

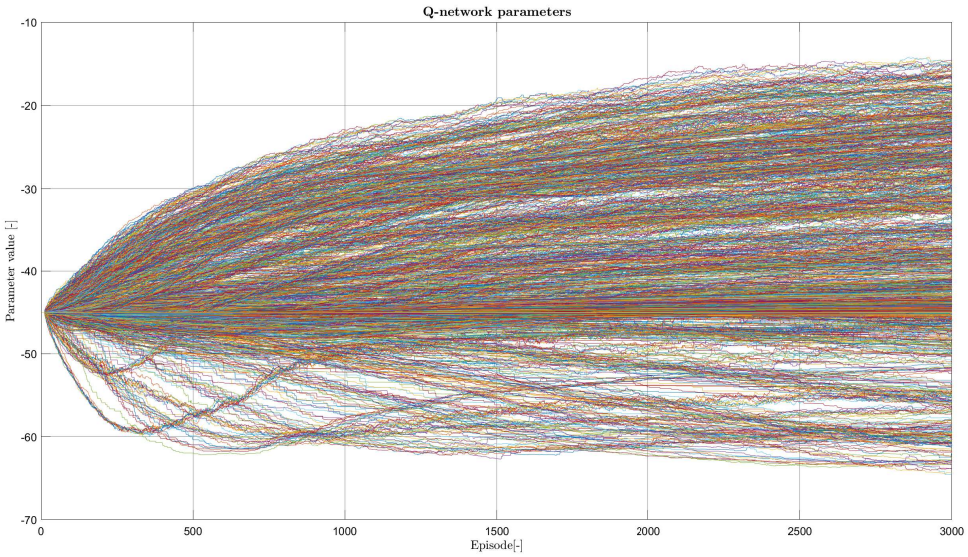



**Figure 4-23:** Evolution of the Q-function parameters during a training run.

## 4-9 Discussion of the results

Eventually, it became clear that the chosen approach would not be rewarding, and that the issue with the learning process had little to do with the control algorithm of choice or the inability to find the right tuning of its parameter, since the experimental setup was unjustifiably exhibiting a radically different behavior with respect to the simulations.

Therefore, it was considered necessary to review the assumptions made about the experimental part of the thesis, focusing on the differences between the system model and the setup.

### 4-9-1 Control input transportation delay

A first assumption was the lack of transportation delay. It was therefore considered that the control input commanded by the algorithm at any given timestep would cause a current running in the magman coils at the next discrete time-step.

In other reports of tests and studies carried out on the magman setup there is no record of a significant transportation delay. For this project, however, it was necessary to design a new I/O interface, in order to have all the necessary software within a single Python IDE, and there was the possibility that the novel interface would be handling poorly the communication between the computer and the control boards, introducing unwanted delays in the control signals.

Therefore, a test was made where the voltage on the cable delivering current to the first coil was connected to the data acquisition board (with a potentiometer in order to avoid saturating the board), logged and compared to the step signal used as control input for the coil.

From the test, it was concluded that the control input does not suffer from any relevant transportation delay, for the control frequency of 50 Hz used during the experiments with the RL algorithm (Fig. 4-24).

The dynamics of the transportation delays for the remaining three actuators were assumed to be similar to the ones of the tested coil, and therefore not tested specifically.

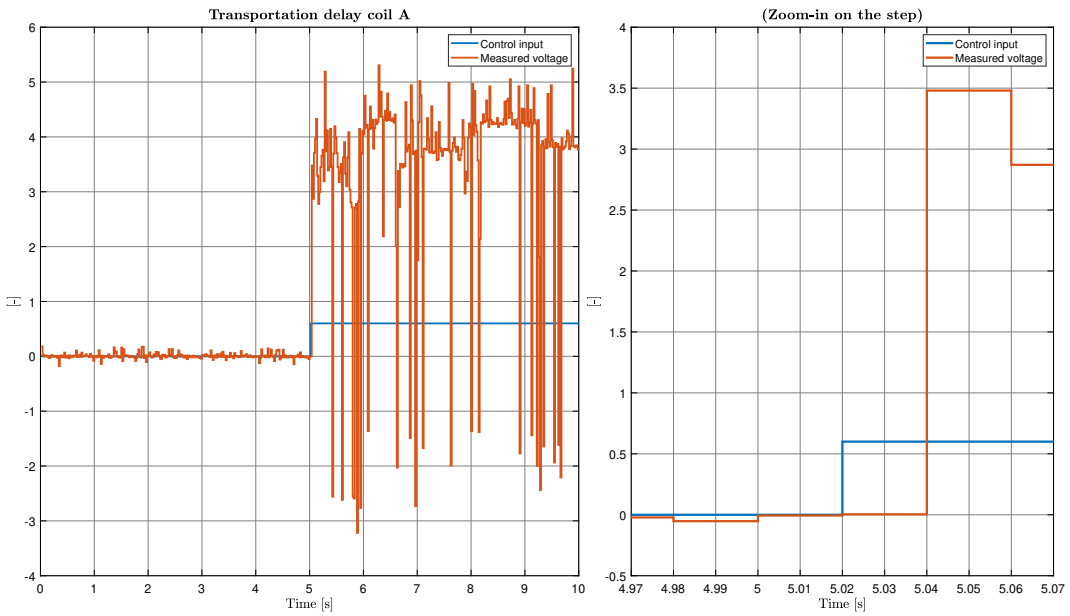



**Figure 4-24:** Experiments showed that the control input does not suffer from a significant transportation delay.

## 4-9-2 Markov Property

The second important assumption was that the Reinforcement Learning task at hand is a Markov Decision Process. The magman setup itself can reasonably be considered to satisfy the *Markov Property*, as the dynamics of motion of the steel ball under the influence of the magnetic field are prominently causal and deterministic, although highly nonlinear and impossible to describe analytically. Clearly, as for all physical phenomena, there are some elements of stochasticity, such could be the constant change in the Earth magnetic field, frequency fluctuations in the power-grid to which the power supply powering the coils is connected, variations in the temperature of the room in which the experiments are carried out that affects some physical properties of the electromagnets, etc., but the contribution of all these processes to the system dynamics can be considered negligible.

However, this consideration does not imply that the RL task is itself a *Markov Decision Process*. With the current implementation of the setup, in fact, it is only possible to measure the steel ball position, but the velocity needs to be reconstructed. Therefore, the learner *sees* a state vector that is partially representing the actual state of the system and partially reconstructed from noisy measurements of past input states. One-step dynamics are not sufficient to determine the evolution of the system. Strictly speaking, achieving regulation control with the magman setup cannot be considered a *Markov decision process*, given the

current setup and task definition.

Although this consideration had already been made before the beginning of the experimental phase, it was not expected to affect the learning process dramatically, as the reconstructed state was expected to be close enough to the actual state of the system not to jeopardize the whole learning process. Experiments carried out in the simulation environment showed quite the opposite.

The simulated environment was therefore modified to gain an understanding of what would be the consequences of discarding the MDP assumption. The following experiment was defined:

- Coils B and C in use (positioned in $\{0.050, 0.075\}$ m)
- Initial states picked uniformly at random in the state space ($x_0 \in [0.0, 0.125]$ m, $\dot{x}_0 \in [-0.4, 0.4]$ m/s)
- Target state = 0.0625 m, 0.0 m/s
- Sampling frequency = 50 Hz
- Number of episodes in the training run = 2000
- Episodes duration = 3 s
- $\epsilon-$greedy exploration strategy. $\epsilon$ linearly annealed from 0.6 to 0.001
- DQN algorithm trained with the *adam* optimizer (learning rate = 0.0001)
- Discount factor = 0.95
- System state represented by RBFs (11 for each state)
- 6 discrete control actions
- Value function updates per step = 2
- Linear reward function ($r_k = w_x|x_e| + w_{\dot{x}}|\dot{x}_e| + w_u|u_e|$), with $w_x = 1$, $w_{\dot{x}} = 0.05$ and $w_u = 0.01$

Two different versions of the experiment were carried out. In the first version, the state seen by the RL agent was the *true* state of the environment, solution of the differential equations of the system model. In the second experiment, the first state seen by the RL agent was the *true* ball position, and the second state was the velocity reconstructed from the past measurements, as it is the case for the experimental setup. The measurement noise affecting the actual system was not modeled.

In the first case, the agent is able to continuously improve the cumulative reward obtained over every episode on average, eventually achieving regulation control and a cumulative reward per episode around $-1$ (Fig. 4-25). The fact that the trend is still increasing indicates that the algorithm has not converged yet and it requires more training.

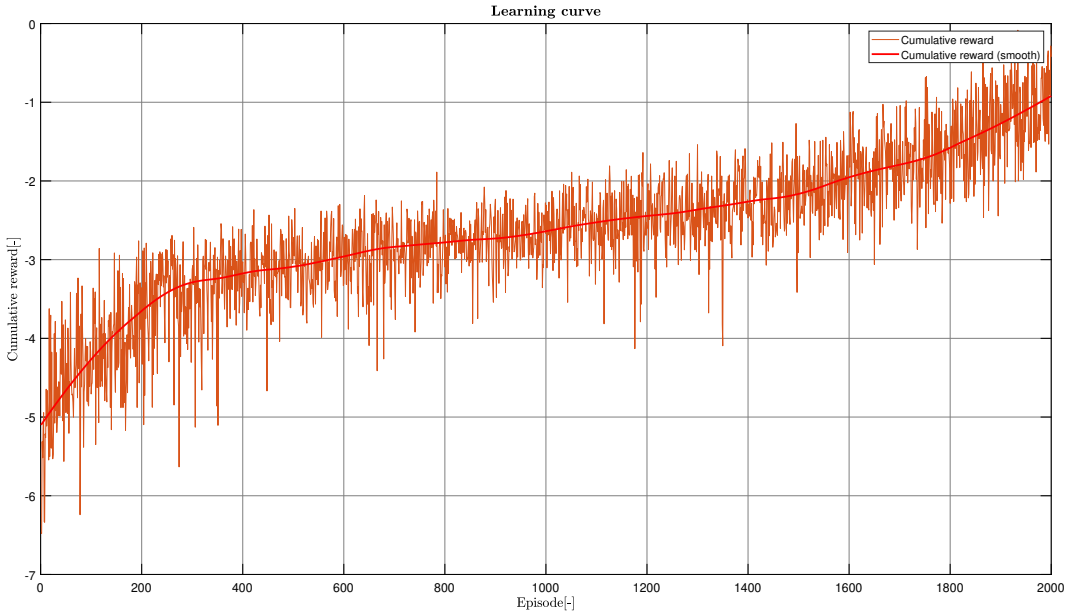



**Figure 4-25:** Learning curve of an agent that sees the true system state.

In the second case, the exact same agent is simply unable to learn the task from the data it is presented with (Fig. 4-26), although the learning curve is increasing with a very mild slope, towards the end of the training, average cumulative reward per episode is around $-4$. It is noteworthy that the performance of this agent is far worse with respect to the performance of the first agent, but still better with respect to the average performance of an agent acting following a random policy (considered as a baseline for comparison).

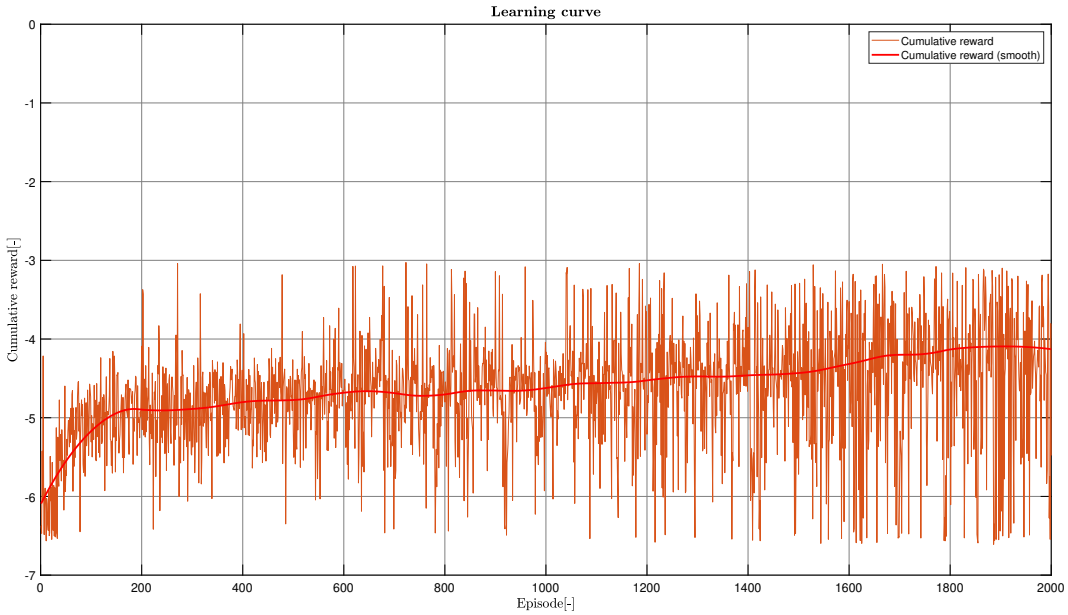



**Figure 4-26:** Learning curve of an agent that sees the partially reconstructed system state.

Another test showed that the agent does not improve its performance on the task even when

it receives much more extensive training (50000 episodes).

In order to understand why this is such an issue for the learner, it is useful to considered the data logged during a randomly chosen episode from the second training run (with the partially observed state vector). During this training run, also the *true* evolution of the system state was logged to allow a comparison.

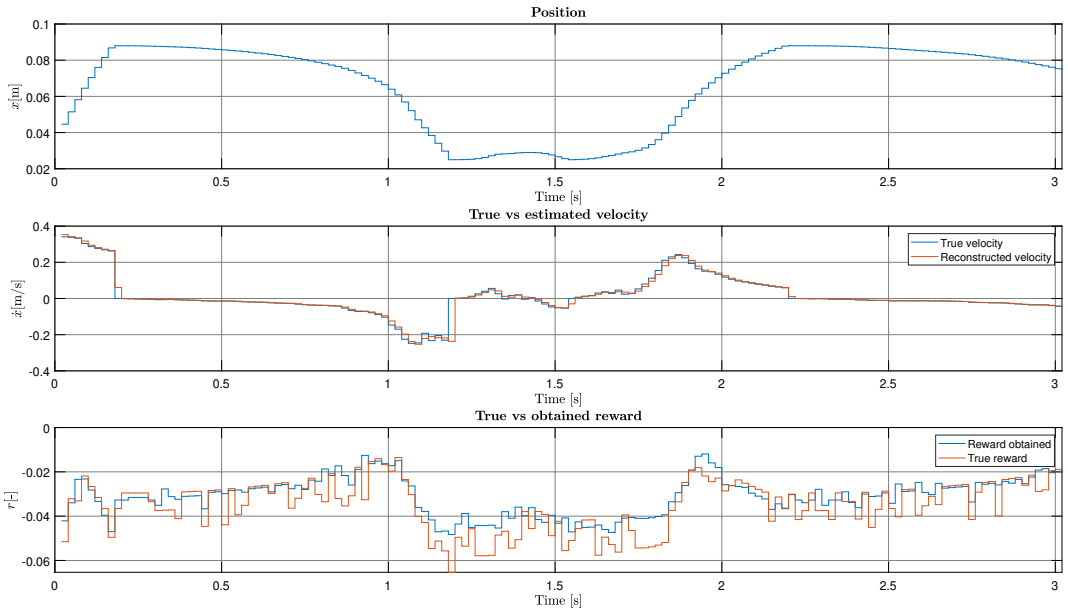



**Figure 4-27:** State evolution and reward during a random episode of the second training run (with the partially observed state vector).

As it appears from Fig. 4-27, the reconstructed velocity is constantly lagging one time-step behind the *true* state of the system, as expected, but the two signals are not too dissimilar.

On the other hand, the effects of such discrepancy on the reward function are more dramatic, as it can be observed in Fig. 4-28.

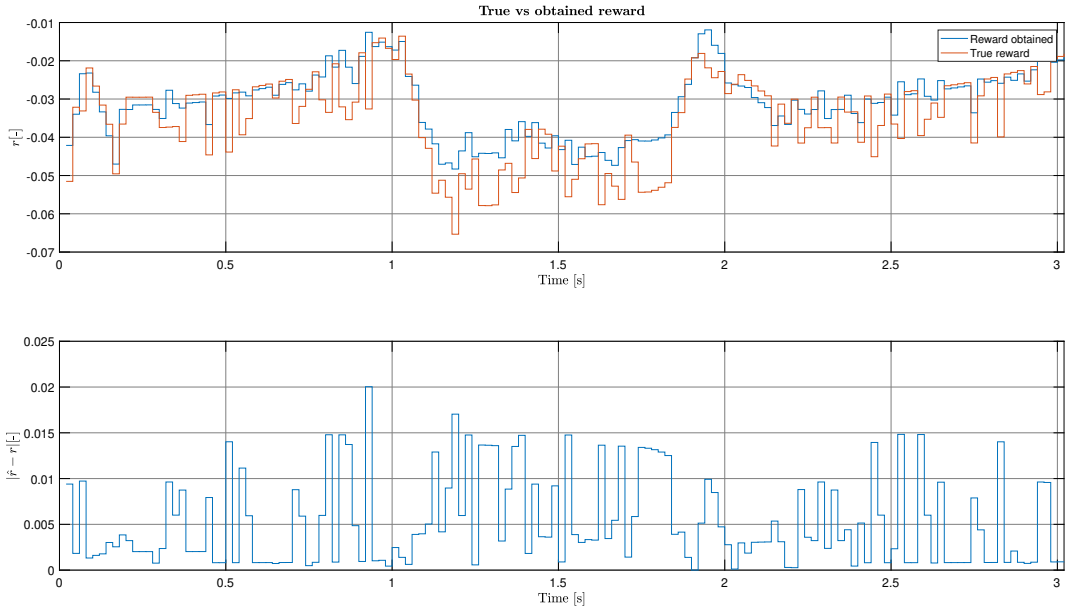



**Figure 4-28:** Comparison of the reward obtained by the system and the reward that would be given to the agent if it could observe the actual system state (Note that the contribution of the control action is the same in both cases)

Such a significant difference (almost up to one fourth of the reward value) affects the update of the value function, since the prediction of future rewards is affected, but this is not all. When it is not exploring, the agent selects the policy action depending on the state it believes the system is in. Fig. 4-29 shows the final policy learned by the agent. As it can be seen, the control actions commanded by the policy are scattered in discontinuous patches across the state space. Therefore, the action selected by the agent can be quite different with respect to the one it should select if it thinks it is in a different state with respect to its actual state.

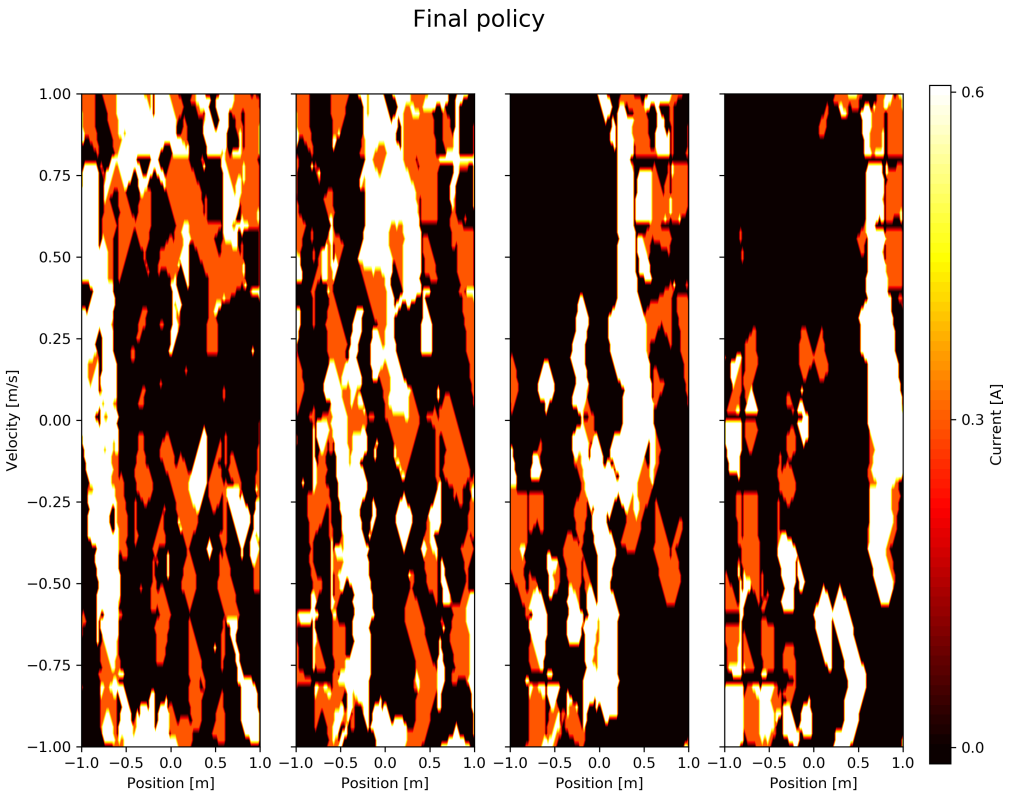



**Figure 4-29:** Final policy learned by the agent that sees the true environment state for the four coils (from coil A to coil D, from right to left).

The combination of these two elements make the task much harder, if not impossible, for the agent to learn.

Furthermore, let it be noted that these simulations were carried out without modeling measurement noise, and on the experimental setup the discrepancy between the reconstructed velocity and the actual velocity, as well as the discrepancy between the reward obtained and the reward corresponding to the actual state of the system, will be more accentuated.

The tests hereby described allowed to conclude that the same high-level definition of the RL task, although is working perfectly well in a simulated environment that provides the learner with full state information, is not suitable to solve the regulation problem on the experimental setup, where part of the state vector is not observed directly.

Simply giving the agent a reward that is not dependent on the velocity but only on the ball position would also not solve the problem, since the agent would still extract actions from the policy based on an inaccurate reconstruction of the state.

# Chapter 5

# Conclusions and recommendation for future work

This research project was developed with the goal to deploy a controller based on Reinforcement Learning (RL) methods to achieve accurate position regulation with a real-world experimental magnetic manipulator (magman) setup.

## 5-1 Conclusions

During this thesis, several agents based on RL methods were deployed in an attempt to solve the magman regulation problem, with the following conclusions.

- A model-based controller based on the Value Iteration algorithm was successfully deployed, both in the simulated environment and on the experimental setup, achieving acceptable performance level in both cases.
- A Q-learning agent (Sutton et al., 1998), tuned by grid-search assisted hyper-parameter manual search, managed to learn the regulation task in a simulated physics environment, but failed when deployed on the experimental setup

It was concluded that the failure of the agents deployed on the real-world setup is mainly due to the following cause.

The physical characteristics of the setup simply make an accurate positioning task very *hard* to learn. Intuitively, reward functions that would encourage the desired behavior would have a steep gradient in the proximity of the target state. In the magman case, however, the target state (an inter-coil position, with zero velocity) and its proximities are visited very rarely during the exploration phase. Therefore, a high reward is not received often. In other words, the reward obtained during interaction with the environment is too sparse for the agent to be able to learn a stabilizing policy. On the other hand, reward functions with more gentle

gradient, such as the linear and quadratic ones, are learnable by the agent, but eventually do not allow it to achieve a high return when deployed on the experimental setup.

The following three other factors are suspected to have contributed, to a lesser extent, to the failure of the agent deployed on the experimental setup.

- The magman setup has a continuous action space, but the Q-learning algorithm operates in environments with discrete action spaces. During this thesis work, the control space was discretized with a relatively low resolution (three possible actions per each of the two coils, and therefore nine possible control actions in total), and while this agent obtained positive results in the simulated environment, empirical evidence suggests that the actual system is more unstable and requires a more careful actuation. Using the Q-learning agent with a finer discretization of the action space, however, is not really a feasible solution, since the sample inefficiency of the algorithm increases exponentially depending on the number of actuators used and on the resolution of the discretization.
- The agents deployed on the experimental setup were not optimally tuned. From the literature, it is evident that RL methods are quite sensitive to hyper-parameters tuning, that can completely jeopardize the learning process if not done properly. In spite of the fact that the algorithm parameters have been carefully tuned in the simulated environment, it is possible that the same parameters were not optimal for the experimental setup, because of some model-plant mismatches. Since hyper-parameter tuning is largely an empirical science, and testing out an extensive number of hyper-parameter configurations on the experimental setup would have been exceptionally time-consuming, it is likely that the tuning used for the agent deployed on the setup was sub-optimal.
- The agent interacting with the experimental setup does not have full access to system-state information, and the steel ball velocity was therefore reconstructed at every time-step by backward-difference of the position measurements. Not knowing the *exact* system state affects to a lesser extent both the reward obtained by the agent and the greedy action commanded by the policy.

## 5-2 Recommendations

In light of the conclusions reached, some suggestions are made for future research.

In order to successfully use RL to learn a magnetic manipulation task, it is advisable to implement a strategy that allows to exploit optimally the experiences collected by the agent during interaction with the environment. The Hindsight Experience Replay (Andrychowicz et al., 2017) method, for example, would increase the sample-efficiency of the learning process and solve the problem of dealing with sparse rewards.

Other techniques such as Prioritized Experience Replay (Schaul, Quan, Antonoglou, & Silver, 2015) could also help to increase the sample efficiency of the algorithms deployed, by replaying more often the transitions that lead to the biggest rewards, from which the agent would learn the most.

Furthermore, it is suggested to use an agent that can naturally work with continuous control actions, such as the actor-critic Deep Deterministic Policy Gradient method (Lillicrap et al.,

2015) or the Normalized Advantage Functions (Gu, Lillicrap, Sutskever, & Levine, 2016) method.

Eventually, advanced hyper-parameters tuning methods based on genetic algorithms (Sehgal et al., 2019), that are not as time-inefficient as the grid-search or random search methods, could be deployed to fine-tune the algorithm parameters for the experimental setup.

Finally, the experimental setup might be improved with an additional sensor measuring the velocity, in such a way to give the agent access to the full state information, or alternatively, more refined reconstruction methods such as an extended Kalman filter could be used instead of the simple backward-difference estimation used in this thesis.

# Appendix A

# Value Iteration

Value iteration is a Reinforcement Learning method that can be used to solve a Markov decision process. It is often used for discrete problems with a limited number of states and actions.

Within this framework, the goal of the agent is to learn an approximation of the Value function ($V^*$) of the MDP. In other words, the agent needs to figure out what is the maximum expected reward that it can obtain by acting optimally, for any given system state $s$. The optimal control policy $\pi^*(s)$ therefore simply consists of the sequence of actions that leads the agent from an initial to the final state with the maximum return. The policy can be extracted from the value function, once it is learned.

At the beginning, the reward function is used to initialize the V-function for every state. Then, the value function is updated using the Bellman equation (A-1):

$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[ R(s,a,s') + \gamma V^*(s') \right] \tag{A-1}$$

Where $T(s,a,s')$ is the transition function from one state of the MDP to the next, and $R(s,a,s')$ is the reward associated to the transition.

A valuable characteristic of the VI method is that it is mathematically proven to converge. Having a proof of convergence is almost never the case for most advanced RL and deep RL algorithms.

The following table summarizes the Value Iteration algorithm:

**Algorithm 3** VI algorithm

1: Initialize the value function using the reward function, for each state.
$\hat{V}_0 \leftarrow \max_a R(s,a,s'), \forall s$
2: **for** k = 1, K **do**
3: Perform a value update (using the Bellman equation).
$\hat{V}_k(s) \leftarrow \max_a(R(s,a,s') + \gamma \sum_{s'} T(s'|s,a) \cdot \hat{V}_{k-1}(s'), \ \ \forall s)$
4: **end for**
5: Extract the optimal policy from the value function.
$\pi(s) = argmax_a R(s,a) + \gamma \sum_{s'}(T(s'|s,a) \cdot \hat{V}_{k-1}(s'))$

In the magman case, both the state space and the action space are continuous. For this reason, they were represented by radial basis functions (RBFs). Each system state was represented by a RBF approximation (a membership degree vector with 17 elements), and the continuous action space of each actuator was discretized in 10 linearly spaced values. In order to simplify the task, only the two middle coils (coil B and coil C) were used. A sampling frequency of 50Hz was chosen. The discount rate in the Bellman equation was set to 0.95. The reward given to the agent during the value iterations was computed with a linear cost function, $r_k = 100 \cdot |x_e| + 0.1 \cdot (|u_1| + |u_2|)$. The target state for the system was set to $\hat{x} = 0.0625\,\text{m}$, $\hat{\dot{x}} = 0.0\,\text{m/s}$. A threshold value $\epsilon_\theta = 0.00001$ was set to stop the algorithm when the change in the parameters of the value function converge to their final value. As an alternative stopping criterion in case the value function does not converge, the maximum number of iterations was set to 1000.

As it is possible to see from Fig. A-1, the method converges in less than 100 value iterations.

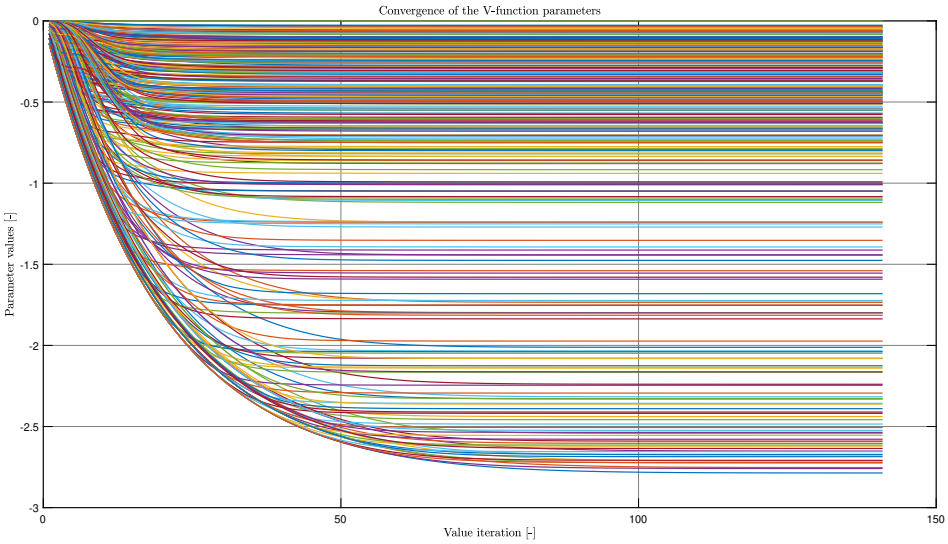



**Figure A-1:** Convergence of the V-function parameters.

Fig. A-2 shows the final policy obtained for the two coils.

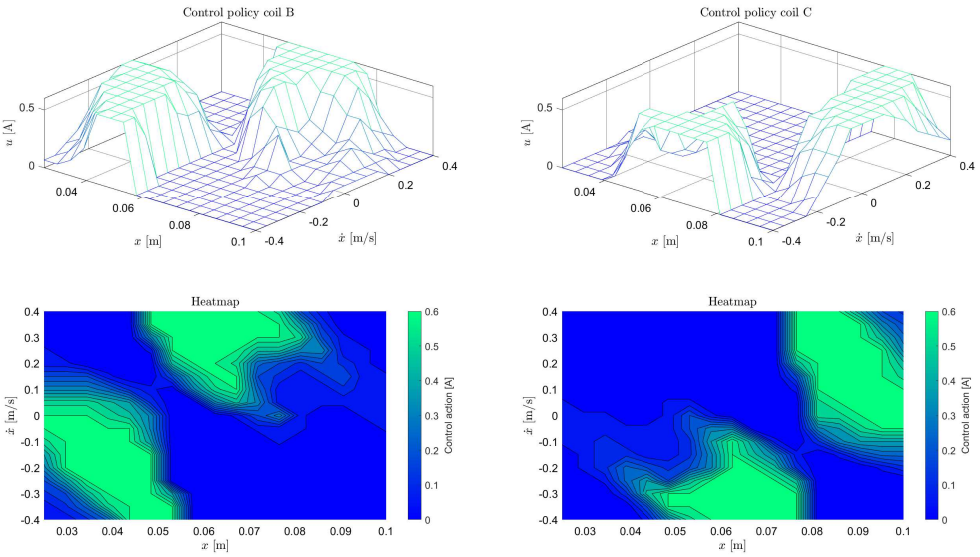



**Figure A-2:** Final policy obtained with the Value Iteration method

Fig. A-3 shows the performance of the agent in 11 tests where the agent is initialized in different parts of the state space. As it can be seen, the performance is optimal in all tests.

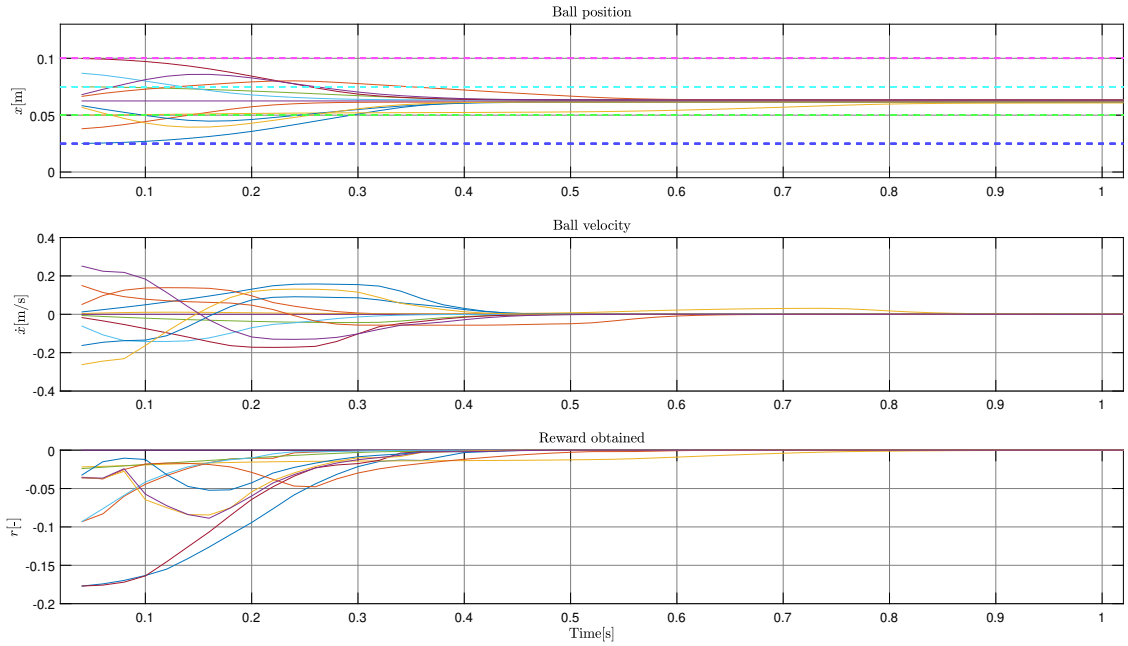



**Figure A-3:** The agent trained with model-based Value Iteration achieves optimal performance.

Eventually, the policy learned off-line was deployed on the experimental setup, and the policy performance was tested against the results obtained in the simulated environment (Fig. A-4 and Tab. A-1).

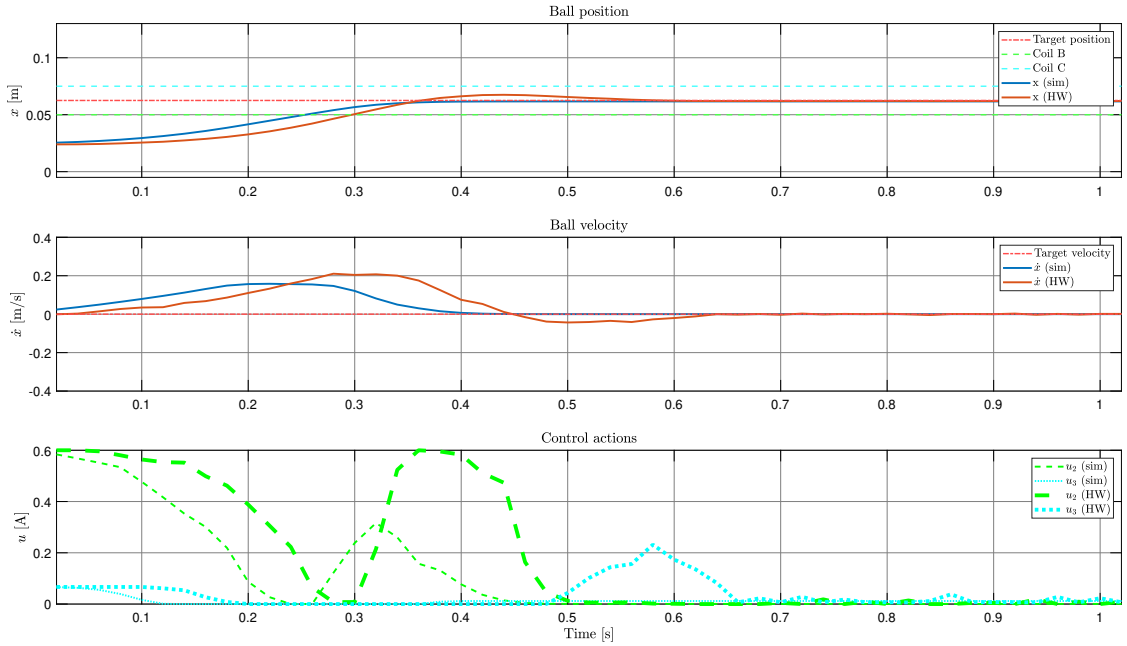



**Figure A-4:** A comparison of the agent's performance in the simulated environment and on the experimental setup.

**Table A-1:** Performance comparison of the policy learned by VI on the simulated environment and on the experimental setup.

| VI-based agent performance | Simulation | Experimental setup |
|---|---|---|
| SS error [m] | 0.000724 | 0.000544 |
| Rise time [s] | 0.22 | 0.18 |
| Settling time [s] | 0.38 | 0.6 |
| Overshoot [%] | 0 | 14.47 |

As it can be seen, the agent is able to achieve regulation with millimetric precision both in the simulated environment and on the experimental setup. The main difference is that the agent deployed on the experimental setup has a longer settling time because of a significant percentage overshoot of around 14% which is not present in the simulated environment.

The success on the experimental setup of the agent trained off-line is due to the fact that the state transition model (2-4) used by the Value Iteration method is quite representative of the actual system, but there are however some model-setup mismatches that are not accounted for by this kind of agent.

For this reason, it is still desirable to develop an agent that is able to learn by pure reinforcement learning, interacting with the real experimental setup.

# Appendix B

# Hardware

This appendix briefly describes the sensor and actuators of the magman experimental setup and the way the I/O devices are interfaced within the Python IDE.

## B-1 Position measurement

The steel ball position along the mono-dimensional track is measured by a *di-soric LAT 61 K 120/120 IUPN* laser displacement sensor, with a frequency of 5kHz. The sampling period can be adjusted depending on the brightness of the object reflecting the laser beam. The sensor readings are acquired by a *Humusoft MF634* data acquisition board which is connected to the computer.

Let it be noted that, with factory settings, the voltage output of the laser sensor is in the range $[0, 11]$V, but the data acquisition board has a maximum analog input voltage of 10V. Therefore, in order to avoid clipping of the observable position range, the analog output range of the laser sensor was changed to: $[0.0, +9.5]$V.

Furthermore, in order to ensure that the laser sensor is used as intended, it is important to note that it features a moving-average filter that pre-processes and stabilizes the measurements. The default window-size is 1024 samples. These factory settings are inadequate for the relatively fast dynamics of the magman setup, and the filter was removed.

These settings can be modified navigating in the sensor menu using the LCD display and the three buttons embedded on the side of the sensor, following the procedure described in detail in the sensor user's manual ("Compact Laser Displacement Sensor LAT 61 User's Manual", n.d.), particularly in Sections *3-3-4 Data Processing Settings* and *3-3-6 Analog settings*.

The sensor can detect objects in the range $[0.060, 0.180]$ m from the laser beam source. The following equation shows how the sensor output (voltage $V_{out}$) is mapped to the ball position

along the mono-dimensional rail ($x$).

$$x = v_{out}\frac{X_{range}}{V_{range}} + \frac{X_{range}}{2} + r_b \tag{B-1}$$

With:

- $X_{range} = 0.12\,\text{m}$, the measurement range of the laser sensor
- $V_{range} = 9.5\,\text{V}$, the voltage output range
- $r_b = 0.012\,\text{m}$, the radius of the steel ball

As aforementioned, the sensor output is sampled and acquired by the *Humusoft MF634* data acquisition board ("MF 634 Multifunction I/O Card User's Manual", n.d.), which has a maximum input frequency of 2.5 MHz. The laser sensor and the data acquisition board are shown in Fig. B-1.

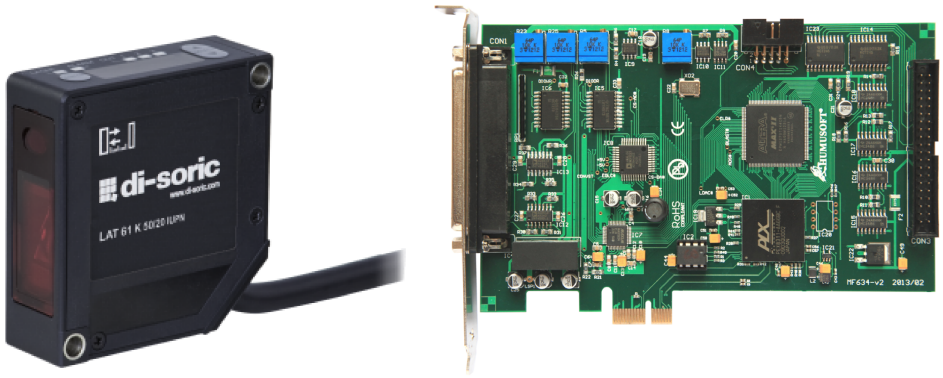



**Figure B-1:** The laser displacement sensor and the data acquisition board used to measure the magman position.

The data is acquired from the control software via a dedicated library written in the *C* language called *hudaqlib*. In order to keep the I/O data handling and the control logic within a single development environment, the *Python* library *ctypes* was used to use write Python-compatible wrappers for the required functions from the original library, allowing to access the device (*HudaqOpenDevice*), acquire the data (*HudaqAIRead*) and close the device (*HudaqCloseDevice*). These wrapper functions allow for seamless integration between the C-library and the Python Software.

**Listing B.1:** *ctypes* function wrappers that allow to read the laser sensor from wihtin Python.

```
1 import ctypes
2 from ctypes import *
3
4 # Open DAQ device handle
5 # ----------------------
6 def open_DAQhandle():
7   # Usage: h = open_DAQhandle()
8   # Opens an handle to access the DAQ device
```

```
9
10    libc = windll.hudaqlib  # Include the header of the "hudaqlib.h" library
11
12    # HudaqOpenDevice CTYPES function signature
13    HudaqOpenDevice_func = libc.HudaqOpenDevice
14    HudaqOpenDevice_func.restype = ctypes.c_size_t  # Function return type
15    HudaqOpenDevice_func.argtypes = [ ctypes.c_char_p, ctypes.c_int, ctypes.c_int
          ]  # Function argument type
16
17    handle = HudaqOpenDevice_func(b'MF634', 1, 0)
18
19    return handle
20
21  # Read the laser displacement sensor
22  # ----------------------------------
23  def read_sensor(handle):
24    # Usage: x = read_sensor(h)
25    # Reads the value measured by the sensor and converts it to position
          measurement
26
27    libc = windll.hudaqlib  # Include the header of the "hudaqlib.h" library
28
29    # HudaqAIRead CTYPES function signature
30    HudaqAIRead_func = libc.HudaqAIRead
31    HudaqAIRead_func.restype = ctypes.c_double  # Function return type
32    HudaqAIRead_func.argtypes = [ ctypes.c_size_t, ctypes.c_uint ]  # Function
          argument type
33
34    V_range = 9.5  # [V] Output voltage range
35    x_range = 120  # [mm] Sensor measuring range
36    x_offset = 0.0  # [mm] Sensor measurement offset
37
38    value = HudaqAIRead_func(handle, 0)  # Voltage measured by the sensor
39
40    x_out = value*(x_range/V_range) + x_offset  # Map the voltage to position
41    x_out = x_out/1000  # Convert from mm -> m
42
43    return x_out
44
45  # Close DAQ device handle
46  # -----------------------
47  def close_DAQhandle(handle):
48    # Usage: close_DAQhandle(h)
49    # Closes the handle to the DAQ device
50
51    libc = windll.hudaqlib  # Include the header of the "hudaqlib.h" library
52
53    # HudaqCloseDevice CTYPES function signature
54    HudaqCloseDevice_func = libc.HudaqCloseDevice
55    HudaqCloseDevice_func.restype = ctypes.c_void_p  # Function return type
56    HudaqCloseDevice_func.argtypes = [ ctypes.c_size_t ]  # Function argument
          type
57
58    HudaqCloseDevice_func(handle)
59
60    return None
```

## B-2 Actuation

The four coils of the magman setup are controlled by two identical control boards. The boards and the four coils of the magman are powered by A 24 V DC supply. Each module controls two of the four coils with PWM current signals. The I/O module on the control boards uses RS-232 communication protocol.

The following Python code snippet shows how the serial messages containing the commands to the control boards are constructed. Although there was no available documentation on the control boards, the communication protocol was extracted from legacy code.

**Listing B.2:** Code used to control the four actuators.

```
import numpy as np
import serial  # PN: pip installation command of this package is <pip install
    pyserial>


# Open serial ports
# -----------------
def open_serialport(portname, brate, t_out):
  # usage: ser1 = open_serialport("COM11", 115200, 1.0)
  #        ser2 = open_serialport("COM12", 115200, 1.0)
  # Open two serial communication ports to the boards controlling the magnets

  porthandle = serial.Serial(port=portname, baudrate=brate, timeout=t_out)

  return porthandle

# Close serial ports
# ------------------
def close_serialport(porthandle):
  # usage: close_serialport(ser1)
  #        close_serialport(ser2)
  #        ser2 = open_serialport("COM7", 115200, 1.0)
  # Close the serial communication ports to the boards controlling the magnets

  porthandle.close()

  return None

# Send control input to the actuators
# -----------------------------------
def send_command(serialhandle1, serialhandle2, u_in):
  # usage: send_command(ser1, ser2, u)
  # Send the desired control action u to the four actuators
  # ser1 and ser2 are two handles to the serial ports used for communication (
      which must be open). u is a vector with four control commands in the range
       [0.0, 0.6] A

  round_vec =np.vectorize(np.round)  # vectorize the np.round function
  fix_vec = np.vectorize(np.fix)     # vectorize the np.fix function
  fmod_vec = np.vectorize(np.fmod)   # vectorize the np.fmod function

```

```
39    # Sanity check and control input saturation
40    for idx in range(len(u_in)):
41      if (u_in[idx]<0.0)or(u_in[idx]>0.61):
42        print('Control input OUT OF RANGE: ', u_in, ' [A]')
43      u_in[idx] = np.maximum(0.0, np.minimum(u_in[idx], 0.6)) # input saturation
44
45    # Form two string messages from the control input according to the
          communication protocol extracted from the legacy code
46    u_send = 2048 + round_vec((u_in/0.55)*2048)
47
48    A1 = 33+fix_vec(u_send[0]/80) # coil A
49    B1 = 33+fmod_vec(u_send[0], 80)
50    C1 = 33+fix_vec(u_send[1]/80) # coil B
51    D1 = 33+fmod_vec(u_send[1], 80)
52
53    A2 = 33+fix_vec(u_send[2]/80) # coil C
54    B2 = 33+fmod_vec(u_send[2], 80)
55    C2 = 33+fix_vec(u_send[3]/80) # coil D
56    D2 = 33+fmod_vec(u_send[3], 80)
57
58    Str1 = "q " + chr(int(A1)) + chr(int(B1)) + chr(int(C1)) + chr(int(D1))
59    Str2 = "q " + chr(int(A2)) + chr(int(B2)) + chr(int(C2)) + chr(int(D2))
60
61    Str1_MSG = [ord(z) for z in Str1]
62    Str1_MSG.append(13)
63    Str2_MSG = [ord(z) for z in Str2]
64    Str2_MSG.append(13)
65
66    MSG1 = np.uint8(Str1_MSG)
67    MSG2 = np.uint8(Str2_MSG)
68
69    serialhandle1.write(MSG1)
70    serialhandle2.write(MSG2)
71
72    return None
73
74  # Power-off the coils
75  # --------------------
76  def poweroff_coils(serialhandle1, serialhandle2):
77    # usage: poweroff_coils(ser1, ser2)
78    # Power-off the four coils
79
80    MSG_PowerOff = np.uint8([80, 32, 48, 32, 48, 13, 10])  # corresponds to the
          control input [0.0, 0.0, 0.0, 0.0] A
81
82    serialhandle1.write(MSG_PowerOff)
83    serialhandle2.write(MSG_PowerOff)
84
85    print("Coils powered off")
86
87    return None
```

## B-3 Connection between control boards and computer

Finally, it was necessary to substitute the USB cables connecting the control boards to the computer due to obsolescence. The new cables are the *FTDI TTL-232R-RPi* cables (USB to TTL level serial UART conversion). Fig. B-2 shows the pin-out of the PCB connector on the boards.

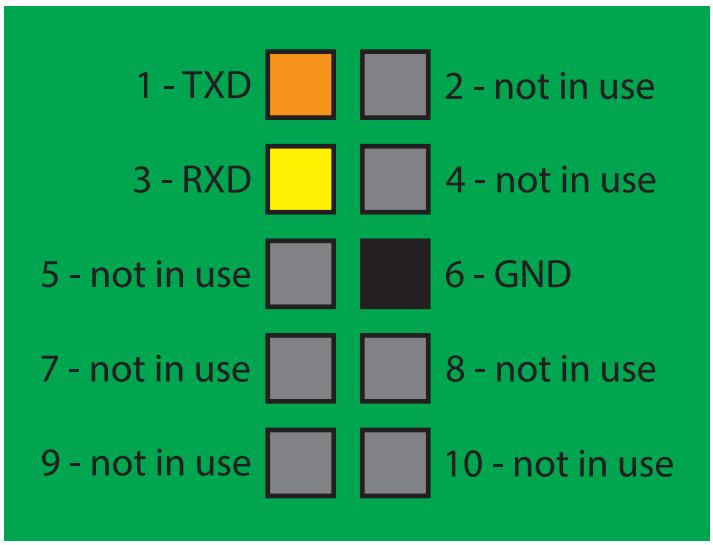



**Figure B-2:** Pin-out scheme of the PCB connector. TXD indicates the transmission cable, RXD indicates the receiver cable, GND indicates the ground cable.

# References


Alibekov, E., Kubalík, J., & Babuška, R. (2018). Policy derivation methods for critic-only reinforcement learning in continuous action spaces. *Engineering Applications of Artificial Intelligence*, *69*, 178–187.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., ... Zaremba, W. (2017). Hindsight experience replay. In *Advances in neural information processing systems* (pp. 5048–5058).

Atkeson, C. G., & Santamaria, J. C. (1997). A comparison of direct and model-based reinforcement learning. In *Proceedings of international conference on robotics and automation* (Vol. 4, pp. 3557–3564).

Bahar, P., Alkhouli, T., Peter, J.-T., Brix, C. J.-S., & Ney, H. (2017). Empirical investigation of optimization algorithms in neural machine translation. *The Prague Bulletin of Mathematical Linguistics*, *108*(1), 13–25.

Barreto, G. A., & Araujo, A. F. (2004). Identification and control of dynamical systems using the self-organizing map. *IEEE Transactions on Neural Networks*, *15*(5), 1244–1259.

Basu, A., De, S., Mukherjee, A., & Ullah, E. (2018). Convergence guarantees for rmsprop and adam in non-convex optimization and their comparison to nesterov acceleration on autoencoders. *arXiv preprint arXiv:1807.06766*.

Baxter, J., Bartlett, P. L., et al. (2000). Reinforcement learning in pomdp's via direct gradient ascent. In *Icml* (pp. 41–48).

Bergstra, J. S., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems* (pp. 2546–2554).

Buşoniu, L., Ernst, D., De Schutter, B., & Babuška, R. (2005). Continuous-state reinforcement learning with fuzzy approximation. In *Adaptive agents and multi-agent systems iii. adaptation and multi-agent learning* (pp. 27–43). Springer.

Compact laser displacement sensor lat 61 user's manual [Computer software manual]. (n.d.). Steinbeisstraße 6 DE-73660 Urbach.

Damsteeg, J. (2015). *Nonlinear and learning control of a one-dimensional magnetic manipulator* (Unpublished doctoral dissertation). Master of Science thesis, Delft University of Technology.

Damsteeg, J.-W., Nageshrao, S. P., & Babuska, R. (2017). Model-based real-time control of

a magnetic manipulator system. In *2017 ieee 56th annual conference on decision and control (cdc)* (pp. 3277–3282).

De Bruin, T., Kober, J., Tuyls, K., & Babuška, R. (2018). Experience selection in deep reinforcement learning for control. *The Journal of Machine Learning Research*, *19*(1), 347–402.

Ferrucci, D. A. (2012). Introduction to "this is watson". *IBM Journal of Research and Development*, *56*(3.4), 1–1.

Friedrichs, F., & Igel, C. (2005). Evolutionary tuning of multiple svm parameters. *Neurocomputing*, *64*, 107–117.

Gu, S., Lillicrap, T., Sutskever, I., & Levine, S. (2016). Continuous deep q-learning with model-based acceleration. In *International conference on machine learning* (pp. 2829–2838).

Guo, X., Singh, S., Lee, H., Lewis, R. L., & Wang, X. (2014). Deep learning for real-time atari game play using offline monte-carlo tree search planning. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems 27* (pp. 3338–3346). Curran Associates, Inc. Retrieved from http://papers.nips.cc/paper/5421-deep-learning-for-real-time-atari-game-play-using-offline-monte-carlo-tree-search-planning.pdf

Hammond, P. (2013). *Electromagnetism for engineers: an introductory course*. Elsevier.

Heess, N., Sriram, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., ... others (2017). Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*.

Hurák, Z., & Zemánek, J. (2012). Feedback linearization approach to distributed feedback manipulation. In *American control conference (acc), 2012* (pp. 991–996).

Kim, H. J., Jordan, M. I., Sastry, S., & Ng, A. Y. (2004). Autonomous helicopter flight via reinforcement learning. In *Advances in neural information processing systems* (pp. 799–806).

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kober, J., Bagnell, J. A., & Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, *32*(11), 1238–1274.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological cybernetics*, *43*(1), 59–69.

Konda, V. R., & Tsitsiklis, J. N. (2000). Actor-critic algorithms. In *Advances in neural information processing systems* (pp. 1008–1014).

Lehman, J., Clune, J., Misevic, D., Adami, C., Altenberg, L., Beaulieu, J., ... others (2018). The surprising creativity of digital evolution: A collection of anecdotes from the evolutionary computation and artificial life research communities. *arXiv preprint arXiv:1803.03453*.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., ... Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, *8*(3-4), 293–321.

Mf 634 multifunction i/o card user's manual [Computer software manual]. (n.d.). Pobřežní 224/20, 186 00 Praha 8-Karlín, Czechia.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... others (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529.

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (icml-10)* (pp. 807–814).

Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R. Y., Chen, X., ... Andrychowicz, M. (2017). Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*.

Polydoros, A. S., & Nalpantidis, L. (2017). Survey of model-based reinforcement learning: Applications on robotics. *Journal of Intelligent & Robotic Systems*, *86*(2), 153–173.

Popov, I., Heess, N., Lillicrap, T., Hafner, R., Barth-Maron, G., Vecerik, M., ... Riedmiller, M. (2017). Data-efficient deep reinforcement learning for dexterous manipulation. *arXiv preprint arXiv:1704.03073*.

Puterman, M. L. (2014). *Markov decision processes.: Discrete stochastic dynamic programming*. John Wiley & Sons.

Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2015). Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.

Schulman, J., Moritz, P., Levine, S., Jordan, M., & Abbeel, P. (2015). High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

Sehgal, A., La, H., Louis, S., & Nguyen, H. (2019). Deep reinforcement learning using genetic algorithm for parameter optimization. In *2019 third ieee international conference on robotic computing (irc)* (pp. 596–601).

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., ... others (2016). Mastering the game of go with deep neural networks and tree search. *nature*, *529*(7587), 484.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. (2014). Deterministic policy gradient algorithms..

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., ... others (2017). Mastering the game of go without human knowledge. *Nature*, *550*(7676), 354.

Simonian, A. (2014). *Feedback control for planar parallel magnetic manipulation* (Unpublished doctoral dissertation). Master's thesis, Czech Technical University in Prague.

Sra, S., Nowozin, S., & Wright, S. J. (2012). *Optimization for machine learning*. Mit Press.

Sutton, R. S., & Barto, A. G. (2011). Reinforcement learning: An introduction.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S., Barto, A. G., et al. (1998). *Introduction to reinforcement learning* (Vol. 2) (No. 4). MIT press Cambridge.

Tassa, Y., Erez, T., & Todorov, E. (2012). Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 ieee/rsj international conference on intelligent robots and systems* (pp. 4906–4913).

Todorov, E., Erez, T., & Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 ieee/rsj international conference on intelligent robots and systems* (pp. 5026–5033).

Verhaegen, M., & Verdult, V. (2007). *Filtering and system identification: a least squares approach*. Cambridge university press.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, *8*(3-4), 229–256.

Zemánek, J., Čelikovskỳ, S., & Hurák, Z. (2017). Time-optimal control for bilinear nonnegative-in-control systems: Application to magnetic manipulation. *IFAC-PapersOnLine*, *50*(1), 16032–16039.