CONTROL AND IDENTIFICATION OF FOURTH ORDER FLUID LEVEL SYSTEM USING NEURAL NETWORKS AND REINFORCEMENT LEARNING

LUCAS GUILHEM DE MATOS

MASTER'S THESIS EM ELECTRONIC AND AUTOMATION SYSTEMS ENGINEERING
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

CONTROL AND IDENTIFICATION OF NON-LINEAR SYSTEMS USING NEURAL NETWORKS AND REINFORCEMENT LEARNING

LUCAS GUILHEM DE MATOS

ORIENTADOR: ADOLFO BAUCHSPIESS, DR.

MASTER'S THESIS EM ELECTRONIC AND AUTOMATION SYSTEMS ENGINEERING

PUBLICAÇÃO: PPGEA.690/2017
BRASÍLIA/DF: MARÇO - 2018

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

CONTROL AND IDENTIFICATION OF FOURTH ORDER FLUIS LEVEL SYSTEM USING NEURAL NETWORKS AND REINFORCEMENT LEARNING

LUCAS GUILHEM DE MATOS

DISSERTAÇÃO DE MESTRADO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE.

APROVADA POR:

[illegible]

ADOLFO BAUCHSPIESS, Dr., ENE/UNB
(ORIENTADOR)

[illegible]

ALEXANDRE RICARDO SOARES ROMARIZ, Dr., ENE/UNB
(EXAMINADOR INTERNO)

Mariana C. B. Matias

MARIANA COSTA BERNARDES MATIAS, Dr., FGA/UNB
(EXAMINADOR EXTERNO)

Brasília, 02 de março de 2018.

FICHA CATALOGRÁFICA

MATOS, LUCAS
Control and Identification of Fourth Order Fluid Level System Using Neural Networks and Reinforcement Learning [Distrito Federal] 2018.
xi+100 p., 210 x 297 mm (ENE/FT/UnB, Master, Electronic and Automation Systems Engineering, 2018).
Master's Thesis – Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Elétrica

1. Reinforcement Learning	2. Neural Networks
3. Adaptive Control	4. Fluid Level
I. ENE/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

MATOS, L. (2018). Control and Identification of Fourth Order Fluid Level System Using Neural Networks and Reinforcement Learning, Master's Thesis, Publicação PPGEA.690/2017, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, xi+100.

Lucas Matos

Lucas Guilhem de Matos

Departamento de Eng. Elétrica (ENE) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

I dedicate this work to all those eager to learn.

ACKNOWLEDGMENTS

I want to thank every person that supported me, not only those who helped me directly, but those who made themselves present during this part of my life. First of all I would like to thank my advisor Prof. Dr. Adolfo Bauchspiess whose guidance was fundamental for developing this work and achieve every result. Also, for the support and comprehension during difficult days. I'd also like to thank two students: Daniel Vicentim and Vinícius Galvão who made themselves present during the whole time and whose ideas and help were outstanding. Gratitude to my dear family and friends who supported me in every single step. Thank you everybody from LARA who made the studying and researching into fun and made the long tiresome lab days into cheerful moments. Finally, I thank FAP-DF for funding this project and make my research possible.

RESUMO

Título:
Autor: Lucas Guilhem de Matos
Orientador: Adolfo Bauchspiess, Dr.

Este trabalho propõe um contolador adaptativo utilizando redes neuras e aprendizado por reforço para lidar com não-linearidades e variância no tempo. Para a realização de testes, um sistema de nível de líquidos de quarta ordem foi escolhido por apresentar uma gama de constantes de tempo e por possibilitar a mudança de parâmetros. O sistema foi identificado com redes neurais para prever estados futuros com o objetivo de compensar o atraso e melhorar a performance do controlador. Diversos testes foram realizados com diversas redes neurais para decidir qual rede neural seria utilizada para cada tarefa pertinente ao controlador. Os parâmetros do controlador foram ajustados e testados para que o controlador pudesse alcançar parâmetros arbitrários de performance. O controlador foi testado e comparado com o PI tradicional para validação e mostrou características adaptativas e melhoria de performance ao longo do tempo, além disso, o controlador desenvolvido não necessita de informação prévia do sistema.

ABSTRACT

Title: Control and Identification of Non-Linear Systems Using Neural Networks and Reinforcement Learning
Author: Lucas Guilhem de Matos
Supervisor: Adolfo Bauchspiess, Dr.

This work presents a proposal of an adaptive controller using reinforcement learning and neural networks in order to deal with non-linearities and time-variance. To test the controller a fourth-order fluid level system was chosen because of its great range of time constants and the possibility of varying the system parameters. System identification was performed to predict future states of the system, bypass delay and enhance the controller's performance. Several test with different neural networks were made in order to decide which network would be assigned to which task. Various parameters of the controller were tested and tuned to achieve a controller that satisfied arbitrary specifications. The controller was tested against a conventional PI controller used as reference and has shown adaptive features and improvement during execution. Also, the proposed controller needs no previous information on the system in order to be designed.

CONTENTS

List of Figures

List of Tables

SYMBOLS LIST

Latin Symbols

w	synaptic weight
s	activation function input
e	instant output Error
k	discrete time
s	reinforcement learning state
r	instant reward
G	expected return
a	action
V	value function of a state
Q	value function of a state-action pair
H	fluid level height

Greek Symbols

φ	activation function of a neuron
ξ	cost function
η	learning rate
δ	local gradient for backpropagation
ρ	radial basis function
μ	center parameter of radial basis function
σ	dispersion parameter of radial basis function
γ	discount factor
δ_{td}	temporal difference error
τ	input to output delay
α	punishment for error variation
β	punishment for tolerance margin
ϵ	tolerance margin

Abbreviations

ANN	Artificial Neural Network
MLP	Multilayer Perceptron
RBF	Radial Basis Function
RBN	Radial Basis Network
RL	Reinforcement Learning

1 INTRODUCTION

It is strange that only extraordinary men make the discoveries, which later appear so easy and simple. ***Georg C. Lichtenberg (1742 - 1799)***

1.1 CONTEXTUALIZATION

1.1.1 History of Neural Networks.

Since immemorial time we have been curious about the human brain and by many means we have tried to understand more about its physiology and, utterly, about cognition, the ability of acquire knowledge and learn through experience. With the advent of mechanics and engineering culminating in the development of machines that could mimic and enhance repetitive and mechanical tasks previously made by human, the urge to develop learning machines and mimic cognition in order to improve such machines was only natural.

The first step through this path was taken by McCulloch and Pitts (1943), when they wrote a paper about neurons, its physical structure and and how they might work trying to understand how the brain was able to deal with highly complex patterns by only connecting simple cells. They proposed a very simple model for the neuron and implemented it as an electric circuit. The main objective of this model was to able to classify simple patterns and until up to today, one of the main ideas around neural networks is the problem of pattern classification and recognition. Some years later, in 1949, Donlad Hebb wrote a book called "The Organization of Behavior" where he postulated that the synaptic connections between neurons were strengthened with usage by biochemical modification or changes in their electrical properties, also, non-existent synapses could be created due to the arrival of an unknown task or pattern to the brain. The idea that already used pathways in a neural network are more likely to be used again and with faster response is one of the main ideas of neural learning and the idea of being able to create new pathways for new tasks is also extremely important.

During the 1950s with the evolution of computing technologies it became feasible to simulate some simple neural networks that were idealized with the theories developed at the time regarding the human brain. By mid 1950s Nathaniel Rochester, an IBM computing researcher headed a group studying pattern recognition and information theory that tried to simulate a neural network and although this first attempt failed soon enough the simulations made by the IBM team started to achieve significant results. It was also Rochester that, alongside Claude Shannon, was responsible for the Darthmouth Conference, using their influence as researchers to enable the conference that is considered by many to be the birth of artificial intelligence.

Rosenblatt (1958), a neuro-biologist of Cornell, intrigued with the eye of a fly, idealized the "perceptron". Based on the neuron model by McCulloch and Pitts (1943), it computes the weighted sum of inputs and limits the output to give a binary answer classifying the given input in one of two classes. The main advance the perceptron had over the McCulloch-Pitts neuron is that it had a learning rule. It was first simulated in 1957 in an IBM 704 computer classifying whether the input image was a triangle or not. Widrow and Hoff (1960) of Stanford developed a neuron model called "ADALINE" and in its multiple version "MADALINE". ADALINE is a perceptron-like neural networks, based on the McCulloch-Pitts model for the neuron, some of the core differences between the MADALINE and the perceptron are the learning rule and the activation function. The name of the model were an acronym to ADAptive LINear Elements and it was used to recognize binary patterns so when reading a series of bits it could predict which was going to be the next bit, while MADALINE was the very first Neural Network to be used in a real world problem as an adaptive filter that can be used to remove echoes on phone line. Surprisingly these models are still in commercial use today.

Although artificial neural networks had had some good results and achievements the traditional von Neumman architecture (Neumman himself had previously suggested to use telegraph relays or vacuum tubes to simulate a neural network) presented more prominent results at first glance and thus, took over the field of computing. At the same time researches such as Minski and Papert (1969) were suggesting that it was not possible to extend the single layered network learning rules to a multi layered network, also in order for the multi layered network to work in a stable manner, it should be fully connected so hardware limitation proved itself very relevant when many of the ideas developed for neural networks were not applicable with current technology. These limitations alongside great exaggeration about the potential of artificial neural networks due to the early successes, caused a general disappointment on the subject since most of the predictions and promises were not held, also, recurrent philosophical questions about artificial intelligence and learning machines coupled with these factors to cause interest on the field to drastically decrease due to lack of researchers and funding as respected voices began to critique the field. Nevertheless, there was some work made during that time, such as some networks developed by Kohonen in 1972, but it was not until the early 1980s that the Neural Network field would have revived enthusiasm and enter its second era.

Hopfield (1982) of Caltech and Cooper et al. (1982) had approaches in supervised learning that would bring back forth attention to the area. Hopfield described a network layout that had bidirectional connections between neurons. The units were McCulloch-Pitts neurons and the network was very well described mathematically being guaranteed to converge to a minimum, although sometimes, to a wrong minimum, displaying wrong pattern recognition or wrong function approximation. Reilly and Cooper described a self organizing network that also used supervised learning where the network had two modules and would separate patterns before classifying them. Still in 1982, Kyoto was home to the US-Japan Joint Conference on Cooperative/Competitive Neural Networks, a conference in which Japan announced increased interest and effort on researches regarding neural networks and on the Fifth Generation o Computing that addresses artificial intelligence (First generation used switches and wires, second generation used the transistor, third state used solid-state technology like integrated circuits and higher level programming languages, and the fourth generation is code generators). With renewed perspectives and the fear of being left behind by Japanese researchers, the US restarted funding to neural network researches while some years later renowned institutions started organizing conferences such as the American Institute of Physics's Neural Networks for Computing in 1985 and the Institute of Electrical and Electronic Engineer's (IEEE) International Conference on Neural Networks in 1987.

Around 1986 the focus was in multi layered neural networks and the new problem lied in how to apply the Widrow-Hoff algorithm through the various layers. Several groups started developing possible solutions but it was from independent research groups that the solution arose. In fact three different groups came up with similar solutions which would later become what we know today to be the backpropagation algorithm. The development of these algorithms was only possible because the activation function in the neurons was changed from a limiter to a non-linear differentiable function which made possible to perform the gradient descent method on a neuron to propagate its error backwards, also, with nonlinear activation functions instead of a limiter, the network output could show not only patterns but could map a set of inputs to a set of outputs in order to perform function approximation. The back-propagation is known to be a slow-learner since it needs a lot of iterations and data to learn but since the number of layers was no longer a problem, several more complex tasks could be achieved by these networks. Since then the development of neural networks themselves is relatively slow because of hardware limitations but the usage of already developed practical solutions is indeed in growth and more and more problems are being tackled with the neural network approach. Recently many works show the spread of neural networks usage and although in essence the neural network is nothing but a pattern recognition device, its various layouts and learning algorithms make them useful for the most various applications.

A so called third era of neural networks came to be when deep convolutional networks were proposed. These networks came to be with the advance of hardware, since they are typically very large networks needing greater processing power and large amount of data to be trained, yet, they can solve the most various and complex tasks.

Narendra and Parthasarathy (1990) were pioneers in using neural networks for identification and control followed some years later by Warwick and Craddock (1996) who showed the usage of neural networks for system identification, more precisely the usage of Radial Basis Networks in comparison to other networks. The problem of system identification itself is way older than the neural networks field, and yet, its still a great challenge where researches thrive. Still regarding system identification, Ferrari and Stengel (2005) showed a study of smooth function approximation with neural networks. Although neural networks were primarily created to perform pattern classification, their usage for function approximation is very common.

Closer to our time, several works show the usage of artificial neural networks in prediction and system identification. Pillonetto et al. (2014) made a research on various methods of system identification where machine learning with neural networks was part of it. Zhu et al. (2014) used radial basis neural networks to predict traffic. The idea of forecasting anything whatsoever with neural networks follows the same basis of system identification where one tries to find correlation between past and current information. Also, it was based on finding the pattern on traffic behavior, what brings back the said main idea around neural networks conception. Usage of neural networks to make prediction and approximation continues to grow. Esfe et al. (2015) used an artificial neural network to model a function to thermal conductivity of a fluid in different temperatures while Wanga et al. (2016) as Pillonetto et al. (2014) made a research in various forms of approximations and pointed out neural network enhanced efficiency. A very interesting work from Lia et al. (2016) shows a self organizing network randomly started for system modeling, concepts introduced by Kohonen and later visited by Hopfield.

Regarding the usage of Multilayer Perceptrons Fan et al. (2015) and Fan et al. (2016) implemented perceptrons to predict carbon and coal prices respectively. Similar to the usage of perceptrons to predict market value, both works use chaotic entries and use the perceptrons to find correlations in between data. results of both works show how the architecture proposed by Rosenblatt is still very useful in all sorts of problems.

Gomes and Medeiros (2015) used radial basis neural networks to predict accidents in power plants, Halalia et al. (2016) used radial basis neural networks to model pressure gradient in oil pipelines and Esfe (2017) continued his work focusing in radial basis networks to model thermal conductivity o fluids. All these work in such different areas have in common the sole idea of finding correlation between past and current information in order to map some sort of function to estimate, predict and model. As said, the development of neural networks themselves have not grown much since the 1990s but the usage of such networks has increased and their efficiency is being tested and restated in several different tasks.

Liu et al. (2016), Lechappe et al. (2016) and Pan and Yu (2017) are recent works showing the usability of neural networks to the task of control. Each work deals with a different problem regarding control such as non-linearity and delay respectively for the first two, and the third deals with the task of controlling a biomimetic system. This is interesting given the date since it shows that the predictive control with neural networks is still a trend.

1.1.2 History of Reinforcement Learning.

Modern Reinforcement Learning came to be around the 1980s and it is the fusion of three different fields of study. The first field, which holds the main essence of modern Reinforcement Learning is the study of learning by trial and error that comes from psychology's behaviorism regarding human and animal learning. Another field is the study of optimal control, with the usage of a value function to be minimized. Typically, the second field would not involve learning at all but the idea of a value function is one of the core concepts to modern reinforcement learning and the basis for its mathematical formulation. The third field is the study of temporal-difference methods which were used in specific cases where the first two would not suffice. The history of Reinforcement Learning was very well detailed and studied by Sutton and Barto (1998) and can be found with further explanation in their book, yet, some major events will be explained here.

The optimal control theory has a large contribution to modern reinforcement learning theory, although not specifically in formalism, but in mathematical formulation and algorithms. Optimal control is the problem of designing a controller that will minimize error and steady system dynamics over time. One way to perform such a task was developed by Richard Bellman with the concept of *return* and *value functions* that represent the return over time. The objective would be to minimize a cost function of the system's state. Bellman developed an equation, known as the *Bellman Equation* and it can be solved by iterative methods, known as dynamic programming. It was also Bellman that presented the discrete stochastic version of the optimal control problem which came to be known as Markovian Decision Processes.

Although dynamic programming and the usage of value function is fundamental for the theory and algorithm development of modern reinforcement learning, the essence on which the subject is based is on the trial and error learning or *unsupervised learning.* In psychology the term *reinforcement* refers to the idea that the outcome of an action tends to change the subjects view regarding said action. Studies in this field date back to Edward Thordnike who, in 1911, came up with the term *Law of Effect* that indicated correlation between the effects of an action and the chance of that action occurring again. Two fundamental features to the theory of trial and error learning are connected to this law. The first is that the actions are selected in order to be experienced, and second, selected after being compared with other actions taken in the same or similar situation. As Sutton and Barto (1998) points out, it is the merger of search and memory meaning that an action is chosen among many options to a specific situation but this choice is influenced by the memory of some already chosen actions. In modern Reinforcement Learning, this merger is usually described as the *exploration vs exploitation* dichotomy, so there must be a balance between the exploration of new actions and the exploitation of already known results.

Around the 1950s, by the beginning of artificial intelligence studies, trial and error methods were being studied as a form of learning to the early artificial neural networks. In 1954, in his Ph.D. thesis, Marvin Lee Minsky idealized what was called a SNARC standing for Stochastic Neural Analog Reinforcement Calculators which worked with charged capacitors as a form of memory for the exploitation and the reward to any response given by the network was given by a supervising humam. It is important to emphasize that even though a human was tending to the process the learning itself is not supervised because it had no right answer or an example to compare its output to. The term *reinforcement learning* though, would not be used until the 1960s, time when Minsky published an article discussing reinforcement learning and the problem of credit assignment that consisted on how to give credit to the various actions taken to reach success. Minsky also researched how a secondary reinforcement, which is common in animal learning, could affect the artificial learning. The secondary reinforcement was the backbone of a technique that would be known as *temporal difference learning* but it was Arthur Samuel who came up with a learning method that used temporal difference ideas applied to the chess playing machine with a function being updated online. This was of major importance since in the majority of cases regarding trial and error learning, the updates would be made just after an *episode* of learning, meaning that the machine would perform the whole task and then, be rewarded in accord to the chosen actions.

One important difference between the studies in neural networks at the time and the reinforcement learning subject is that the reinforcement learning is not a *supervised* type of learning, meaning that it has no parameter of comparison, such as error functions or examples to compare the taken actions to, while the pattern classification methods widely used by Frank Rosenblatt and by Bernard Widrow and Marcian Hoff were supervised. In their research on artificial neural networks, they used concepts of reinforcement learning using even terms as *reward* and *punishment* but the learning was made in a supervised manner. In bibliography, as suggested by Sutton and Barto (1998), the term *trial-and-error* is related to *unsupervised* learning, but it can be often seen erroneously used to describe a network that learns based on error to correct the weights, which is in fact *supervised* learning since the error comes from a comparison to a given parameter. The confusion between these types of learning and the advance in neural networks using supervised learning and later, the disappointment regarding neural networks made the research of reinforcement learning to come almost to a halt, still, some notable works deserve comment as they brought light to new concepts that would be used in modern reinforcement learning.

Michie (1961, 1963) brought to attention a trial and error system to play *tic-tac-toe*. "MENACE" as it was called, standing for Matchbox Educable Naughts and Crosses Engine worked in an episodic manner where every action chosen by MENACE would receive reward or punishment according to the final outcome. Later that decade, Michie and Chambers (1968), developed yet another *tic-tac-toe* example called GLEE and a controller called BOXES that was one of the first trial and error controllers which was used to control a pole being balanced in a cart. The falling of the pole or the cart reaching the end of the track would generate a failure signal. This work was one of the earliest examples of control tasks with incomplete knowledge of a system. Widrow and Hoff had done this task before but with supervised learning, meaning that the controller had a teacher and did not learn by himself, that is, not by trial and error.

In 1973, a very important concept used in modern reinforcement learning and in this work was born when Widrow et al. (1973) used a modified LMS algorithm in which the learning was based on success and failure opposing learning from examples. The concept of *learning from a critic* is the basis for modern actor-critic methods and opposes the *learning from a teacher* which is supervised learning. Later in that decade, Witten (1977) published the first article containing a learning rule based on temporal difference that would be called TD(0) and it was used as part of an adaptive controller to solve markovian decision processes.

Around the 1960s and 1970s the artificial intelligence field had a decrease in productivity and researches still in this area were focusing on supervised methods of learning, and so the adaptive character of learning was disappearing from the artificial intelligence scene. Klopf (1972) with various works published during the 1970s and early 1980s affirmed that the drive to achieve results and avoid undesired situations, fundamental features of animal behavior, were being left out of the scene. His idea is in fact the core essence of trial and error learning, since this method strive to achieve goals and avoid problems as actions that are responsible for success are rewarded and actions that are responsible for failure are punished. Influenced by his work, Richard S. Sutton and Andrew Barto worked towards showing that reinforcement learning was different from supervised learning and also that reinforcement learning could be used as a viable learning algorithm to neural networks. Sutton and Barto (1981) also used the concept of temporal difference in the trial and error learning creating whats is know as the actor-critic architecture.

Sutton (1988) separated the temporal difference learning from the control problem, presenting it as a general estimation method and proving some mathematical rules of convergence. At the time, renewed interest in neural networks and artificial intelligence made the field of reinforcement learning more prominent in these areas. Watkins (1989) developed the Q-Learning algorithm that completed the merger between temporal difference learning and optimal control theories. it was through this work that the three main ideas were united in a single and more robust theory set for the subject of reinforcement learning. Tesauro (1995) was the first to use temporal difference learning to create a backgammon playing program, known as TD-Gammon. Sutton et al. (1991) presented the idea that reinforcement learning was indeed an algorithm of optimal control showing that the algorithms of reinforcement learning focus in minimizing a value function, but using the trial and error concept. Just as in the case of neural networks, not much in the theory of reinforcement learning itself has evolved since the late 1990s, but instead, the usage of the reinforcement learning algorithms and ideas in several areas shows persistent growth.

The usage of Reinforcement Learning as a learning algorithm for Neural Networks can be seen in Slusny et al. (2008) where the exploration of a maze by a robot is made using a reinforcement learning algorithm and radial basis neural networks. In fact this is a matter of minimizing a cost function which is approximated in a radial basis network with a reinforcement learning algorithm. Napoli et al. (2014) used a radial basis networks with reinforcement learning algorithms to implement a semantic identification where using unsupervised learning guarantees usability in any kind of text and content.

Regarding usage of reinforcement learning algorithms in control tasks, it has been used plenty and still is. During the last 2 decades there are several works showing enough result to conclude that reinforcement learning can deal with a range of difficulties and more importantly with a range of different systems and circumstances that indicates that it is a good solution for the adaptive optimal control challenge. Among several works, those who caught attention by some degree of similarity whit this work were performed by Deisenroth and Rasmussen (2009), Syafiie et al. (2011), Lin and Zheng (2012), Yang and Jagannathan (2012), Xu et al. (2014), Kiumarsi et al. (2014), Liu et al. (2015), Luo et al. (2015), Pradeep et al. (2016), Lillicrap et al. (2016) and Kamthe and Deisenroth (2017).

1.2 MOTIVATION

Control is one of the core keys in the development of automation and that is why developing control theory and control techniques is essential. Although those have stride largely since around the 18th century, there are still various causes that enhance the difficulty of the control problem such as non-linearities and time-variance. Time-variant systems are recurrent. Several day-to-day applications present variation on its dynamics, such as voltage controlled/variable gain amplifiers, which are important in sound mixing and synthesizers, aircrafts that have different surface of contact during takeoff, landing and cruise and which fuel consumption makes the weight change over time, a servo motor system with a friction term that changes with time, for example as the bearing lubricant warms up, etc.

To deal with the lack of knowledge on a system to be controlled and parameter variability, reinforcement learning is a very powerful tool as shown by Coulom (2002). Being an unsupervised form of learning, it does not depend on knowledge of the system as it learns through experience and when used as the learning algorithm of a neural network, it can deal with variability of parameters. The objective of every control process is to reach optimality, which is part of the essence of reinforcement learning itself as seen in Vamvoudakis et al. (2016) and Sutton and Barto (1998). That being said, the motivation of this work is to study and detail techniques to develop a controller using artificial neural networks with reinforcement learning.

Another motivation to this work comes from the chosen environment for testing the algorithms. The fourth order fluid level system represents an opportunity to study a very common problem in industry that is fluid level control. This task is found in a series of industrial segments such as water treatment, chemical, petrochemical, biomedical, food and beverage, pharmaceutical and etc.

1.3 WORK DESCRIPTION

Control process will be performed on a fourth order fluid level system which is described in Chapter 3. The system has four consecutive tanks with adjustable gate valves separating them. This system was chosen because it has non-linearities and high valued input to output delay.

The control will be made with a neural network based reinforcement learning controller with the actor-critic method, described in Chapter 2. The functions used in the reinforcement learning problem will be approximated by radial basis neural networks and to deal with the system's delay, a system model will be estimated by a multilayer perceptron to predict future outputs and bypass the delay.

1.4 OBJECTIVES

Beforehand, some premises for the work must be defined. It is assumed that a controller using neural networks and reinforcement learning will perform with no information on the system dynamics only the average input to output delay of the system to be controlled. That being said, the main objective of this work is to design a controller that will demonstrate learning capability and adaptability. This means that the controller should perform better over time and be able to perform efficiently whit no information on the system dynamics and when the system's parameters are changed.

Regarding objective parameters for comparison, the controller will perform a regulatory task and must achieve stationary error inside the margin of 0.5cm reaching the acceptable region faster over time.

To achieve such controller, black box system identification of the system must be done in order to deal with the system's delay. Since the premise is that the average delay is available to the controller, the identification process will be used to bypass the delay.

1.5 RESULTS

Results on the identification process show great performance in comparison to classical models. The Fit to validation data was higher then these models and in some cases, surpassed 95%. The investigation on the different types of networks and layouts not only confirmed features already described in the literature, but also, the information gathered was used to define which networks to be used in the control problem regarding its features. Multilayer Parceptrons were found to be more stable, while Radial Basis Networks were found to respond faster. The Radial Basis Network that used fixed centers performed better then the other networks and so, was chosen to approximate the functions for the controller, while the Multilayer Perceptron with 2 hidden layer has shown remarkable stability, thus it was used for the system identification task.

Results on the controller performance showed that nearly all the objectives set at the beginning were achieved, with exception to the ability to deal with parameter variability which was not tested. The designed controller, when using the right reward function and parameters, could perform regulatory control with the desired efficiency maintaining the stationary error inside the previously defined margin of 0.5cm and showed learning capacity as the performance clearly enhanced over time.

When compared to a PI the designed controller showed worse results when responding to the first reference values, but over time, it became more and more efficient and evened the PI performance surpassing it eventually and being able to steady the reference value faster and with least overshoot.

1.5.1 Comparison to Other Work.

The difference from cited work above from this work is that in almost all cases, whit the exception of Deisenroth and Rasmussen (2009), the problem is either episodic, meaning that the controller tries the same trajectory over and over again until it finds the optimal path or, the control is task-based meaning that the reinforcement learning algorithm is acting in the tasks the controller chooses but not in the actuators themselves. Take the exploration of the maze by the robot as an example. The robot is said to move forward but the speed and the control of the wheels, so the robot can go in the right direction, is transparent to the algorithm because it is already implemented. So, although previously cited work is important to understand the essence of reinforcement learning and its usage alongside neural networks, other works were used as a basis for comparison.

Wang et al. (2016) present a controller using neural networks, but *not* reinforcement learning and although reinforcement learning is the core of this dissertation, an adaptive controller for industrial systems is just ideal for comparing the results and methodologies. Papierok et al. (2008) used radial basis networks with reinforcement learning to control the walking of a robot. As said before this is a task-based control, with episodic learning. The number of actions as the number of mapped positions for a robot are finite, but, even so, it uses a temporal difference learning algorithm, meaning the mathematics development and learning rates can be related.

The control of a quad-rotor by Hwangbo et al. (2017) returns interesting results and although the control is not made directly in the actuator of the helices the possible movements of the quad-rotor are part of a continuum, so, the control of the quad-rotor is continuous. The algorithm is a simpler form of the traditional policy evaluation from classic reinforcement learning seen in Sutton and Barto (1998). Cui et al. (2017) performed control of an AUV (*Autonomous Underwater Veichle*) using the same cost function for minimization and Rao et al. (2016) use a radial basis network with reinforcement learning to control a quadruped robot applying the algorithm to each single leg. The results shows improvement in convergence speed and control accuracy, which matches the so said adaptive characteristic of reinforcement learning.

Finally, the most relevant work for comparison have also performed control with radial basis networks and reinforcement learning in similar environment and tasks, which is fluid level control. Noel and Pandian (2014) used reinforcement learning to control a nonlinear system of fluid level. The neural networks used were proposed by them, meaning that the efficiency and learning rate can be easily compared. Ramanathan et al. (2018) also used reinforcement learning algorithm to perform control of a fluid level system in conical tanks. These work should set a good parameter as basis of comparison and the similarity to this work must be analyzed.

1.6 DOCUMENT PRESENTATION

Beyond this first chapter, chapter 2 brings theoretical background to those who are not familiar with artificial neural networks and reinforcement learning. The chapter explains neural networks structures, fundamentals and mathematical development of supervised learning algorithms, later, it explains the essence of modern reinforcement learning and the learning algorithms and structures based on this idea. Chapter 3 has a description of the system on which the control was performed with all pertinent information such as dimensions, behavior of the most serious non-linearities and the experimental set-up regarding data exchange and control circuits.Chapter 4 explains which were the experiments made in order to identify the system and validate the method. Chapter 5 follows the same line but explaining the development of the control algorithm and its correlation with reinforcement learning ideas. Chapter 6 presents the results obtained and the analysis of those results in light of the state of the art and finally, chapter 7 brings the conclusion achieved by this work and propositions for future work.

2 THEORETICAL BACKGROUND

Nothing in life is to be feared, it is only to be understood. Now is the time to understand more, so that we may fear less. ***Marie Curie (1867 - 1934)***

2.1 NEURAL NETWORKS

2.1.1 The Multilayer Perceptron

The Perceptron is a neuron that works as a binary classifier proposed by Rosenblatt (1958), as depicted in Figure 2.1. The response is either positive or negative to a specific feature and its response does not show how much this neuron reacts to some feature, but only if it reacts to it or not. The perceptron's output gives information on whether the input signal belongs to a given class, or in other words, if it shows some specific feature or not. The early perceptron was proposed for pattern classification, but the usage in this work will be for function approximation.

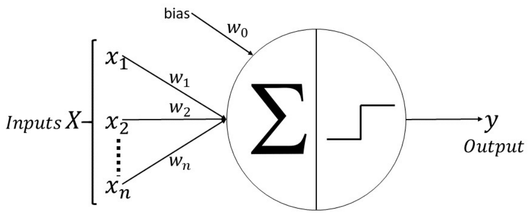


Figure 2.1: Rosenblatts's perceptron.

The “multilayer perceptron” (MLP), illustrated in Figure 2.2, is and Artificial Neural Network (ANN) made by two or more concatenated layers of perceptron neurons that have, instead of a limiter, a non-linear differentiable activation function, usually a sigmoid such as the logistic function and the hyperbolic tangent function. The network if fully connected, meaning that all the neurons in the network are connected to all other neurons in previous and next layers, which is needed for convergence as seen in Minski and Papert (1969). Also, all the connection between neurons are weighted. Since the output of each neuron in the network is continuous due to the activation functions, the MLP can also be used for mapping input to output data in order to perform function approximation, a feature that was explored in this work. As described in Chapter 4, the MLP was used to create a model of the chosen system in order to predict future outputs and bypass delay.

From Figure 2.3, that illustrates a neuron in a hidden layer with more detail, the notation used in this work will be defined. X is the network's input vector and Y is the network's output vector. The input of neuron j in layer l is defined as s_j^l and y_j^l is defined as the output value of neuron j in layer l. Finally w_{ji}^l is the weight that connects the i^{th} neuron of layer $l-1$ to the j^{th} neuron of the layer l. It can also be inferred from Figure 2.3 that,

$$s_j^l(k) = \sum_{i=0}^{N} w_{ji}^l(k) y_i^{l-1}(k) \tag{2.1}$$

with k being the discrete time iteration and N being the number of neurons in layer $l-1$. Also,

$$y_j^l(k) = \varphi_j\left(s_j^l(k)\right) \tag{2.2}$$

where $y_0^l(k) = 1$ is the bias of layer l and layer $l = 0$ is the input layer, and thus $y_j^0(k)$ is not defined by Equation 2.2, but it is the input vector. Note that the input vector is not truly a layer of the network but it will be represented as one in order to simplify notation.

In order to train a MLP one should update solely the values of the weights. The values of these weights is what ultimately defines the network itself since it is in these values that the experience acquired from learning is stored. There are several learning algorithms that can be used to train the MLP, but the most common is the backpropagation algorithm, which was also used in this work.

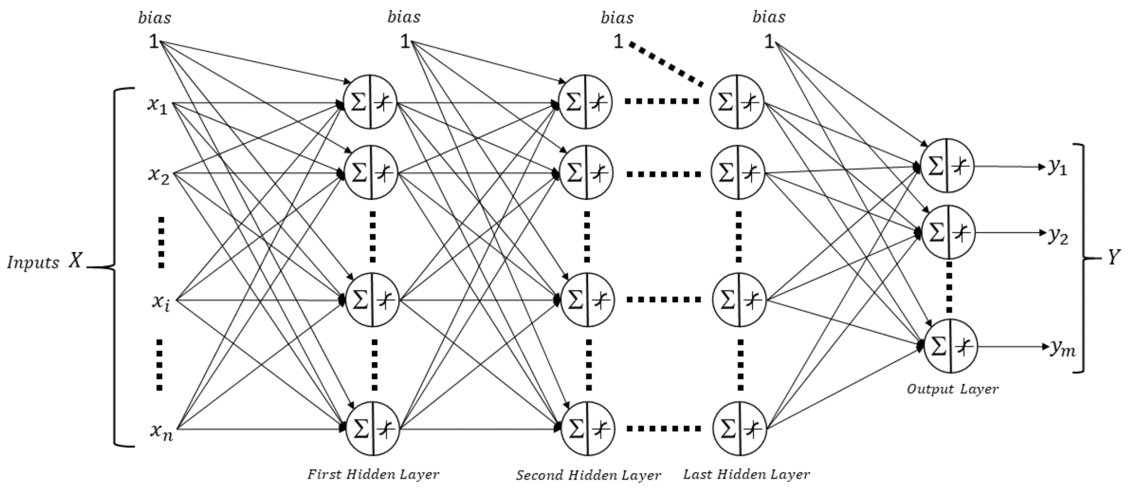


Figure 2.2: Multilayer perceptron.

2.1.1.1 The Backpropagation Algorithm

The backpropagation algorithm is a form of supervised learning and consists in sending corrections backwards through the network to change the weights accordingly to its influence on the network's output. The algorithm applies a correction in a given weight that is proportional to the partial derivative of a given cost function in relation to that weight. A more detailed version of the algorithm's explanation can be found in Haykin (2009), Chapter 4, section 4.4, although, the description in this work follows the same steps with a change in notation regarding the input to the activation function, as Haykin (2009) uses the letter v, in this work the letter s is used instead.

Suppose a MLP with one or more hidden layers, and an output layer with C neurons. Lets define $e_j(k) = d_j(k) - y_j(k)$ the output error of the j^{th} neuron in the output layer in discrete time k, where $d_j(k)$ is the desired

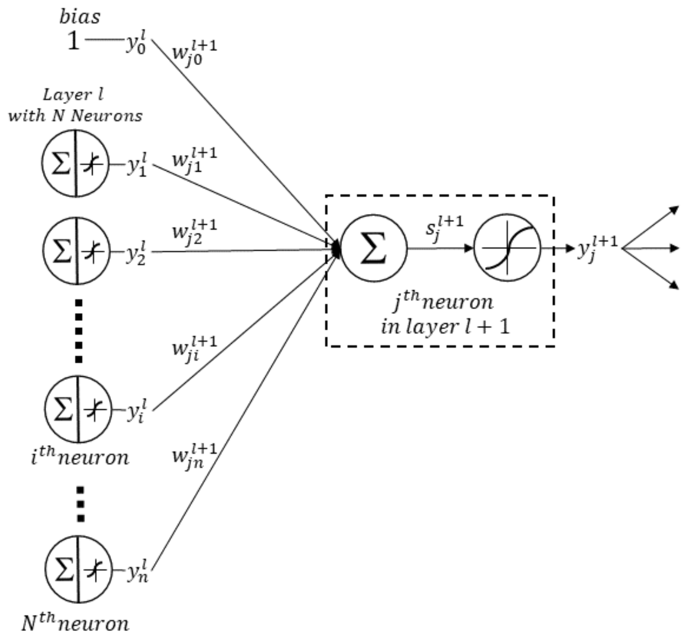


Figure 2.3: Detailed hidden neuron on a MLP.

output and $y_j(k)$ the calculated output, also, lets define $\xi_j(k)$ as the cost function to be minimized for the j^{th} neuron of the output layer as seen in Equation 2.3. The cost function for the full network is the summation of the expression for each neuron in the output layer, as seen in Equation 2.6, where C is the number of neurons in the output layer. Note that the cost function can be any, but the quadratic error was chosen because it gives simpler expressions for explanation.

$$\xi_j(k) = \frac{1}{2} e_j^2(k) \tag{2.3}$$

$$\xi(k) = \frac{1}{2} \sum_{j \epsilon C} e_j^2(k) \tag{2.4}$$

The algorithm consists at first in calculating the gradient of the cost function with respect to the weights and use it as a parameter for weight increment. The expression to be calculated is

$$\frac{\partial \xi(k)}{\partial w_{ji}(k)}.$$

Following the *chain rule* of calculus and using Equations 2.1 and 2.2, the expression can be written as,

$$\frac{\partial \xi(k)}{\partial w_{ji}(k)} = \frac{\partial \xi(k)}{\partial e_j(k)} \frac{\partial e_j(k)}{\partial y_j(k)} \frac{\partial y_j(k)}{\partial s_j(k)} \frac{\partial s_j(k)}{\partial w_{ji}(k)}. \tag{2.5}$$

Each part of the equation can be easily solved as seen below. From Equation 2.6,

$$\frac{\partial \xi(k)}{\partial e_j(k)} = e_j(k). \tag{2.6}$$

Note that the instant error is defined by $d_j(k) - y_j(k)$ and since the expected value of the output is a constant during the iteration,

$$\frac{\partial e_j(k)}{\partial y_j(k)} = -1. \tag{2.7}$$

From Equation 2.2, also using the *chain rule,*

$$\frac{\partial y_j(k)}{\partial s_j(k)} = \varphi_j^{'}(s_j(k)) \tag{2.8}$$

and finally, the last expression can be inferred from Equation 2.1 as

$$\frac{\partial s_j^l(k)}{\partial w_{ji}^l(k)} = y_i(k). \tag{2.9}$$

where $l - 1$ denotes the layer before the output layer.

Using Equations 2.6 through 2.9 in Equation 2.5 yields

$$\frac{\partial \xi(k)}{\partial w_{ji}(k)} = -e_j(k)\varphi_j^{'}(s_j(k))\, y_i(k). \tag{2.10}$$

Since $\frac{\partial \xi(k)}{\partial w_{ji}(k)}$ shows how much the cost function is affected by a given weight, the increment in that weight, denoted by $\Delta w_{ji}(k)$, should be proportional to this value. The expression that defines the value of $\Delta w_{ji}(k)$ is

$$\Delta w_{ji}(k) = -\eta \frac{\partial \xi(k)}{\partial w_{ji}(k)} \tag{2.11}$$

where η is the learning rate and the minus signal exists in order to send the weight on the opposite direction of its influence since the cost function in based on the error, this is why the method is called *Gradient Descent.* Using Equation 2.10 in Equation 2.11 yields

$$\Delta w_{ji}(k) = \eta e_j(k)\varphi_j^{'}(s_j(k))\, y_i(k). \tag{2.12}$$

Haykin (2009) defines the expression $e_j(k)\varphi_j^{'}(s_j(k))$ as the *local gradient* $\delta_j(k)$ and divides the expression of weight updating in three parts, the learning rate, the local gradient, and the *input* of the layer, that in Equation 2.12 is the term $y_i^{l-1}(k)$ and so the expression becomes

$$\Delta w_{ji}(k) = \eta \delta_j(k) y_i(k) \tag{2.13}$$

The only problem with the expression above is that the terms $e_j(k)$, $d_j(k)$ and, consequently, $\delta_j(k)$, are only well defined for the neurons in the output layer since there is no expected response for a neuron located on a hidden layer. The so called error for the hidden neuron in layer l is propagated backwards from the neurons in layer $l+1$, hence the name *backpropagation* of the algorithm, and since the network is fully connected, it receives the errors from all the neurons in layer $l+1$, also because of the connections, the error of a neuron in layer $l+1$ will be propagated to all the neurons in layer l and it is reasonable to conclude that this propagation is weighted according to the influence of each neuron in layer l to the neuron in layer $l+1$, thus the expression for the error in the i^{th} neuron in layer l is given by

$$e_i^l(k) = \sum_{j=0}^{M} \delta_j^{l+1}(k) w_{ji}^{l+1}(k) \tag{2.14}$$

with M being the number of neuron in layer l. Expression 2.14 has layer $l+1$ as the output layer, but this step can be made recursively to propagate the error to layers $l-1$, $l-2$ and so on until layer 1.

With the local gradient defined as

$$\delta_i^l = \varphi_i^{'}\left(s_i^l(k)\right) e_i^l(k), \tag{2.15}$$

using Equation 2.14 in Equation 2.15 yields

$$\delta_i^l = \varphi_i^{'}\left(s_i^l(k)\right)\left(\sum_{j=0}^{M} \delta_j^{l+1}(k) w_{ji}^{l+1}(k)\right) \tag{2.16}$$

with M being the number of neuron in layer l.

Finally, using Equation 2.16 in equation 2.12 the expression for updating weights in hidden layers becomes as follows,

$$\Delta w_{ji}^l(k) = \eta \varphi_j^{'}\left(s_j^l(k)\right)\left(\sum_{n=0}^{M} \delta_n^{l+1}(k) w_{nj}^{l+1}(k)\right) y_i^{l-1}(k) \tag{2.17}$$

with M being the number of neurons in layer $l+1$ and remembering that layer 0 is the input layer.

2.1.2 Radial Basis Networks

2.1.2.1 Radial Basis Functions

Radial basis functions (RBF), are real valued functions in which the output value depends on the distance of the input to a central point and decreases or increases monotonically according to this distance and a dispersion parameter as seen in Orr (1996); Haykin (2009).

A common example of a radial basis function is the Gaussian function seen in equation 2.18 and Figure 2.4 which show some examples, in which as the distance from the center μ increases, the function value decreases in a way dictated by the dispersion parameter σ^2 also known as the variance.

$$f(x) = e^{-(\frac{||x-\mu||^2}{\sigma^2})} \tag{2.18}$$

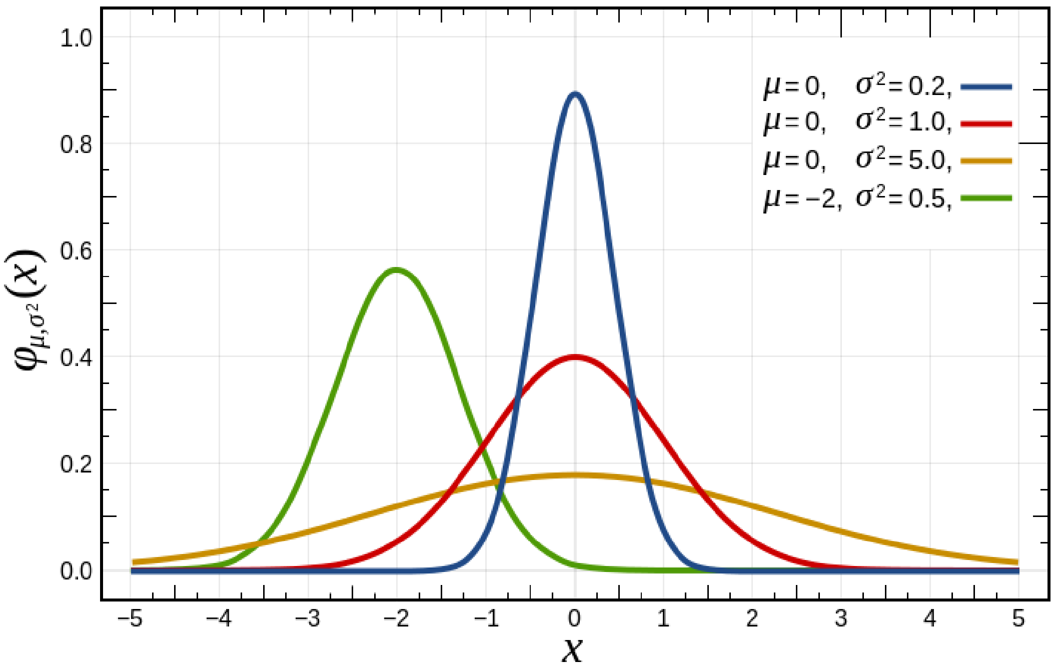


Figure 2.4: Gaussian functions.

In opposition to the perceptron, the RBF does not classify a given input testing if it belongs or not to a specific group, or shows or not a specific feature, but shows *how close* from a specific group the input is or *how much* an input influences in the outcome of the RBF in order to approximate a function Yu et al. (2014).

2.1.2.2 Network Structure

A Radial Basis Function Network (RBN), is nothing more than the weighted sum of outputs of RBFs of the input signal Orr (1996); Haykin (2009). Figure 2.5 shows the structure for the RBN, and from the figure and definition, the expression that defines the output for the RBN can be inferred as follows:

$$y = \sum_{k=1}^{N} w_i \rho(||x - \mu_i||)$$

where N is the number of neurons in the hidden layer. Alike the *Fourier Series*, that represents any signal by a weighted summation of sinusoidal signals, the RBN can represent any function by performing a weighted summation of non-linear functions.

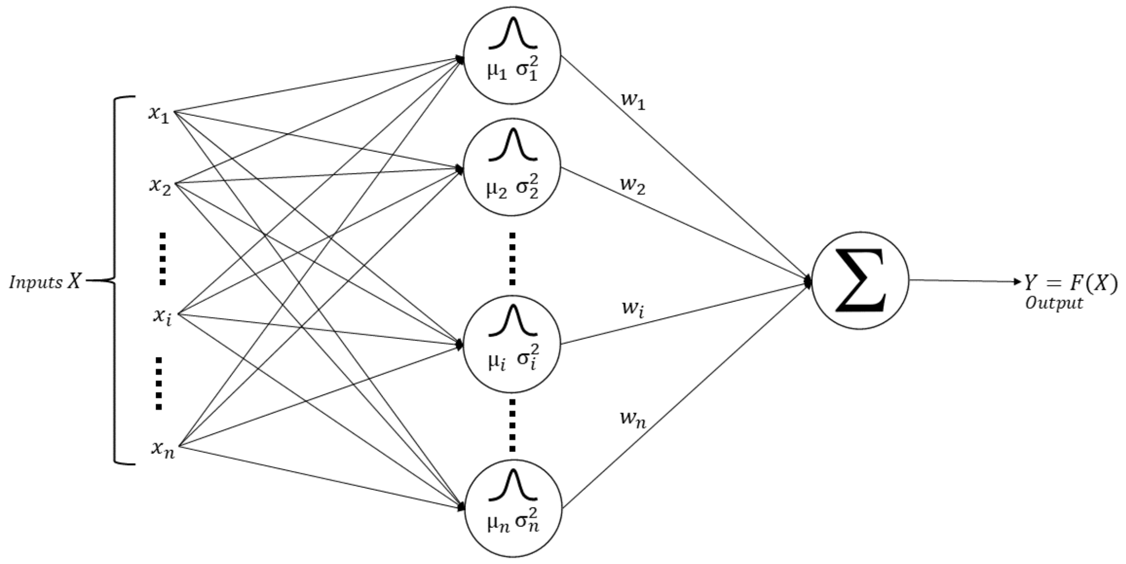


Figure 2.5: The Radial Basis Network structure.

To design a RBN for function approximation, one must consider which activation function is being used, its spread and centers and consider that the response of the neurons must be overlapping meaning that the RBF's must have a degree of intersection in order to contemplate all the input space. In order to reduce the number of neurons, besides the usual training of the weights, the centers could also be trained, either by clustering methods or via the back propagation algorithm as well. Both methods can be found with deep description in Haykin (2009) and will be discussed further in Chapter 4. RBNs where used to approximate the functions needed by the reinforcement learning as it will be described in following section 2.2.

2.1.3 Clustering

Clustering is a form of unsupervised learning whereby a set of observations (i.e., data points) is partitioned into natural groupings or clusters of patterns in such a way that the measure of similarity between any pair of observations assigned to each cluster minimizes a specified cost function, Haykin (2009). A cluster, or a group of similar data observations, has a parameter called *center* that is used as a reference to set the parameter that will be used to measure similarity since it would be impractical to compare the new observation with all other previous observations and even if it was, this information might not be available. Regarding function approximation with clusters, the training will result in cluster centers spread on the input data range. Greater concentration of clusters in a region of the input set means that the output is more sensitive to this input region, while sparse cluster centers, means that the region does not affect the output so drastically. For all the networks used for system modeling the input was the vector containing the delayed input signal and the delayed measured output. The clustering process for the system used in this work is illustrated in Figure 2.6 where the centers of each cluster that was defined by a similarity parameter is depicted. In Figure 2.6 the values of the centers are not as used in the coding because the values for the signal sent to the actuator and the fluid level were normalized, yet, from the Figure, it can be clearly seen that the clusters are mapping a region of correlation between input and output. The distribution of these centers show the region on which the system works. The x-axis shows the input signal and the y-axis the output signal.

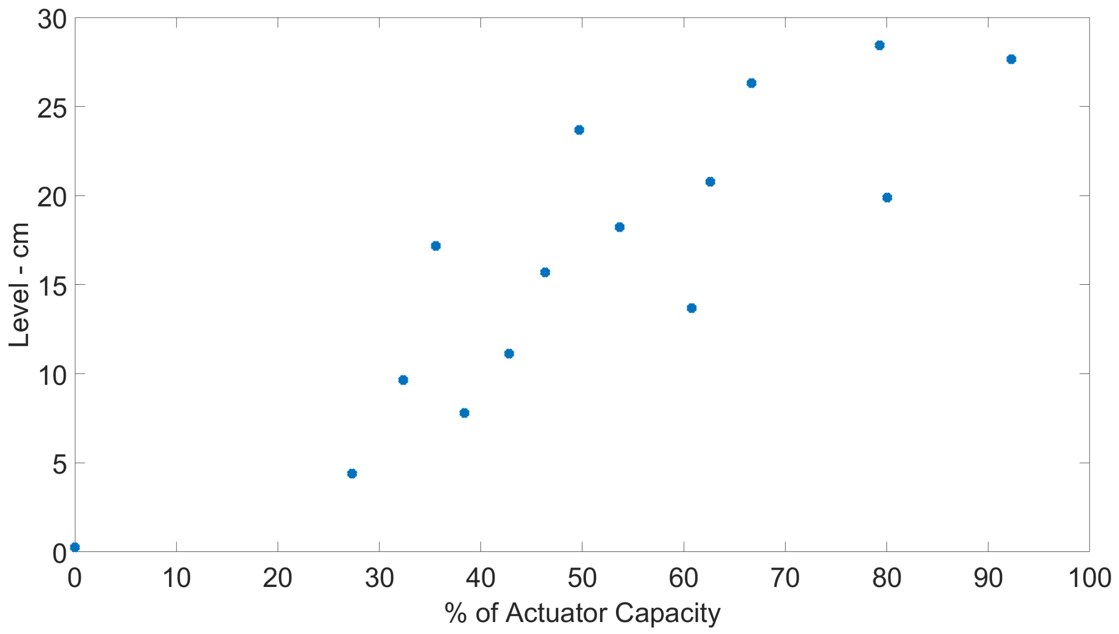


Figure 2.6: Observations assigned to clusters.

The center creation for a clustering problem can be *static* or *dynamic*. Static means that the number of clusters is defined before the process and will remain the same during the taks whether the center values are updated or not. When the number of centers is static a new observation will be assigned to the cluster whose center presents the greatest similarity to it and new data is always assigned to some existent cluster. Dynamic means that new clusters are created only when necessary, that is, when a new observation does not match data similarity requirements for all existing clusters, a new cluster is created, although, usually, the algorithm limits the maximum number of clusters.

The values for the cluster center can be updated or not. If not updated, these values remain the same since the creation of the cluster, for both cases, static and dynamic. The update can be made online, meaning that at every new observation a correction to the center values will be made, also, the values can be updated after an episode, using all the collected data during the episode to define the best values for the cluster centers.

2.2 REINFORCEMENT LEARNING

2.2.1 Concept and Elements

Although not being the only path, and sometimes not the most efficient, experience is the most familiar way for us to learn. Learning through experience is to analyze the result of our actions in a surrounding environment and understand whether it brings us closer or further to our goal. Since infants, it is common for us to analyze our movements in order to reach places we desire to go, to evaluate flavor in order to eat only what we like, etc. In psychology that is a field of study called *Behaviorism* that states that all behavior is in fact reflexes to results of interaction with the environment, but also taking in account previous experience, that is, previous results from previous interactions. Reinforcement Learning (RL), is a machine learning technique that uses the same principle as Behaviorism and uses the experience of interacting with the environment as the parameter for learning. The elements are analogous, but not precisely the same once the RL has definitions and elements narrowed in order to be part of a computational algorithm.

The RL problem can be presented as an individual or *agent,* that according to an *action policy,* interacts with a given *environment* through *actions* chosen by the policy. At each interaction the environment generates a *reward* to the agent that shows him how good the action was, that is, if the action brings the agent closer or further from its *goal*. Using these rewards as parameter, the action policy is modified to try achieving greater reward with new actions in the next interactions. Sutton and Barto (1998).

The most conceptually simple elements in the RL problem are the agent, the environment and the actions, yet, the interface between these elements is not always simple or intuitive. The agent is the learner, the element that acquires experience while the environment is the element the agent gets its experience from. Analogous to the real world, different environments generate different kinds of experience and each environment responds differently to similar interactions. The action is the way the agent interacts with the environment in which he is inserted on, that is, it shows how the agent *behaves* in the given environment. Although behavior is not an element of the RL problem, a reward is in fact the reflex to a behavior expressed by the action, not to the action itself. The action is what the agent do in a given situation and the behavior is how the situation has changed because of that action which is shown to the agent through rewards.

The action policy is a set of rules with which the agent defines what action to take in a given situation in the environment. This set of rules are changed as new information on the agent-environment interaction comes to light in the form of rewards. The way this set of rules is changed is not important, there are many different techniques, but it suffices for the reader to understand they can be changed.

Rewards define the immediate quality of an action and show whether the behavior of the agent is in accord with his goal. In the RL problem formulation rewards are given solely by the environment, opposing intuition, and are not generated in the agent as it is in most systems or in real life (humans as agents generate rewards through senses and psychological responses), but are generated in the environment. The definition of which elements of the problem belong to the agent and to the environment is what defines how the reward will be generated. Suppose a human being, the sensing mechanisms might be considered part of the environment as seen in Sutton and Barto (1998). In Robotics, suppose a maze travel problem, the sensors used for navigation, even being physically part of the robot are considered, for the RL problem, as part of the environment. In short, everything that is used to generate reward is considered part of the environment.

An example follows in order to illustrate the RL problem in a given situation with the elements described so far. Imagine a lost soldier looking for an extraction point in a jungle. He can see the Helicopter in the sky, but cannot see his way through the woods and he has to walk until he finds the clearance where the helicopter is flying above. The soldier is the agent, the jungle and his eyes and ears are the environment, actions are steps in any direction and behavior is getting closer or further from the helicopter. The action policy is what direction should he walk to and the reward is given according to the change of distance from the helicopter. So it begins. Suppose the man's action policy is to go north so he chooses his action and takes a step in the north direction. Suppose as well that the helicopter is straight to the south of him, the environment will show him, through the reward, that he is getting further from his goal so this behavior will be considered bad and receive negative reinforcement to change his action policy, that is, the direction in which he is walking. Suppose the man's action policy changes to go south so he chooses a new action and takes a step south. In this interaction, the environment shows that he is closer to the goal so it enforces this behavior through the reward and the action policy will be enforced in this sense. And so it goes until the soldier reaches the clearance.

Note that in the given example, the action of taking a step is not good or bad, but it depends on what it represents in the given situation. Getting closer or further from the helicopter can be defined as bad and good but in both cases, steps are needed, also if the helicopter was in a completely different position, the actions and directions would still be the same, but the way they would be interpreted would change.

The RL problem is formulated discretely and even to continuous systems, the RL problem is based on discrete interactions, thus, it is based on the concept of *states.* The situation in which the agent is regarding the environment and the agent's goal is the state of the problem. For example, in the lost soldier problem it could be the distance between the soldier and the helicopter, how close the soldier is to a river, how many dangerous animal are close to him, etc. so, for each position in the forest the soldier is in is a different state and evokes different behavior and rewards.

The action policy described earlier uses the state's information to decide which action to take, or as said, how to behave. In fact the action policy if a function that maps states to actions, that is, what action to take in a given state. Also in fact, rewards are given upon reaching a state, so the reached state is what describes whether a behavior was bad or good.

2.2.1.1 Return and Value Function

As said before, rewards define the immediate quality of an action, but the objective of the RL is to achieve maximum accumulated reward, meaning that the agent is not only interested in instant high valued rewards, but even more, in expected future rewards, which is called *expected return, or G.*

The expected return in this work is defined as the discounted sum of expected future rewards, as it can be seen in Equation 2.19 and it is the Bellman Equation for the expected rewards. The term γ is the discount factor and has a value between 0 and 1. It exists to give less value to rewards that are further from the current state.

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + ... + \gamma^M r_{t+M} \tag{2.19}$$

where M is the reward horizon. In theory, as proposed by Bellman, M equals ∞, yet since an infinite horizon in impractical, we define it to be finite.

The expected return of a given state is also called the state's *value* and it is mapped to every state by the *value function.* The value function, just as the action policy, is a function of the information given by the states and it maps states to values, that is, what is the expected return of each state visited by the agent. Figure 2.7

shows the structure of the RL problem as described so far.

Since the main goal of the RL is to achieve maximum return, the action policy tends to choose an action that leads to the highest valued state that can be achieved from the current state. It is wise for the designer of the problem to use a little bit of randomness in the action policy so the agent will explore states he would not choose because of the policy and maybe find out that it is, in fact, a better way. This is called *exploration* and it is useful to find new paths for the problem, sometimes even more efficient. Using only previous information is called *exploitation* where the agent only exploits what it already knows.

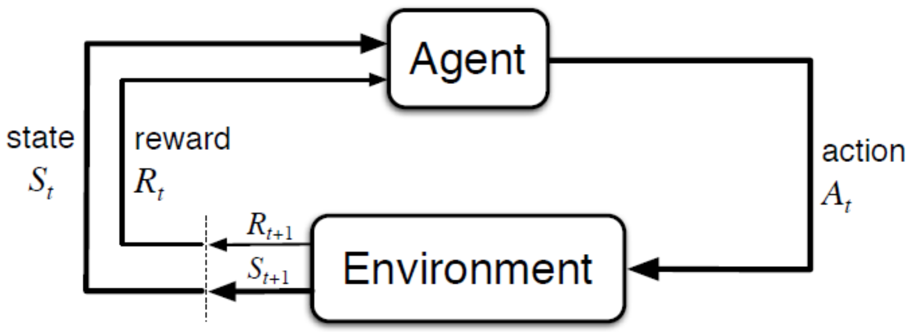


Figure 2.7: The Structure of the RL process Sutton and Barto (1998).

2.2.2 Temporal Difference Learning

When it comes to the learning process, every function needs a goal in order to compare it's output and so correct its parameters to match the given answer. One of the techniques that are closely linked with RL is the Temporal Difference Learning (TD Learning*)*. Mainly, it consists in bootstrapping the value function, that is, using an estimate to correct another estimate. The reason for using estimates is that the RL learns based on experience and have no prior information on the functions that are used for the action policy and the value function, hence, it starts estimating them.

Recall from last section the equation that defines the value of a state;

$$G_t = V(s_t) = \sum_{i=0}^{\infty} \gamma^i r_{t+i+1} \tag{2.20}$$

the expansion of this equation gives

$$V(s_t) = r_{t+1} + \sum_{i=1}^{\infty} \gamma^i r_{t+i+1} \tag{2.21}$$

that generates

$$V(s_t) = r_{t+1} + \gamma \sum_{i=0}^{\infty} \gamma^i r_{t+i+2} \tag{2.22}$$

that yield

$$V(s_t) = r_{t+1} + \gamma V(s_{t+1}) \tag{2.23}$$

which states that the sum of the discounted estimated value of the next state value and the next state reward is the target to the actual estimated value. The error between the target and the actual estimated value $\bar{V}$ is called *Temporal Difference Error,* denoted by δ_{td}, that is;

$$\delta_{td} = r_{t+1} + \gamma \bar{V}(s_{t+1}) - \bar{V}(s_t). \tag{2.24}$$

The δ_{td} is the value that will be used as parameter for updating the value function and/or the action policy. In this work, it is the value used to update the weights of the neural networks.

2.2.3 SARSA Algorithm

The name SARSA stands for State-Action-Reward-Sate-Action. It comes from a different form of updating the functions used by the agent. Instead of calculating the value using only the state as input, it calculates the value of a state-action pair, that is, the value of taking an specific action in a given state. This brings a different view to the problem since the agent now chooses which is the best action to take in a given state, not only which state is better to achieve.

As seen in Sutton and Barto (1998), the action will be selected by the action policy and only then the value of taking that action in this specific state will be estimated. Also, in order to estimate the next value so the target for update can be complete, the action that would be taken in the consecutive state needs to be defined using the same policy. The value of a state-action pair is denoted as $Q(s_t, a_t)$. In this case δ_{td} is given by:

$$\delta_{td} = r_{t+1} + \gamma \bar{Q}(s_{t+1}, a_{t+1}) - \bar{Q}(s_t, a_t) \tag{2.25}$$

and is used as in the non SARSA case for updating the neural networks.

2.2.4 Actor-Critic Method

The actor-critic is a TD method that has separated structures for the action policy, known as Actor, and the value function estimation, known as Critic. The Actor is the structure that defines the actions that will be taken and the critic is the structure responsible for evaluating the Actor's decisions. Notice that both the actor and the critic are part of the agent as none of them are part of the environment. Both elements learn to achieve the same goal, but represent different concepts.

Since it is the Critic's responsibility to evaluate the Actor's performance it is the Critic that defines the Actor's goal. The Critic's output is the TD error, using the value calculated with its own function and the reward obtained from the environment. This TD error is used for updating the Actor's action policy and the Critic's value function. The organization of the Actor-Critic Method can be seen in Figure 2.8.

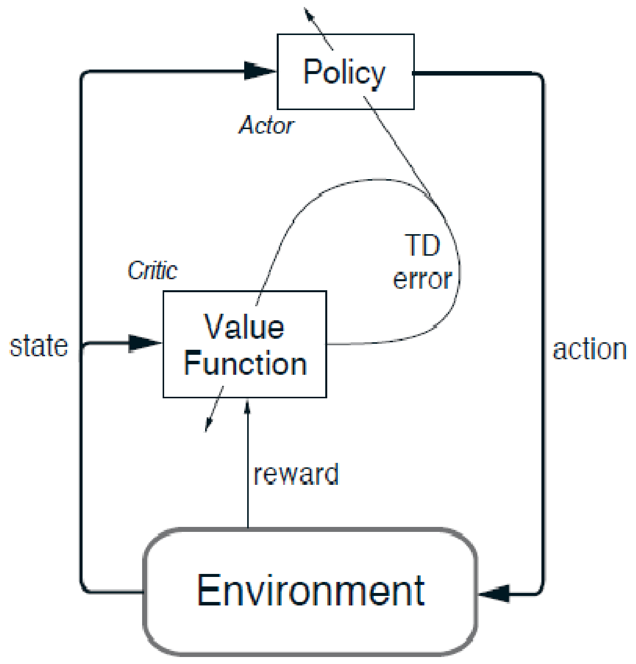


Figure 2.8: Actor-Critic method as seen in Sutton and Barto (1998).

Figure 2.9 shows the intended structure for the controller. The system model will be approximated by a Multilayer Perceptron and the process on deciding which network to use an with which techniques is describe in Chapter 4while the functions used by the Actor and Critic will be approximated by RBNs, which will be described in Chapter 5 alongside the usage of the δ_{td} . The following chapter describes the system that will be used as the environment for the RL problem.

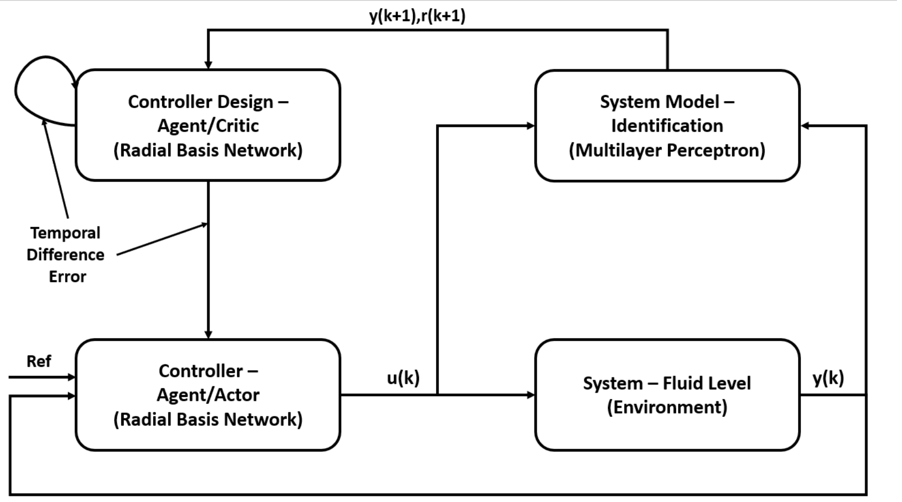


Figure 2.9: Controller structure.

3 EXPERIMENTAL ENVIRONMENT

A man who dares to waste one hour of time has not discovered the value of life. ***Charles Darwin (1809 - 1882)***

3.1 SYSTEM DESCRIPTION

The chosen system was instrumented by Bernardes et al. (2006) and consists in four aligned tanks with adjustable gate valves between each pair and a fluid reservoir. The bases of the four tanks measure 10cm in length and 6cm in width and while the first three tanks maintain this same section along their 50 cm height, the fourth tank has a trapezoidal form as seen in Figure 3.1. The First tank has two overflow openings at the top to avoid leakage and is fed with fluid by a pump, which is sealed and submerged in the reservoir in order to avoid overheating. To measure the fluid height there is a level sensor located in the fourth tank. The opening of the valves can be adjusted by a sliding gate that will increase or decrease the rift size to change the system dynamics if needed. A specific position was chosen for the most of the tests and experiments. Although the equations for the system's dynamics can be obtained, they are unnecessary since the premise of this work is to perform identification and control with no such information, so, regarding dynamics, the system will not be detailed further.

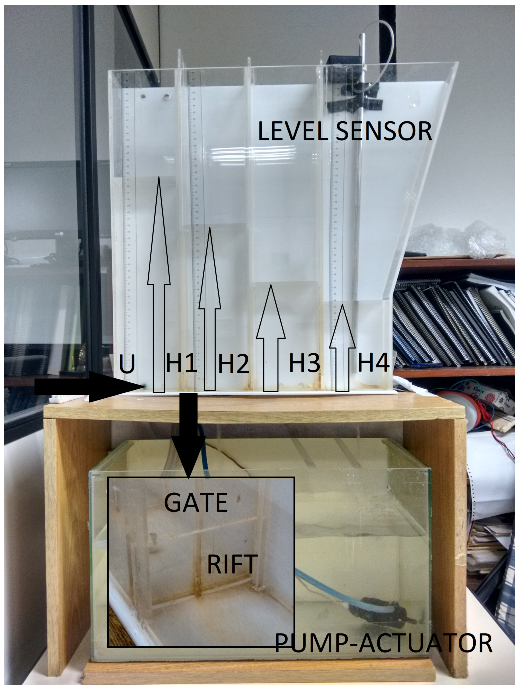


Figure 3.1: Four Tank System

The notation observed in Figure 3.1 has $u(k)$ as the input signal in discrete time k, and H_1, H_2, H_3 and H_4 as the level of each tank respectively. The use of capital H denotes that this is the level for large signals.

This system was chosen because it presents a great range of time-constants and it has various non-linearities such as dead-zone and saturation, which are interesting for testing the RL algorithm and for its remarkable delay. Theoretically the system presents another non-linearity that is square-root, and although this is a smooth non-linearity, it also enhances the difficulty of the control problem.

The system delay, dead-zone and saturation are illustrated respectively in Figures 3.2 to 3.4. Note that the non-linearities and delay were measured solely to justify the usage of the system, but information about them were not implemented on the controller. The dead-zone exists up to around 18% of the actuator's capacity while the saturation starts at around 90% of the actuator's capacity and the average delay is around 20 seconds. The dead-zone is settled since it is a feature of the actuator, but the delay and saturation might change once the valve configuration change, since they are both features of the system dynamics, which is dictated by the valves.

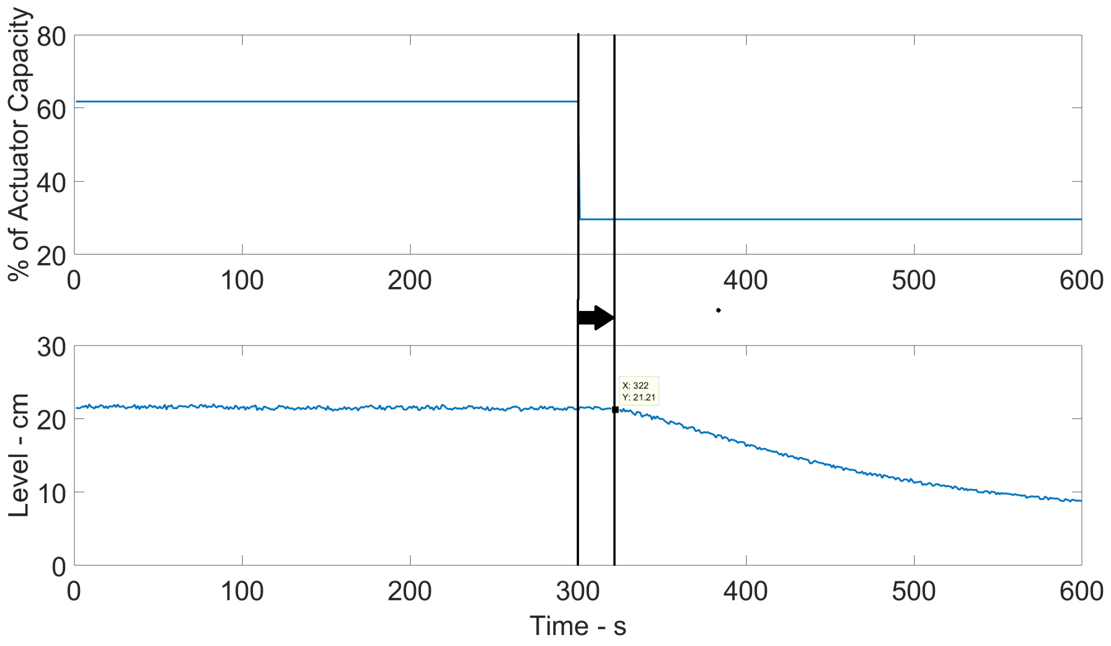


Figure 3.2: System Input-Output Delay

To perform experiments, collect data and send control signals, the setup seen in Figure 3.5 was assembled. The Pressure sensor is connected to the analog input of an Arduino MEGA 2560 which is then connected to a computer running the MatLab^{TM} software. The Arduino also generates a PWM signal that, after having passed through another circuit, is sent to the pump.

With information presented on the experimental environment, the methodology on the identification process and control task can be explained. The next two chapters will show respectively the procedure to achieve the approximation and control.

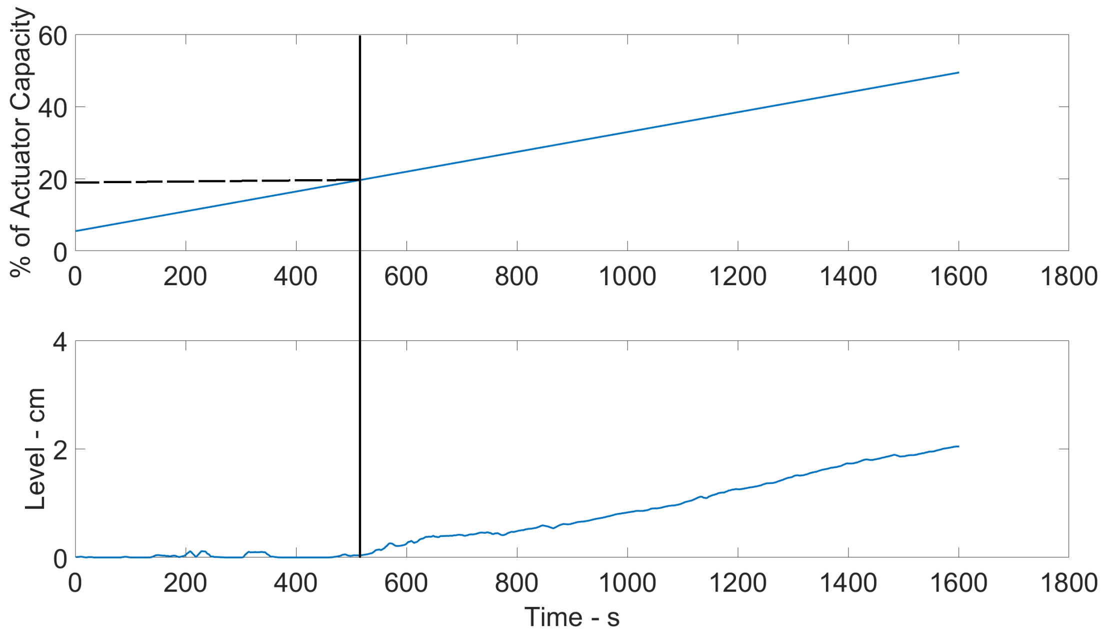


Figure 3.3: Dead Zone

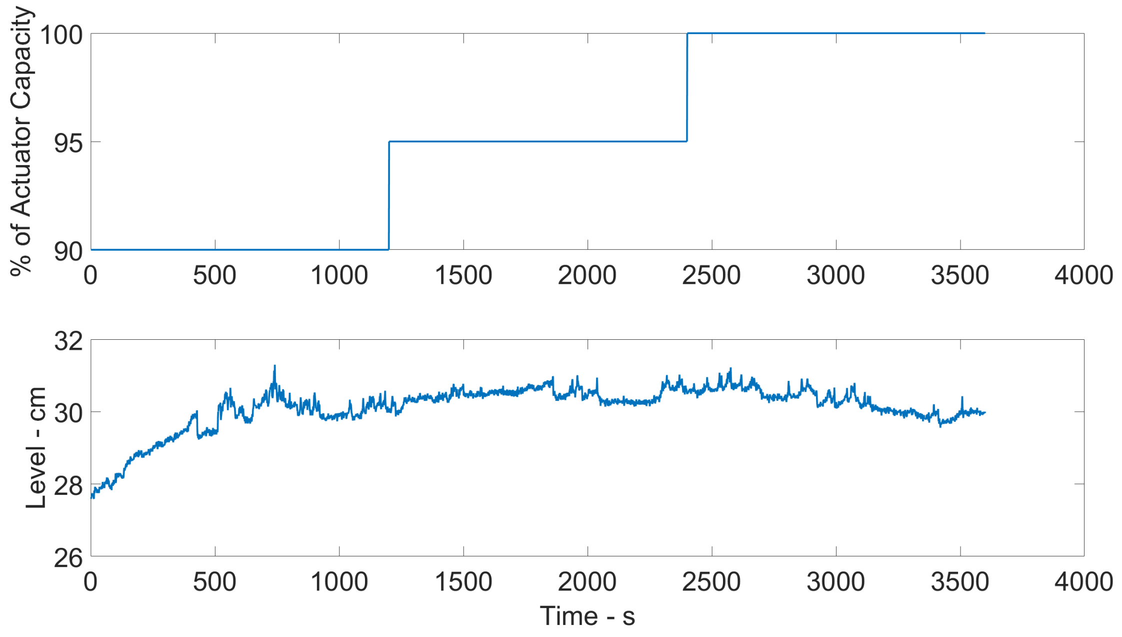


Figure 3.4: Saturation

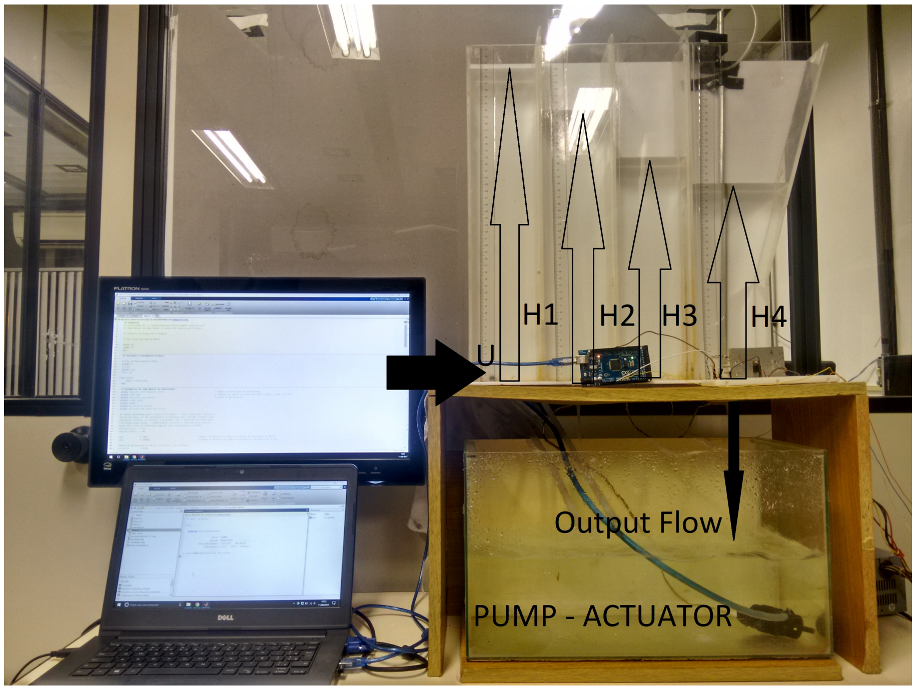


Figure 3.5: Full Control Process Layout

4 SYSTEM IDENTIFICATION WITH NEURAL NETWORKS

To me, mathematics, computer science and the arts are insanely related. They are all creative expressions. ***Sebastian Thrun (1967 - Present)***

4.1 PRIMARY DEFINITIONS

In this work, the task of identifying a system was needed in order to approximate a predicting model of the system. The predicting model was necessary because the system, as any real one, has delay, so it is virtually impossible to compute the immediate reward of the system using the next measured state, so the reward of a given action, or behavior, should be given by the estimated future state, bypassing the delay.

One of the main functions of an ANN is to map a set of inputs to a set of outputs, thus, an ANN has the identification task as part of its essence, also, ANNs are black box methods for online identification since in opposition to classic methods it does not need a model with physically significant parameters, but solely input and output data which is a premise of this work. Moreover, identifying only the input to output correlation makes it easier to use the ANN as a predictor, since correlating past to recent data will not need any physical significance.

In this work, identification by classical methods, as approximation to linear models, such as ARX and ARMAX, or nonlinear models, such as Hammerstein and Wiener models, was performed in order to compare the efficiency and applicability of the proposed neural network method, even though in this work these classical methods were not used for the control problem, only for the comparison for the identification problem.

Initially, the identification was made by batch training with 16 to 17 hours of input/output data. For the classic models, the average delay of the system was set as 20 seconds, based on measurements seen in Figure 4.1 where the measured delay for each transition is depicted as a black circle and the average value of those is just below 20 seconds. for the neural network methods, the output used for the correction of the network was the output 20 seconds in the "future" so the delay was bypassed.

After the offline training, the best ANNs were tested online in two different manners; the ANN acquired after the batch training with further online training, and the ANN layout without the batch training, only with the online training. This was made to see which model/network layout suits the problem better regarding precision and learning rate. The main objective is to achieve acceptable fitness to the validation data with less time as possible because this mode has to serve the control problem, that is, has to show results as fast or faster than the controller, and should be as precise as possible. The mathematical steps for each learning algorithm and the used layouts are explained below.

In every case the normalized root mean squared error (NRMSE) which is shown in Equation 4.1 was used as parameter to measure fit between data. The value of fit between data using this method varies from $-\infty$ to 1, with 1 being the perfect fit and $-\infty$ the worst possible fit. The results were expressed in form of percentage.

$$NRMSE = \frac{\|xref - x\|}{\|xref - mean(xref)\|} \tag{4.1}$$

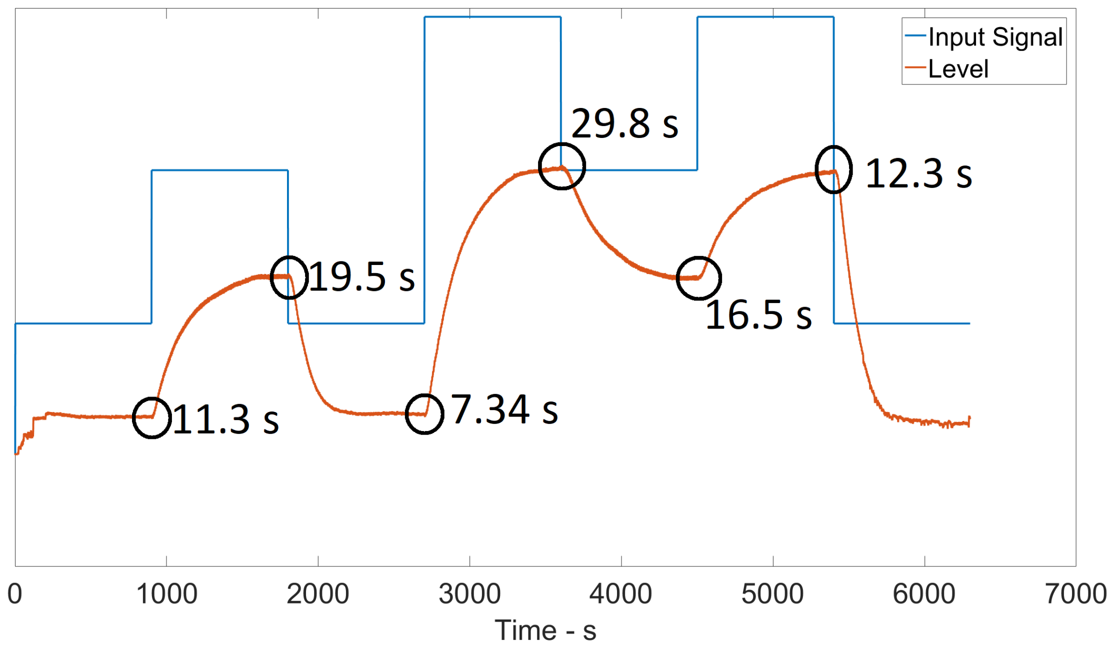


Figure 4.1: Delay Measurement

where $xref$ is the reference data, x the input data and || indicates the 2-norm of a vector. The fit is calculated as seen below:

$$fit = 1 - NRMSE. \tag{4.2}$$

4.2 CLASSIC MODELS

Before performing the identification process with neural networks, the chosen system was identified with classical methods for comparison. The chosen methods is by approximating the system to given models, such as ARX, ARMAX and Hammerstein and Wiener Models. All the models were approximated using the 17 hour Data Set and the Identification Toolbox of MatLab^{TM}.

ARX stands for *Autoregressive with Exogenous Inputs Model* and defines that the model output depends on a linear combination of previous outputs (autoregressive), current and previous input values (exogenous) and a stochastic value to represent noise. The generic expression for an ARX model can be seen below,

$$A(q)y(k) = B(q)u(k - \tau_d) + v(k)$$

where the polynomial $A(q)$ and $B(q)$ are given by

$$\begin{aligned} A(q) &= 1 - a_1 q^{-1} - ... - a_{n_y} q^{-n_y} \\ B(q) &= b_1 q^{-1} + ... + b_{n_u} q^{-n_u} \end{aligned}$$

being q^{-1} a delay operator so that, $y(k)q^{-1} = y(k-1)$, n_y is the number of regressive terms, n_u the number of previous input terms, τ_d the system's delay and $v(k)$ white noise.

ARMAX stands for *Autoregressive Moving-Average with Exogenous Inputs Model*, and differs from the ARX model because it also uses previous values of the noise to estimate it alongside. The generic expression for the ARMAX model is given by,

$$A(q)y(k) = B(q)u(k-\tau_d) + C(q)v(k)$$

where the polynomial $C(q)$ is given by

$$C(k) = 1 + c_1 q^{-1} + ... + c_{nv} q^{-n_v}$$

The numbers n_y, n_u, and $n_v$were varied in the range of 1 to 10 and the average system delay was chosen to be 15 seconds based on measurements on different open loops tests of the system, as said and seen in Figure 4.1. For the sake of simplicity, only the best fitting option of each model was used for the comparison.

The non-linear classical methods used were the *Hammerstein and Wiener Models*. They consist in taking a linear model, such as the ARX model and put it in cascade with a non-linear function, either before the model (Hammerstein), after the model (Wiener) or in some cases, both. Figure 4.2 shows the structure for the mentioned models.

The expressions for the Hammerstein and Wiener Models, with an ARX for the linear part, can be written respectively as,

$$y(k) = a_1 y(k-1) + ... + a_{n_y} y(k-n_y) + f.[u(k-1-\tau_d) + .. + u(k-n_u-\tau_d)]$$

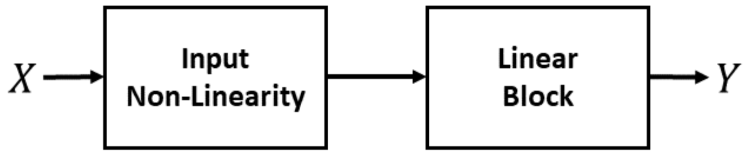
and

$$y(k) = f.[a_1 y(k-1) + ... + a_{n_y} y(k-n_y) + b_1 u(k-1-\tau_d) + ... + b_{n_u} u(k-n_u-\tau_d)]$$

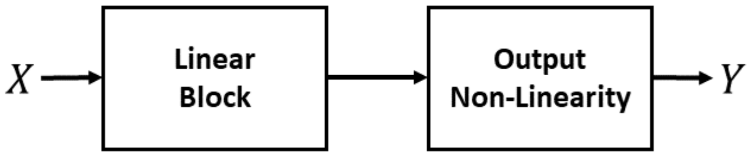
as seen in Aguirre (2015), where τ_d is the delay. The linear model chosen to be used with the Hammerstein and Wiener Models was the best fitting ARX obtained, also, the models were tested with some non-linear functions available in the Identification Toolbox of MatLab^R, and just as for the linear models, only the best fitting model was used for the comparison.

4.3 MULTILAYER PERCEPTRON

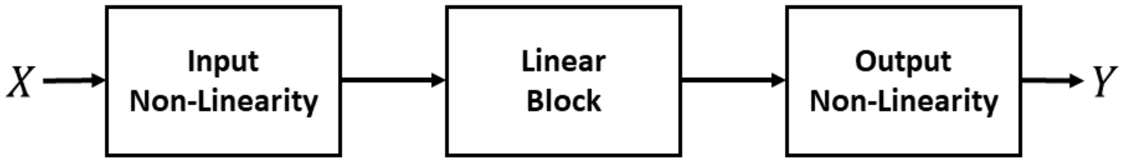
Different layouts of MLPs were used in this task. With the number of neuron in each layer varying from 5 to 100, one hidden layer and two hidden layer MLP were tested and, as the linear models, only the best layout



(a) Hammerstein model.



(b) Wiener model.



(c) Composed non-linear model.

Figure 4.2: Hammerstein & Wiener non-linear models.

of each was selected for usage in the control process and in the comparison. The vector used as input was composed solely by the past output and input of τ_d iterations before, $[y(k-\tau_d), u(k-\tau_d)]$. This was made to test how the MLP handles the identification process given that no model was identified before, therefore, there is no information about what terms should be useful in the task, and no information whatsoever about the system except for the average delay, which is a premise of this work.

The activation function used in each neuron was the logistic function, seen in Equation 4.3, where k defines the inclinations of the curve, $x_0$is the center value in the x-axis and L is the maximum value of the curve. The learning rate was set to 0.01, and the weights were started randomly with values between -0.5 and 0.5. In this work, L and k where set as 1 and x_0 was set as 0.

$$\varphi(x) = \frac{L}{1+e^{-k(x-x_0)}}. \tag{4.3}$$

Using Equation 4.3 on Equation 2.2 yields the mathematical expressions for the neurons output on the MLP,

$$y_j^l(k) = \frac{1}{1+e^{-\left(s_j^l(k)\right)}}. \tag{4.4}$$

Also it is important to know the derivative of the logistic function since it is needed in the weight correction steps. Being $\varphi(x) = \frac{L}{1+e^{-k(x-x_0)}}$ the derivative of the function is given by,

$$\varphi^{'}(x) = \varphi(x)\left(1-\varphi(x)\right). \tag{4.5}$$

Using Equation 4.5 in Equation 2.12 yields the expression for the weight update used for the output layer of the MLP,

$$\Delta w_{ji}^{l}(k) = \eta e_j^l(k)\varphi_j\left(s_j^l(k)\right)\left(1-\varphi_j\left(s_j^l(k)\right)\right)y_i^{l-1}(k). \tag{4.6}$$

For the hidden layers, using Equation 4.5 in Equation 2.17 yields,

$$\Delta w_{ji}^{l}(k) = \eta\varphi_j\left(s_j^l(k)\right)\left(1-\varphi_j\left(s_j^l(k)\right)\right)\left(\sum_{n=0}^{M} e_n^{l+1}(k)\varphi_n\left(s_n^{l+1}(k)\right)\left(1-\varphi_n\left(s_n^{l+1}(k)\right)\right)w_{nj}^{l+1}(k)\right)y_i^{l-1}(k). \tag{4.7}$$

The MLP output is calculated using Equations 2.1 and 2.2 with the activation function being the one seen in Equation 4.4 in a feedfoward manner which is the flow of information as seen in Figures 2.2 and 2.5. Once the output is calculated and then compared to the measured value to generate $e(k)$, the error is propagated backwards recursively using Equations 4.6 and 4.7.

4.4 RBN WITH BACKPROPAGATION ONLY FOR THE WEIGHTS

The RBN used in this case was made with Gaussian activation functions, seen in Equation 2.18 and the weights were started randomly with values between -0.5 and 0.5. The variance of the Gaussian functions was fixed at 0.2 and the centers were started both randomly and uniformly distributed. The input vector is the same as the used for the MLP and it was used for all the RBNs tested.

Since the Gaussian differs from the Logistic function, the expressions used for the weight updates is not the same as the on used for the MLP. The output neuron of the RBN was chosen to be a linear neuron, meaning that its differential equals 1 and thus, the local gradient equals the instant output error, so the equation for updating the weights become

$$\Delta w_i(k) = \eta e(k)\rho(||x-\mu_i||). \tag{4.8}$$

In this case, the centers of the network were fixed during the identification, even when started randomly.

4.5 RBN WITH BACKPROPAGATION FOR THE WEIGHTS AND CENTERS

In this case, the same networks used in the previous case were tested, but along the identification process, the centers of the RBFs were also updated meaning that the number of parameters to be accounted for when minimizing the cost function doubled.

To use the backpropagation algorithm to update the center parameters, the differentiation of the activation function must be made as a function of the center instead of the input of the neuron. In the case of the Gaussian function used in this work, the derivative as function of the center is given by,

$$\frac{\partial e^{-(\frac{||x-\mu||^2}{\sigma^2})}}{\partial \mu} = \frac{2(x-\mu)}{\sigma^2} e^{-(\frac{||x-\mu||^2}{\sigma^2})}. \tag{4.9}$$

Using Equation 4.9 in Equation 2.17 results in the expression for updating the centers with backpropagation being,

$$\Delta\mu_i(k) = \eta e(k) w_i(k) \frac{2(x-\mu_i)}{\sigma^2} \left(e^{-(\frac{||x-\mu_i||^2}{\sigma^2})} \right) \rho(||x-\mu_i||). \tag{4.10}$$

The weights were also updated with Equation 4.8.

4.6 RBN WITH CLUSTERING FOR THE CENTERS

In this case, the same RBNs used with the Backpropagation algorithm were used having the centers of the activation functions updated via a Clustering algorithm.

In order to measure similarity between cluster and data, it suffices to determine a parameter of similarity that must be respected. The chosen parameter (and the most commonly used in literature), was the euclidean distance between data and center. The expression for the euclidean distance is given by:

$$d(X,\mu) = \sqrt{(X_1-\mu_1)^2 + (X_2-\mu_2)^2 + ... + (X_n-\mu_n)^2} \tag{4.11}$$

with n being the size of the input vector and therefore, the size of the center vector.

Since all values were normalized between 0 and 1, the clusters were defined with values between 0 and 1, and so, the greatest distance that could be achieved is $\sqrt{n}$. The number of clusters accepted by the controller varied from 10 to 100 and the parameter of similarity in each test was $\frac{\sqrt{n}}{C}$, with C being the number of clusters. This parameter was used as the maximum value the euclidean distance between cluster and data could be for the new data to be assigned to that cluster. If the distance from all already defined clusters was greater then the parameter value and the number of clusters defined was not yet the permitted number a new cluster was created to receive this new data, but if the number of clusters reached the maximum value, the data would then be placed in the cluster from which it had the smaller distance.

The cluster's center values could be updated or not. If they were not updated a new cluster center would equal the very first input vector that had the distance greater then the defined parameter to all the already defined clusters. If updated, the cluster center values would be calculated with the recursive average of the input vectors assigned for that cluster, using the following expression,

$$\mu(k) = \frac{c\mu(k-1) + X(k)}{c+1}$$

with c being the number of input vectors already assigned for that cluster. This method is known as k-means clusters.

Another way to use the cluster algorithm is to have the values of the clusters defined previously, so, instead of creating new clusters when needed, the number of clusters was already defined.

5 SOLVING THE CONTROL PROBLEM WITH REINFORCEMENT LEARNING

Equipped with his five senses, man explores the universe around him and calls the adventure Science. ***Edwin Powell Hubble (1889 - 1953)***

5.1 THE CONTROLLER STRUCTURE

The control problem consists, in simple words, in comparing the desired situation of a system with the current one and act to match both of them. As described in subsection 2.2.1, the elements needed to describe the RL problem are, the agent, the environment, the state, an action policy, actions, a reward function, a value function and the goal. Also, as described in subsection 2.2.4, the agent is divided in two different elements, the actor and the critic Lin (2011).

As said before, the agent is the learning element which will use its experience to choose what action to take, hence, the controller itself. The environment is the four tank system and the level sensor. The sensor is also part of the environment because it is the part responsible for identifying the state where the agent is and to reward or punish the agent.

The state of the RL is given solely by the reference signal and the measured fluid level, that is, it shows where the agent is and where it should go. In fact, the state could be any two of the following three: the measured output, the reference signal and the error signal, since every one can be inferred from the other two. Figure 5.1 shows the controller structure using the RL for design. All blocks except for the system are functions approximated by neural networks.

5.1.1 Using RBN for Action Policy and Calculating State Value.

The action policy, which is also the actor part of the agent, and the value function were approximated by RBNs with Gaussian functions. From Figure 2.5 it can be noticed that the equation that define the RBN output is as follows:

$$Y = \sum_{i=1}^{N} w_i \rho(||X - \mu_i||) \tag{5.1}$$

where N is the number of neurons in the hidden layer, X is the input vector and μ the center vector for that neuron, Orr (1996).

The output of the RBN used for the action policy is the action itself which is the control signal sent to the system. Since the controller uses the SARSA method, the output of the RBN used for the value function depends on the state-action pair and it is used by the Critic to calculateδ_{td} which will be used to train both networks. Using the δ_{td} as the target error for the updating means that δ_{td} is the cost function to be minimized, so the gradient descent method was applied to the squared δ_{td} as seen in Wang et al. (2007) and, using Equation 4.8, the following expression for updating the weights was used,

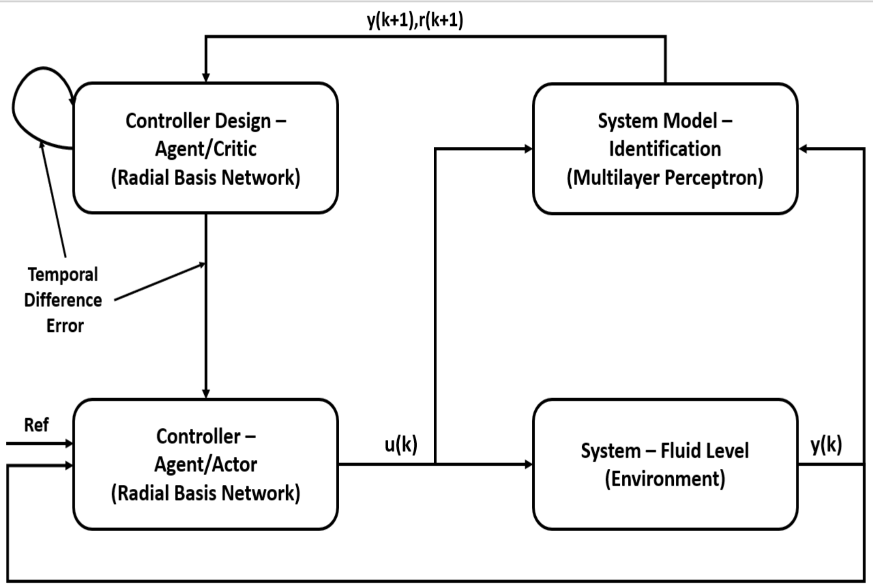

y(k+1),r(k+1)
Controller Design – Agent/Critic (Radial Basis Network)
System Model – Identification (Multilayer Perceptron)
Temporal Difference Error
Ref
Controller – Agent/Actor (Radial Basis Network)
u(k)
System – Fluid Level (Environment)
y(k)

Figure 5.1: RL applied for control.

$$\Delta w_j(k) = \eta \delta_{td} A \rho(||X - \mu_j||) \tag{5.2}$$

where η is the learning rate for the weights of the neural network, and A is a measure of the controller behavior, that is, it shows how the controller output changed. If the controller output grew, then A is positive, else, A is negative. This is necessary because, as mentioned in Chapter 2, the reward is a response to *behavior*, and not to the action itself, so the updating of the network needs to address the *change* in the controller output.

The centers were updated with the back propagation algorithm because it showed better performance in comparison with the cluster algorithm, the difference will be seen in the following chapter.

5.1.2 The Reward Function

There were four different reward functions used for the chosen network. During the rest of this document they will be addressed as *RF1, RF2, RF3 and RF4.* RF1 was the simplest. It returned a negative value α if the the output level was going further from the reference value and 0 if the output level was getting closer.

RF2 used the same logic with a “discrete” reward, but using a tolerance margin as well so if the level output was not inside the tolerance margin defined by a value ε, the function wold return a punishment of value β in addition to the previously defined α punishment.

RF3 was made in a “continuous” manner and the value of the punishment was proportional to the output error. It is important to notice that even though the output error is being uses, this is *not* supervised learning. It does make sense to idealize proportional rewards though, because it shows how good an action was, which is foreseen in RL theory. RF3 was defined as seen in Equation 5.3.

$$r_k = -\,|e(k)|. \tag{5.3}$$

Finally, RF4 was defined with the addition of the error variation, working analogously to a derivative in order to decrease overshot. The module of the variation of the error on the output level would be summed in the reward, as seen in Equation 5.4.

$$r_k = -\,|e(k)| - (|e(k)| - |e(k-1)|) = -2|e(k)| + |e(k+1)|. \tag{5.4}$$

There should be little difference between the responses of the controllers using RF3 and RF4 because the idea of a varying reward function is already implicit in RF3 and also, the variation on the output error is very small because the controlled system is very slow.

5.1.3 Algorithm for Controlling a Fourth order Fluid Level system with Radial Basis Networks and Reinforcement Learning.

With all the elements and functions defined, the algorithm can be completed. It begins with the initialization section. The first step is to initialize the system itself, that is, to measure the current fluid levels, define the state and define all the parameters that will be used during the process. The parameters include the learning rates, the error margin, the reward penalties and etc. From the definition of the SARSA method, the next step is to choose an action using the current action policy, which in this case will be the output of the actor's RBN. With the action defined, the value $Q(s_k, a_k)$ can be initialized.

Following the initialization, the iterative part of the algorithm develops. The action chosen by the actor is taken and will generate a change in the system's state, but since the system's change will be delayed and in order for the RL to work it needs an immediate reward, the next state s_{k+1} will be predicted by the model being identified. The predicted future state will be used for calculating the reward r_{k+1}, the action a_{k+1} and finally, with the predicted state and action at hand, the predicted value $Q(s_{k+1}, a_{k+1})$ can be calculated. Using the reward r_{k+1}, Q_k and Q_{k+1}, the δ_{td} can also be calculated and used to update the networks used by the actor and the critic.

Usually, the new state and action are defined as the current ones before the loop finishes, but since the new state and actions are actually predictions, the next state should be defined by a new measure of the system's output and from this new measure, a new action should be defined by the actor. Once the action is defined, the procedure of the iterative part can be repeated as described in Algorithm 1.

Algorithm 1 - Control of a fourth order fluid level system with reinforcement learning and radial basis neural networks.

1. Initialize s_k and all the constant parameters to be used by the controller.
2. Initialize $a_k$using the RBN defined for the Actor.
3. Initialize Q_k using the RBN defined for the Value Function.
4. Take action a_k on the model to predict the next state s_{k+1}.

5. Observe the reward r_{k+1} using RF1, RF2, RF3 or RF4.
6. Calculate a_{k+1} using the RBN defined for the Actor.
7. Calculate Q_{k+1} using the RBN defined for the Value Function.
8. Calculate δ_{td} using equation 2.24.
9. Train both RBNs using the δ_{td} and Equations 4.8 and 4.10 or using clusters.
10. Optional - Update the system model. (The system model can be used without online training, just the batch.)
11. Perform a new measurement of system output, define a new s_k and following, a new a_k.
12. Repeat steps 4 to 12.

The identification process was taken as a basis for choosing the ANN to be used in each task of the controller, also, the algorithm was tested with the various forms of RBN training described, and with the four different reward functions. In order to avail the sensibility of the controller to some situations, parameters such as sample time, learning rate and initial conditions of the neural networks were varied. The information collected from such experiments was used to define the parameters of a controller to be tested against a PI for comparison.

6 EXPERIMENTAL RESULTS

The good thing about science is that it's true whether or not you believe in it. ***Neil deGrasse Tyson (1958 - Present)***

6.1 SYSTEM IDENTIFICATION

As described in Chapter 4, the identification process was made first with classical models. Figures 6.1, 6.2 and 6.3 show the response of each best fitting classical model chosen for comparison. Figures 6.1 and 6.2 show respectively the linear ARX and ARMAX models, while figure 6.3 shows the results for Hammerstein and Wiener model. These figures show the model output to the validation input in comparison to the measured validation output. The Fit of each model expressed in percentage calculated with the NRMSE parameter with Equation 4.2 are shown in table 6.1. As said previously, the fit achieved by this method varies from $-\infty$ to 100%, with 100% being the perfect fit and $-\infty$ the worst possible fit.

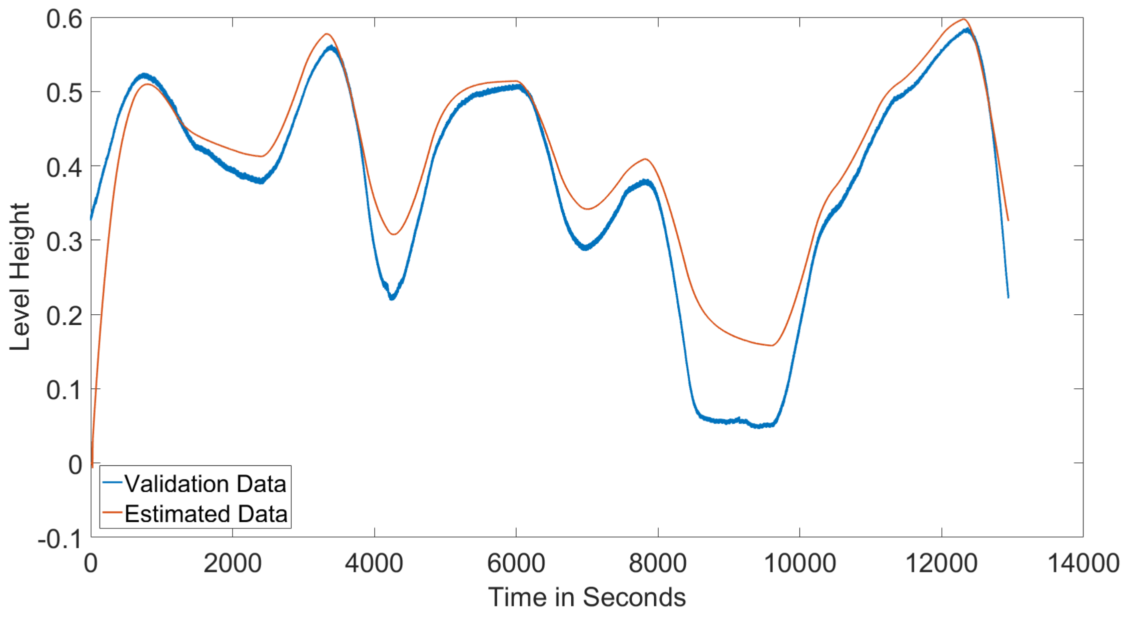


Figure 6.1: ARX model in comparison to validation data.

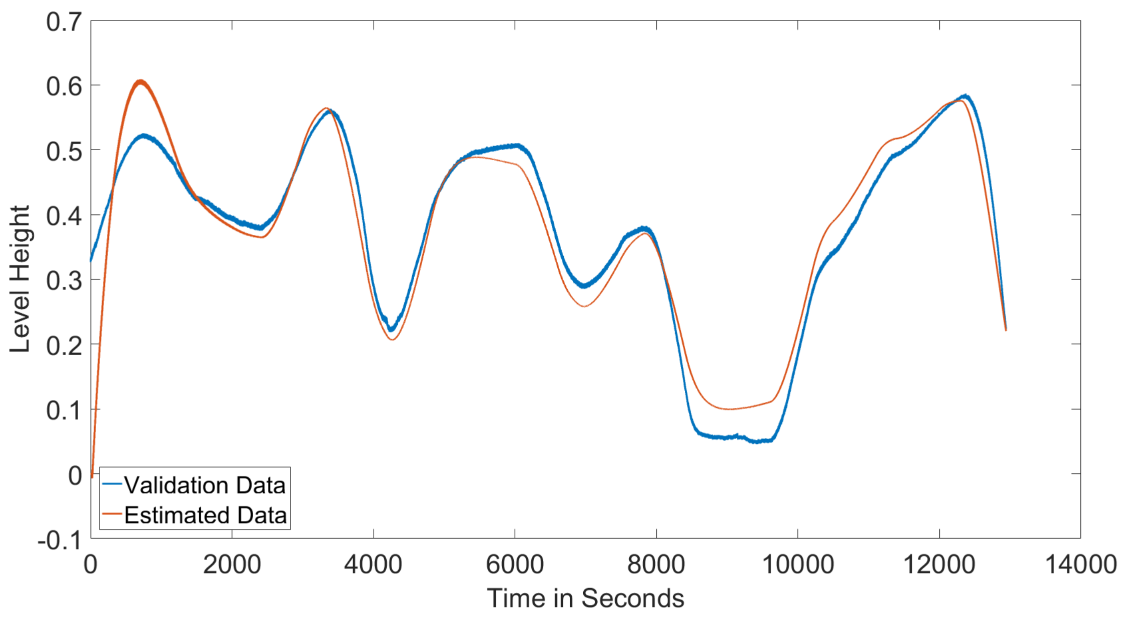


Figure 6.2: ARMAX model in comparison to validation data.

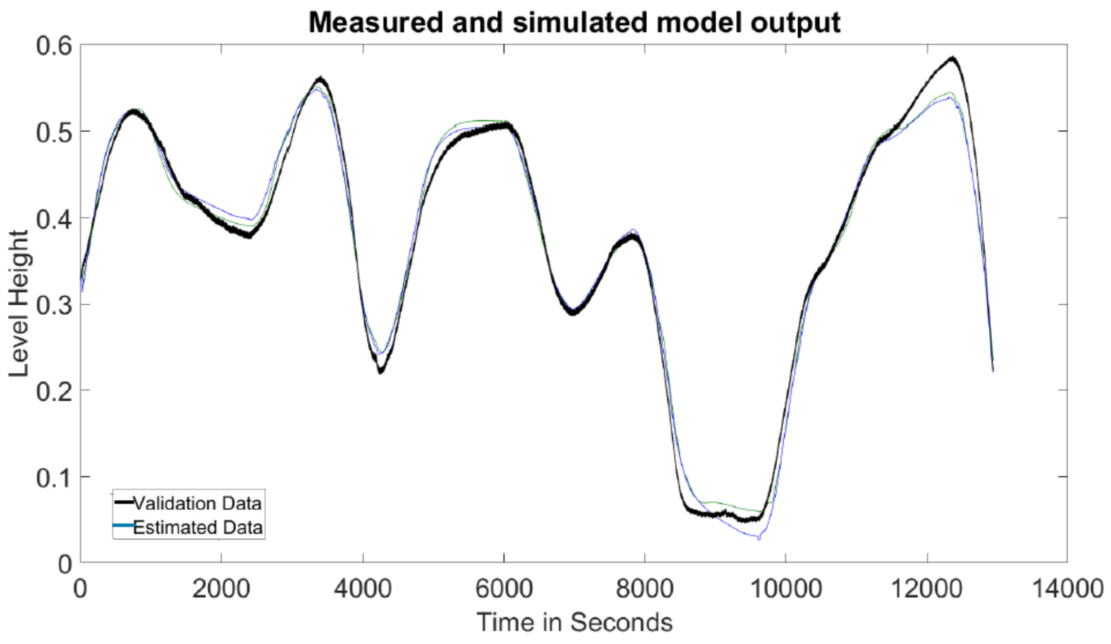


Figure 6.3: Hammerstein & Wiener model in comparison to validation data.

Table 6.1: Fit of each classical model

Model	Fit
ARX	48%
ARMAX	58%
Hammerstein and Wiener	85%

Table 6.2 shows the best fitting model for each layout/learning algorithm used for the identification part as well as the highest fit to the validation data achieved during training. Each model was tested online on the real system with and without previous training.

Table 6.2: Best fitting model of each network layout/learning algorithm with batch training.

Network Layout/Learning Algorithm	Number of Neuron in Hidden Layers	Number of Epochs to Achieve Max Fit	Max Fit
MLP with 2 Hidden Layers	90-15	37	92.55%
MLP with 1 Hidden Layer	5	100	89.67%
RBN with Fixed Centers Started Randomly	30	50	93.05%
RBN with Fixed Centers Started Uniformly	16	43	96.82%
RBN With Randomly Started Centers Using Backpropagation	20	26	93.53%
RBN With Uniformly Started Centers Using Backpropagation	9	18	94.35%
RBN With Randomly Started Centers Using Clusters	30	22	93.88%
RBN With Uniformly Started Centers Using Clusters	36	50	95.51%
RBN with Online Creating Centers but no Center Correction	15	35	67%
RBN with Online Creating Centers with Center Correction	15	43	93.28%

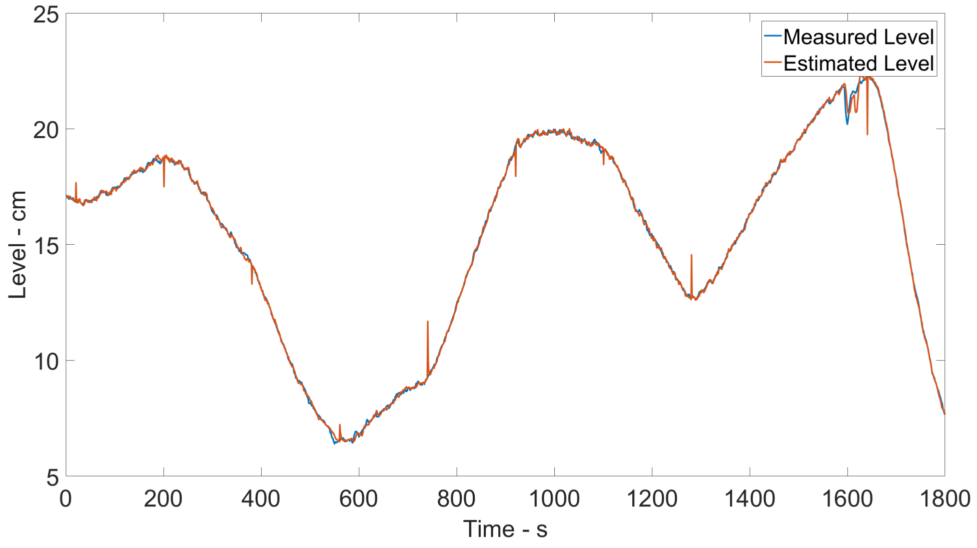
Figures 6.4 to 6.13 show, in the same order as seen in Table 6.2 the online performance of each ANN whit and without previous learning from the batch training.

Regarding the performance with previous knowledge from the batch training, the MLP are shown to be much more stable, which was expected. Since the MLP with two hidden layer that performed best was the biggest network among the best fits, it was also expected that after enough training it would not only be the most stable network but the network that would interpret better the system, being able to identify more characteristics of the system depicting less spikes than the other networks and tracking the system's response more efficiently. Yet, it is interesting to notice that in general, apart from the MLP with 2 hidden layers, the smaller networks responded better and that is notable from data in table 6.2 that the RBNs train a lot faster then the MLPs. It is also important to notice that the Classical models became irrelevant in face of the Fits achieved by the neural networks.

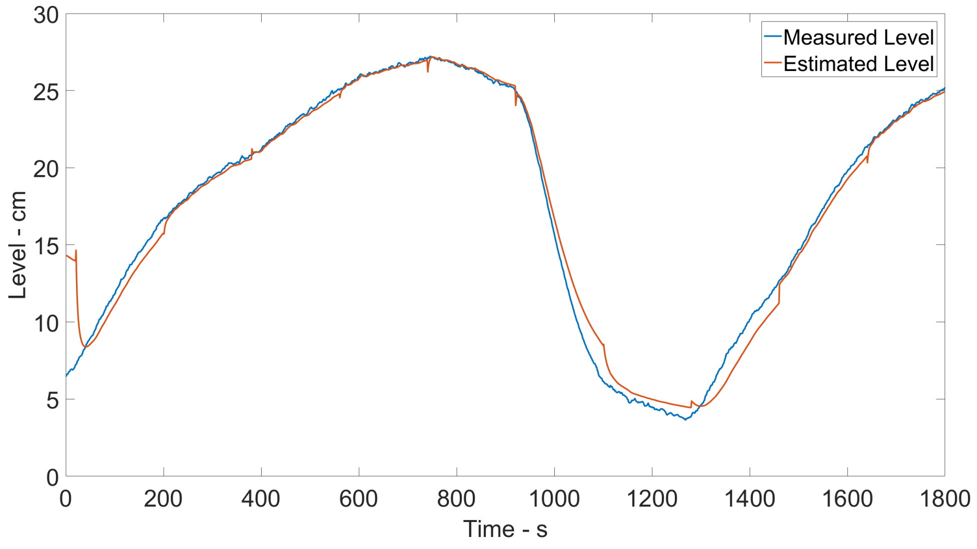
Regarding the performance without previous knowledge, the MLP with two hidden layers showed unexpected result, because since it is the larger notwork, it should take the most time to learn, but it can be seen that it is very efficient to track the system response, although the single time it looses track, it takes a lot of time to find it back, in opposition to almost all RBNs which, even depicting some spikes, when the input signal changes, take very short time to find its way back to the right response.

With this information the decision of which network will be used in which task can be made. The MLP was chosen for system identification, since it demonstrates more stability and the RBN with fixed centers was chosen to perform the actor and the critic of the controller, since it shows the fastest recovery when loosing track of the function being approximated.

Figure 6.4 presents the online approximation result for the MLP with two hidden layers. The networks performance with previous training shows remarkable stability with less and smaller spikes than the other networks. Surprisingly the performance with no offline training is also one of the best results. This is unexpected because the MLP with two hidden layer is the biggest network and should take the longest time to learn.



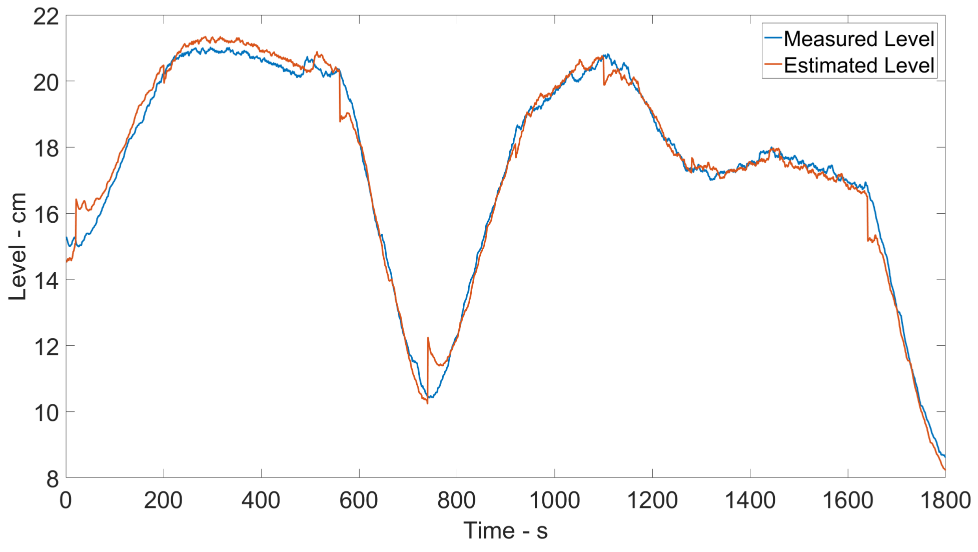
(a) With previous learning



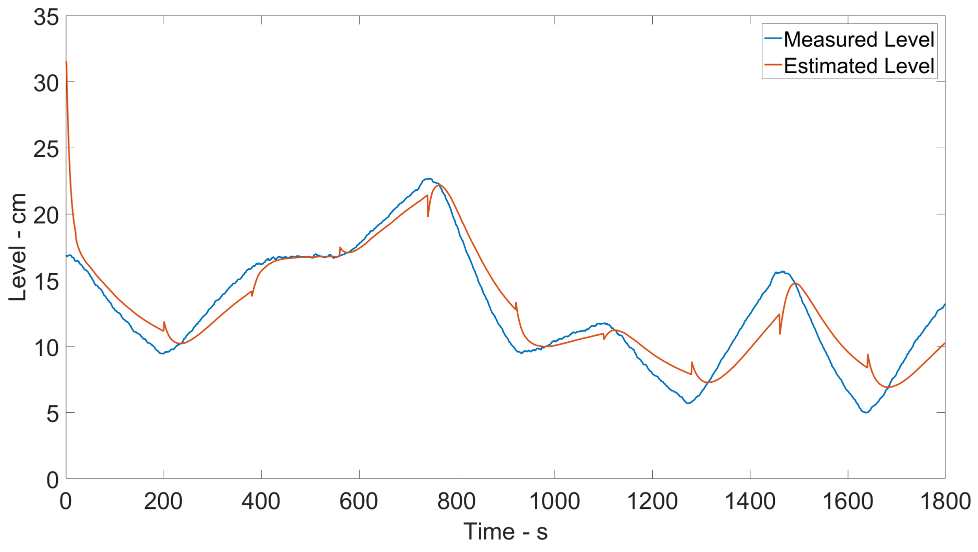
(b) Without previous learning

Figure 6.4: MLP with two hidden layers.

Figure 6.5 shows the online approximation result for the MLP with one hidden layer. The network performed during 30 minutes with and without previous training from the batch data. The networks performance with previous training is good and relatively stable if compared to the other networks but when tested with no previous training it shows a very slow adaptation to the system's real response, which is expected because as seen in Table 6.2, it is the network that takes the most epochs to reach max Fit.



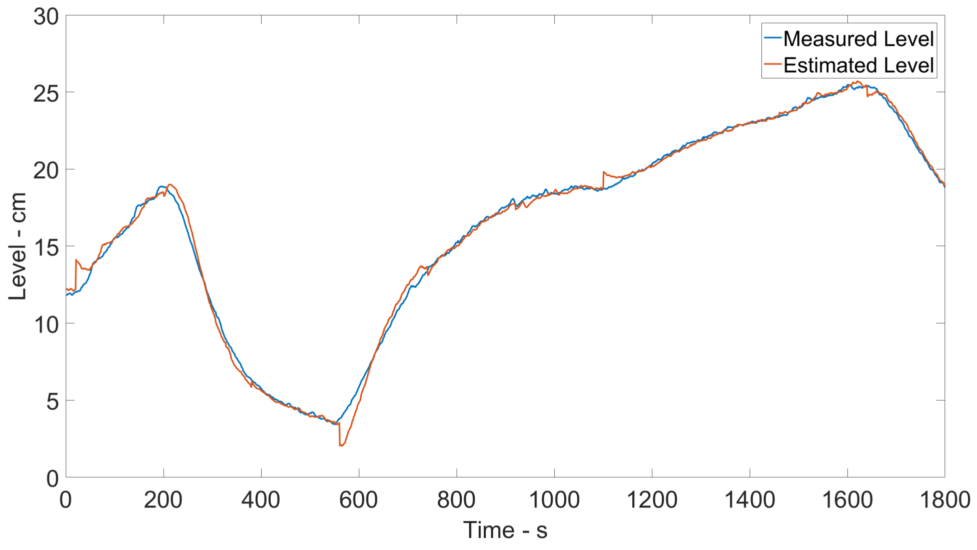
(a) With previous learning



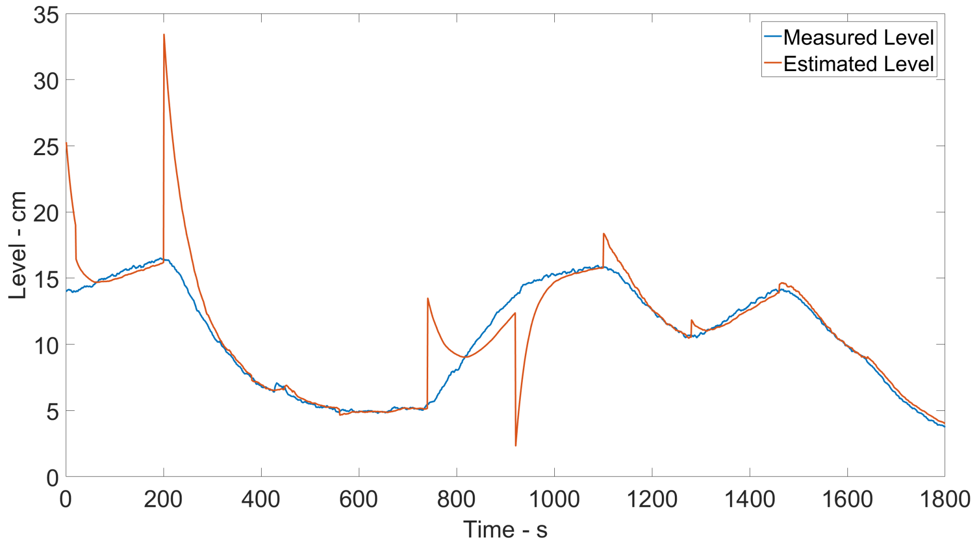
(b) Without previous learning

Figure 6.5: MLP with one hidden layer.

Figure 6.6 shows the online performance of the RBN that had the weights updated with the backpropagation algorithm described in Chapter 2 and 4 and had the centers started randomly and fixed during the experiment. The network shows good stability and high Fit to the data when performing with previous training but, as every other RBN, with no previous training is depicts big spikes in the output, although, can go back to track the real value way faster then the MLPs tested.



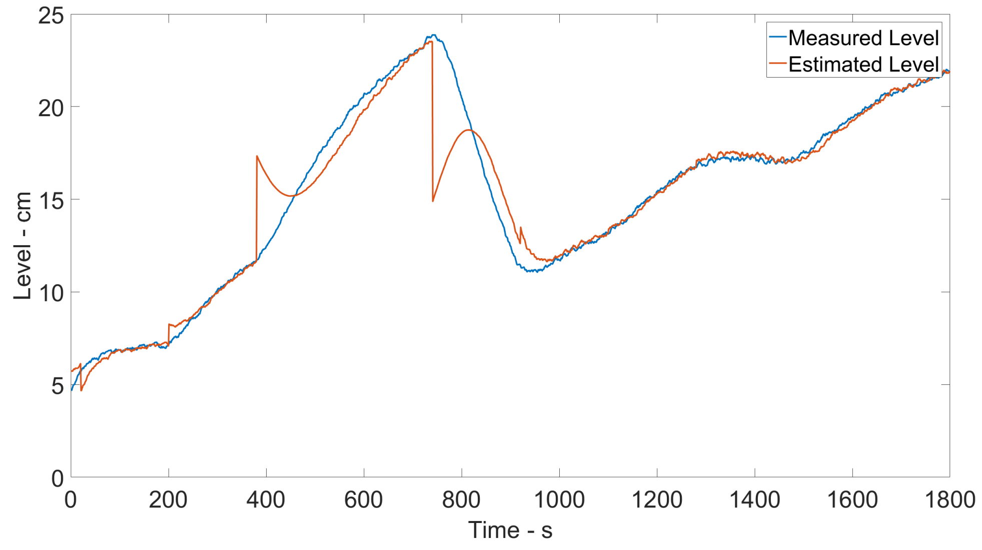
(a) With previous learning.



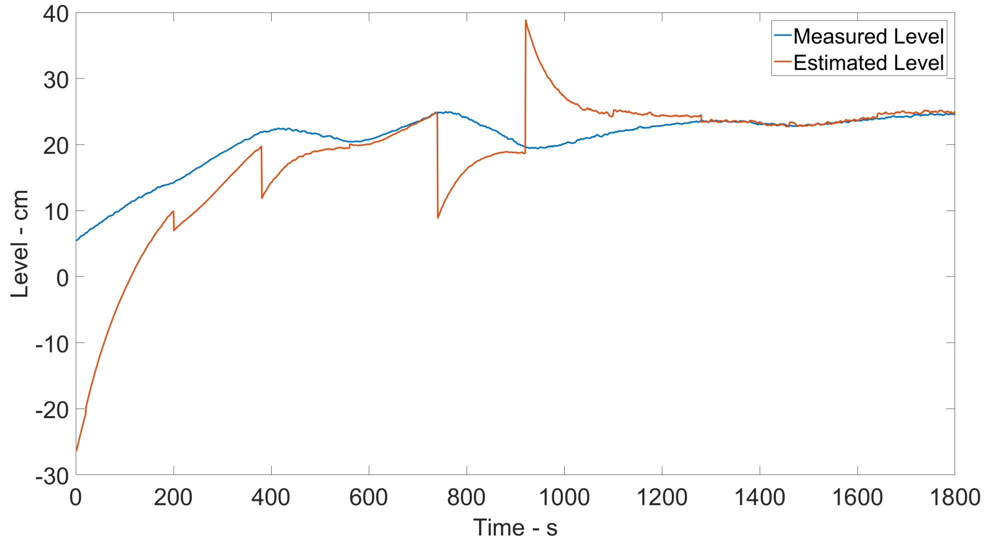
(b) Without previous learning.

Figure 6.6: RBF with fixed centers randomly started.

Figure 6.7 shows the results for the RBN that had the centers started uniformly distributed and fixed during the experiment while the weights were updated via the backpropagation algorithm.



(a) With previous learning.

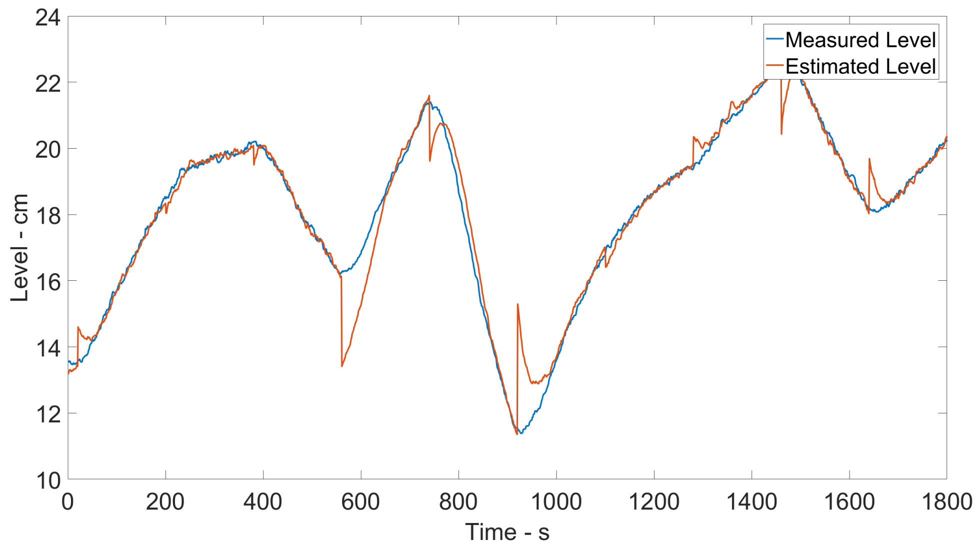


(b) Without previous learning.

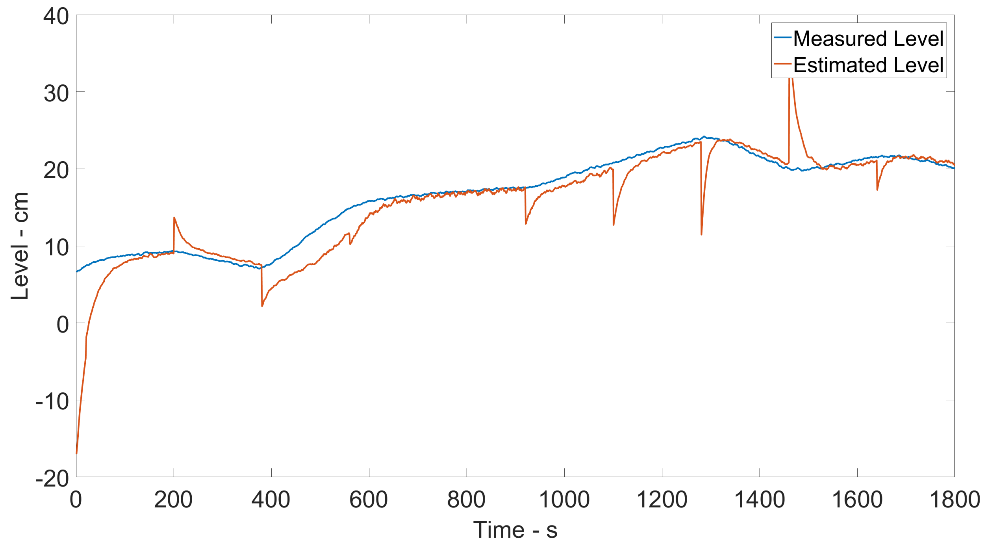
Figure 6.7: RBF with fixed centers uniformly started.

The results for the RBN that had the centers started randomly and had both centers and weights updated with the backpropagation algorithm can be seen in Figure 6.8.

6.8



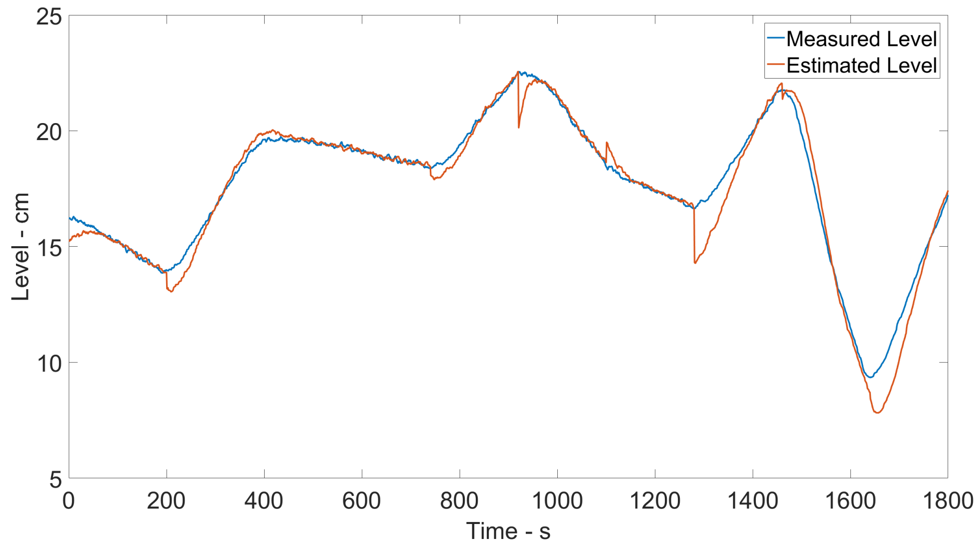
(a) With previous learning.



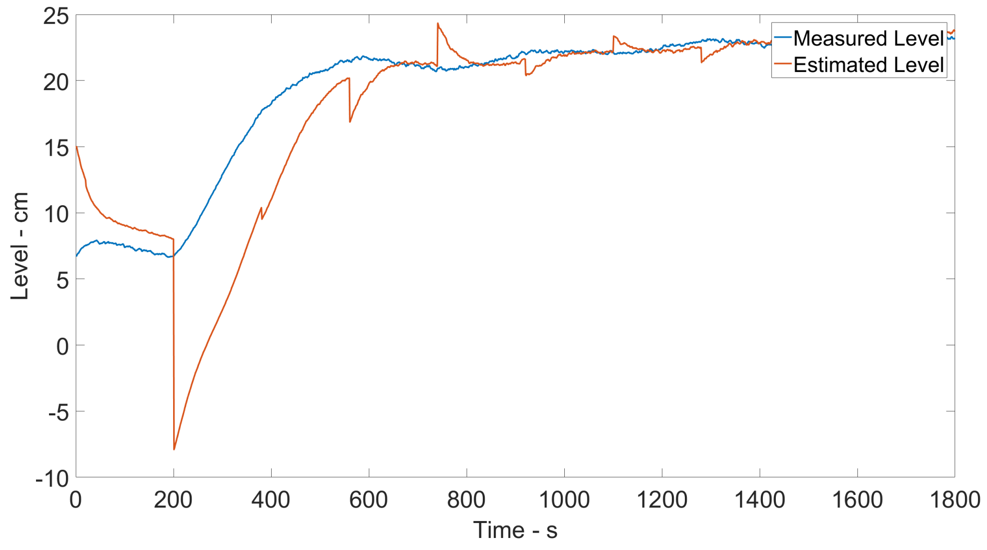
(b) Without previous learning.

Figure 6.8: RBF with backpropagation for center update and randomly started centers.

The results shown in Figure 6.9 belong to the RBN that had the centers started distributed uniformly and had both centers and weights updated with the backpropagation algorithm. It shows little difference in performance if compared to the RBN depicted in Figure 6.8, which is expected since the weights are also being updated, the solution for the minimizing problem tends to be closer.



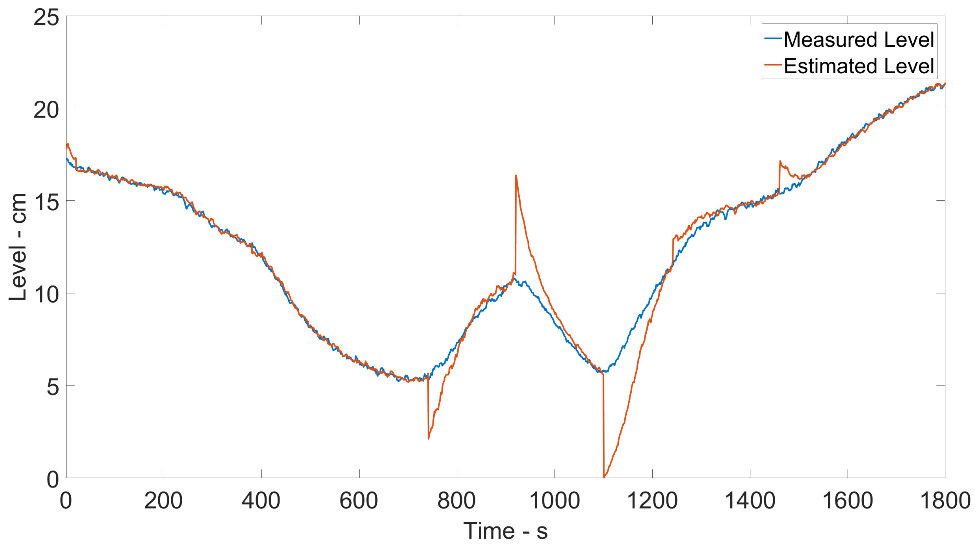
(a) With previous learning.



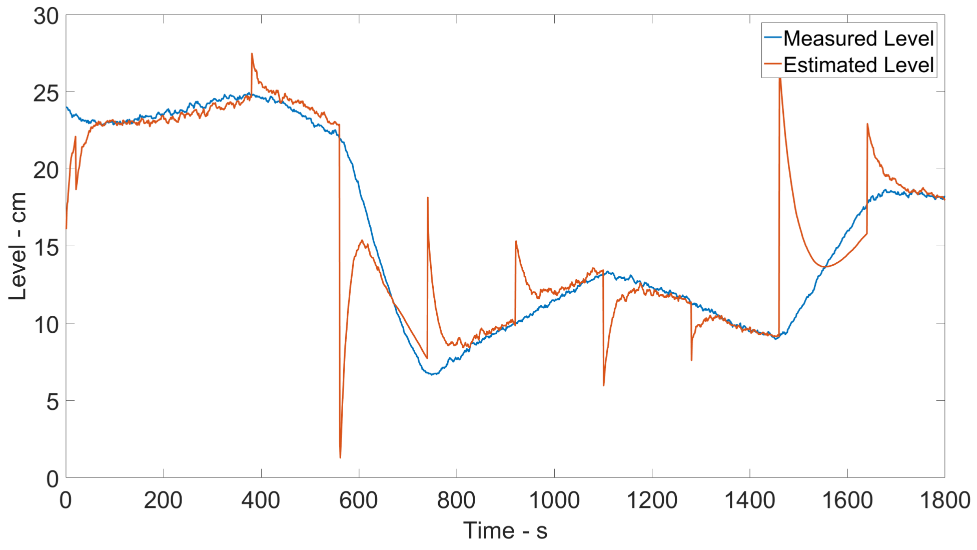
(b) Without previous learning.

Figure 6.9: RBF with backpropagation for center update and uniformly started centers.

Figure 6.10 presents the performance of the RBN with randomly started centers which are being updated with the k-means cluster algorithm. The usage of cluster for updating the centers in this work proved to be mare unstable the the other techniques.



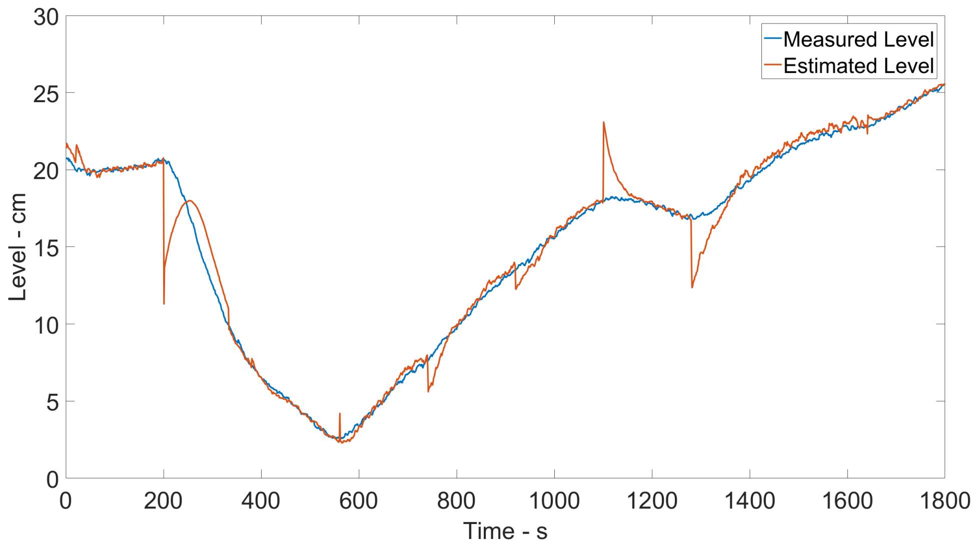
(a) With previous learning.



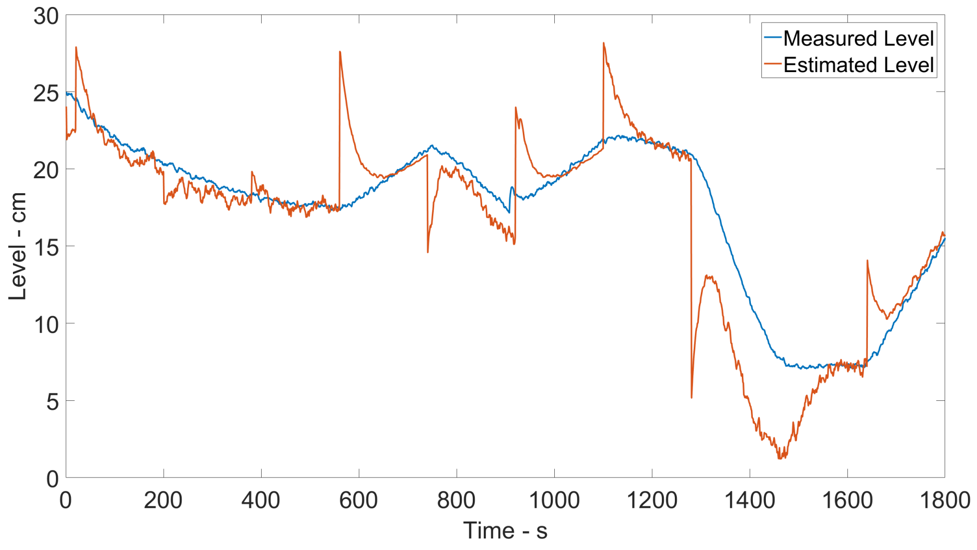
(b) Without previous learning.

Figure 6.10: RBF with fixed number of clusters and randomly started centers.

Figure 6.11 shows the results for the RBN with uniformly distributed started centers updated with k-means clustering. It shows alike behavior to the previous shown network for the same reason the RBNs with back propagation also show alike behavior, because all the variable parameters of the network are being updated.



(a) With previous learning.



(b) Without previous learning.

Figure 6.11: RBF with fixed number of clusters and uniformly started centers.

Figures 6.12 and 6.13 show the performance of the RBNs that had variable number of cluster which were created online when needed. In both cases the final number of clusters is 15 which is expected since the input space and the similarity parameter is the same for both cases. Figure 6.12 shows the network's performance when the clusters are created online but not updated and Figure 6.13 shows the case when the clusters are also updated online.

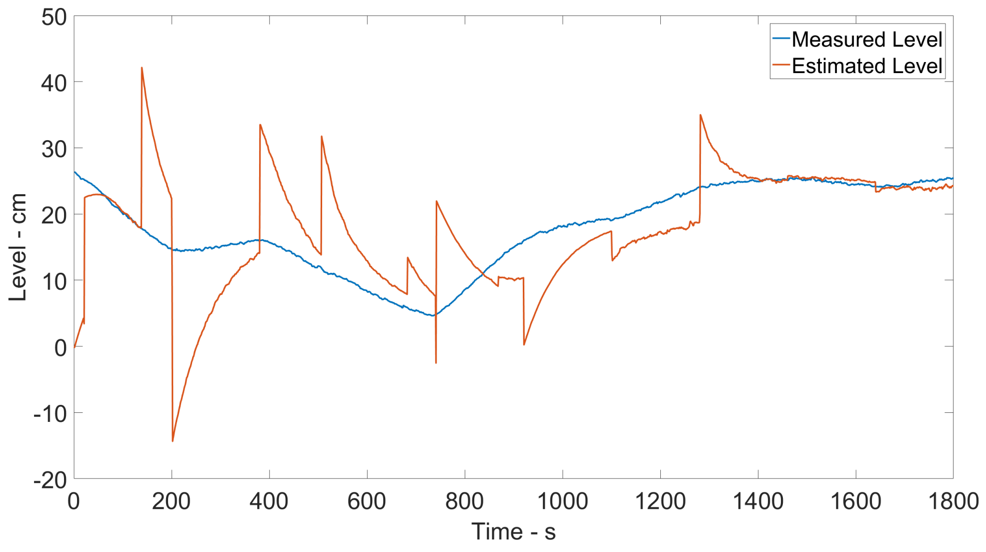


Figure 6.12: RBF with not corrected varying number of clusters.

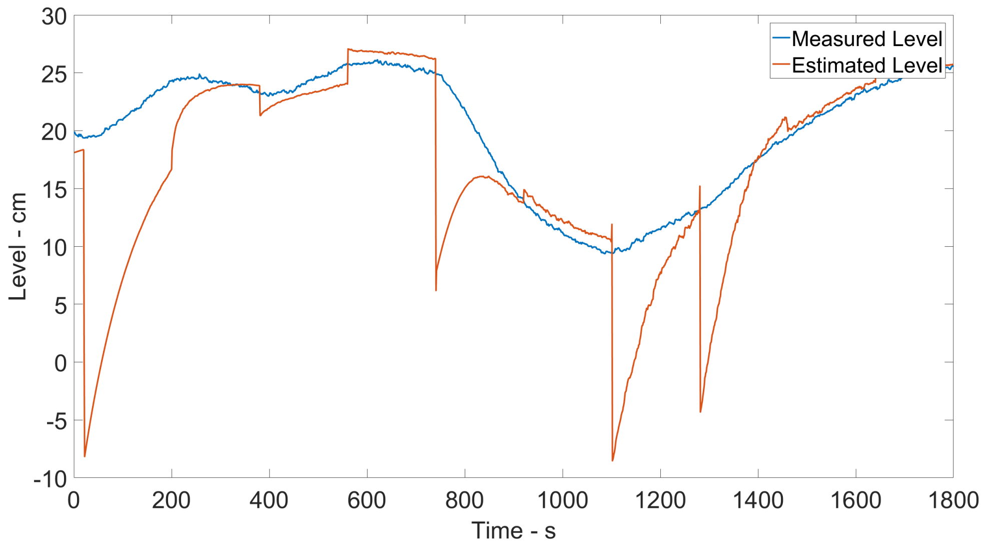


Figure 6.13: RBF with corrected varying number of clusters.

6.2 SYSTEM CONTROL

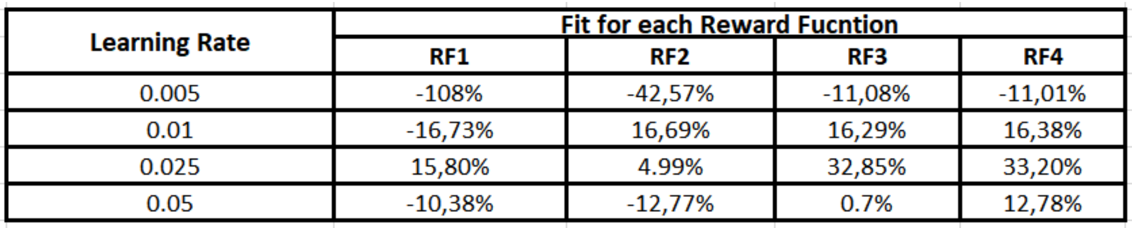
Even though the algorithm was implemented in a real environment, many of the tests and comparisons were made in simulation because of practicality and better usage of time, yet, every set of simulations has at least one real environment run for the sake of comparison and validation of the simulation data. With the algorithm of Chapter 5 in mind, some different reward functions were tested and analyzed in order to see how the system and algorithm reacts.

Figures 6.14, 6.16, 6.18 and 6.20 show the controlled system with RF1, RF2, RF3 and RF4 respectively using different learning rates. The learning rate is the size of the step taken towards the goal each iteration so all the results are as expected as a very small learning rate will not suffice for the controller to follow changes in the reference value and a large learning rate will render the system unstable, as seen in all four Figures. It can also be seen that a "discrete" approach to the reward function is not very efficient for this task, in opposition to what is seen in Wang et al. (2007) and that as expected there is very little difference between the responses of the controller using RF3 and RF4, since the system is very slow and then the derivative is very small. Regarding the objectives defined for this work, RF3 and RF4 were able to, with specific learning rates, perform as desired with stationary error below the margin of 0.5cm defined previously and the response of all reward functions show clear signs of a learning process as the controlled level is lowering overshoot and getting to the reference faster each time the reference value is repeated.

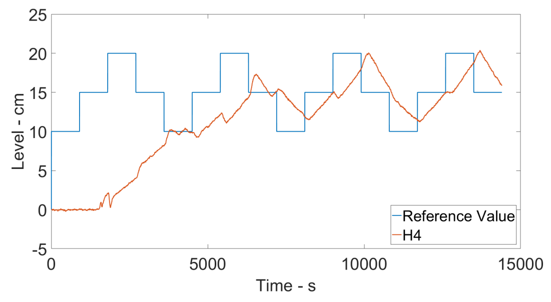
Table 6.3 show the fit of data collected in each experiment. The fit was calculated using the same parameter as for the identification process, the NRMSE. The values are expressed in percentage.

Figures 6.15, 6.17, 6.19 and 6.21 show the controller performing on the real fluid system. It can be seen that the performance of the controller on the real environment is extremely alike to the performance achieved by the controller on simulation, which shows that the inferences made from the simulation are trustworthy. These figures not only show that the behavior on the real environment follows the one observed in simulation but the calculated fit also shows similarity with the respective simulated experiment. The fits for the real tests are respectively 18.08%, 20.75%, 39.53% and 24.25%. The fits achieved by the controller on the real environment as slightly higher.

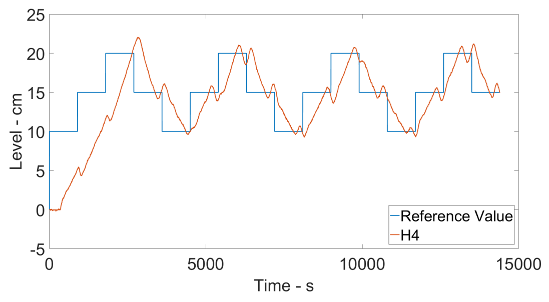
Table 6.3: Fits of each learning rate for each reward function.



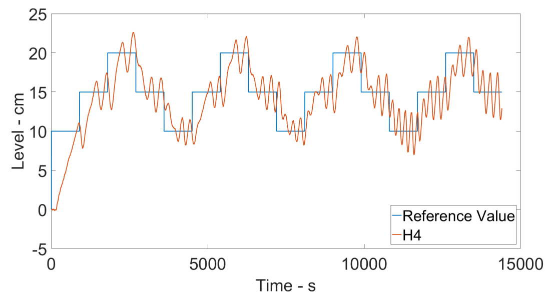
Learning Rate	Fit for each Reward Fucntion			
	RF1	RF2	RF3	RF4
0.005	-108%	-42,57%	-11,08%	-11,01%
0.01	-16,73%	16,69%	16,29%	16,38%
0.025	15,80%	4.99%	32,85%	33,20%
0.05	-10,38%	-12,77%	0.7%	12,78%



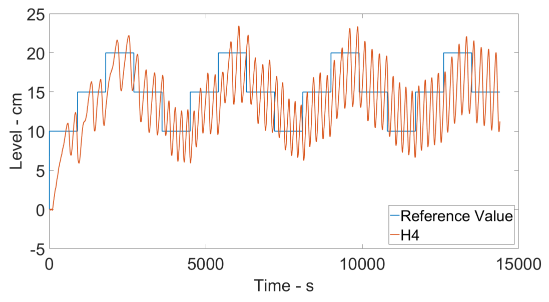
(a) Sample time 1s and Learning Rate 0.005



(b) Sample time 1s and Learning Rate 0.01



(c) Sample time 1s and Learning Rate 0.025



(d) Sample time 1s and Learning Rate 0.05

Figure 6.14: Various learning rates with RF1.

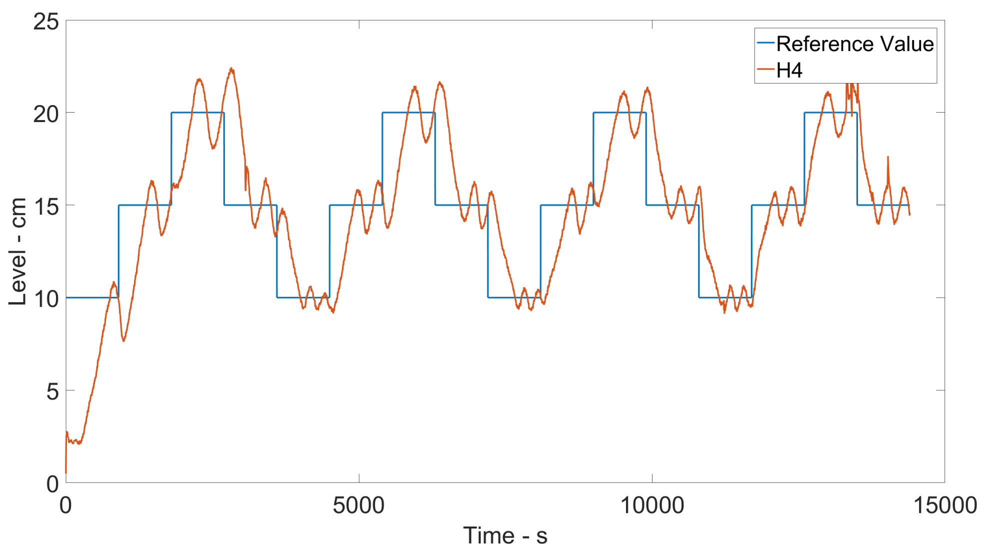
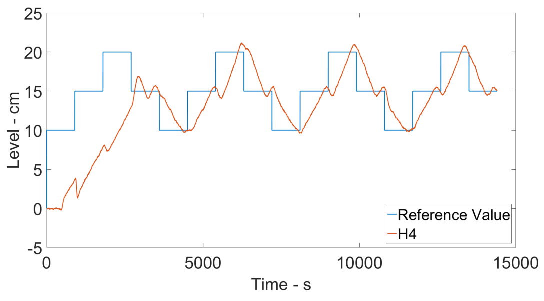
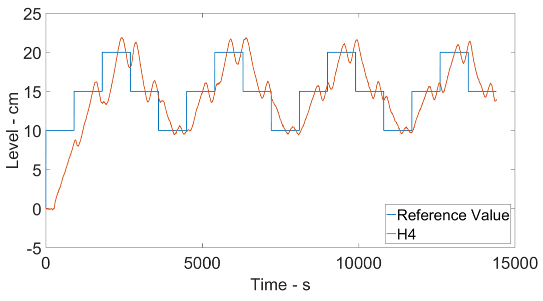


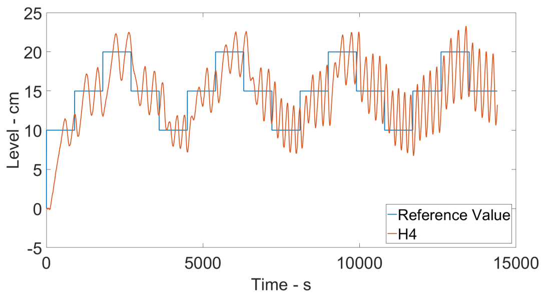
Figure 6.15: Online test with RF1. Learning rate 0.025 and sample time 1s.



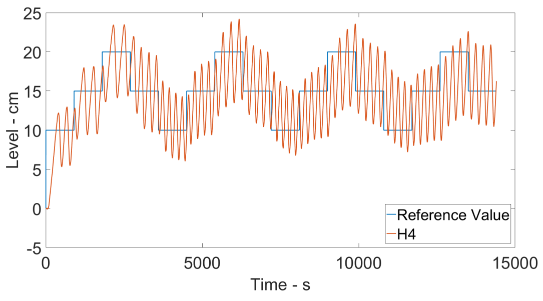
(a) Sample time 1s and Learning Rate 0.005



(b) Sample time 1s and Learning Rate 0.01



(c) Sample time 1s and Learning Rate 0.025



(d) Sample time 1s and Learning Rate 0.05

Figure 6.16: Various learning rates with RF2.

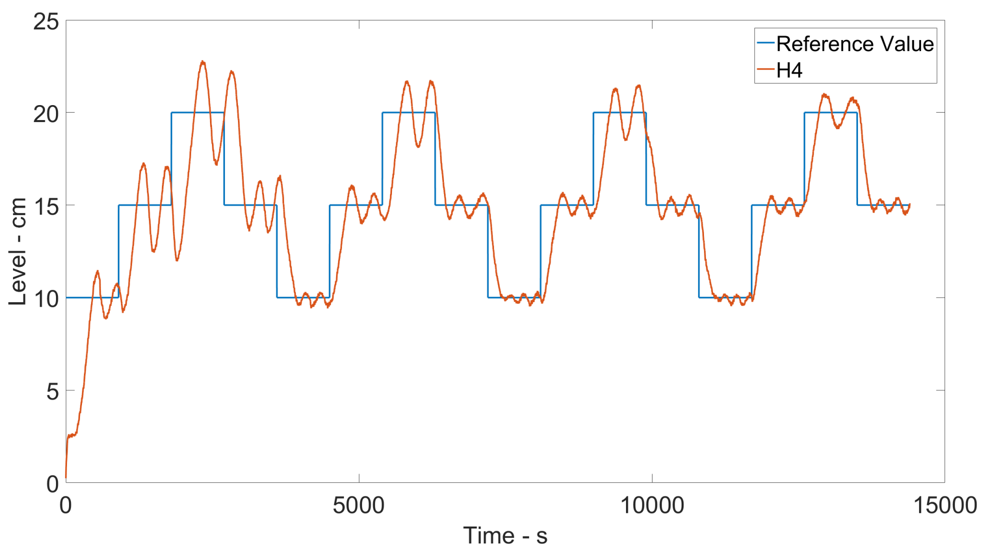
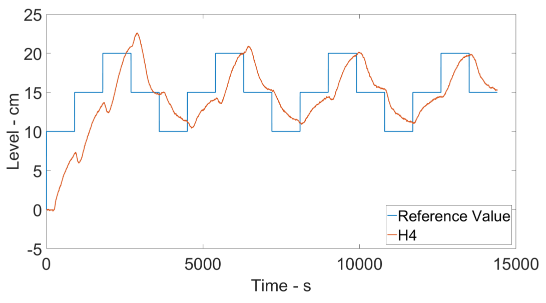
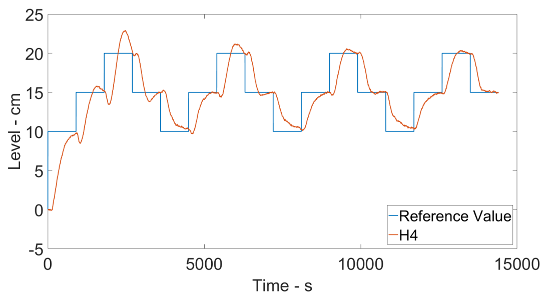


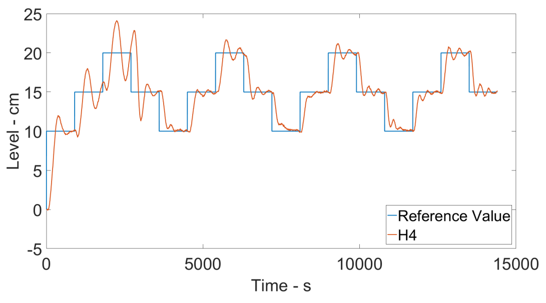
Figure 6.17: Online test with RF2. Learning rate 0.01 and sample time 1s.



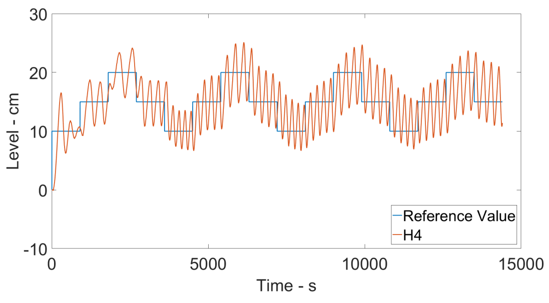
(a) Sample time 1s and Learning Rate 0.005



(b) Sample time 1s and Learning Rate 0.01



(c) Sample time 1s and Learning Rate 0.025



(d) Sample time 1s and Learning Rate 0.05

Figure 6.18: Various learning rates with RF3.

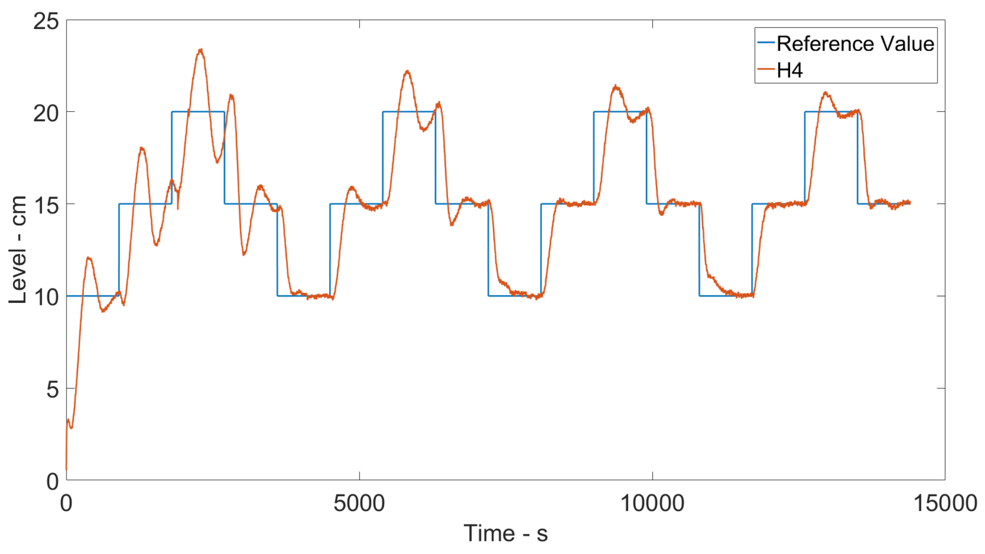
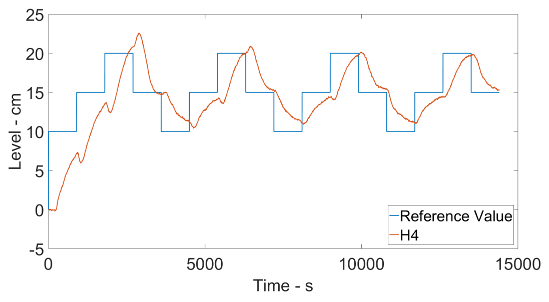
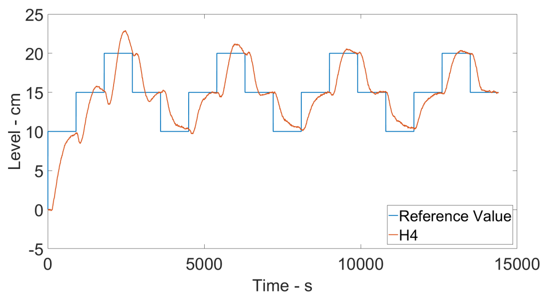


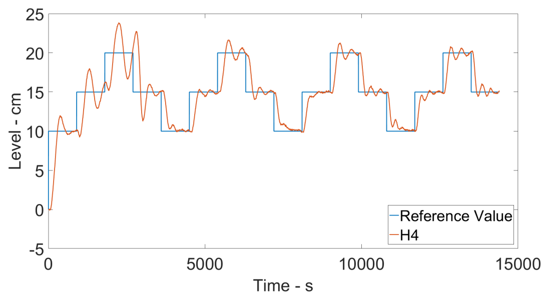
Figure 6.19: Online test with RF3. Learning rate 0.025 and sample time 1s.



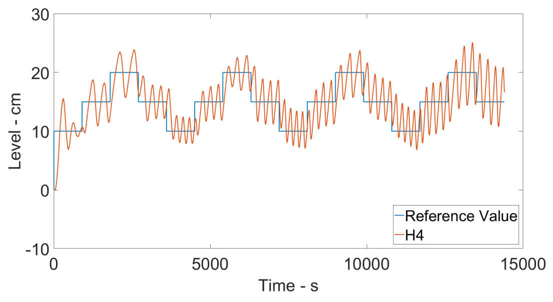
(a) Sample time 1s and Learning Rate 0.005



(b) Sample time 1s and Learning Rate 0.01



(c) Sample time 1s and Learning Rate 0.025



(d) Sample time 1s and Learning Rate 0.05

Figure 6.20: Various learning rates with RF4.

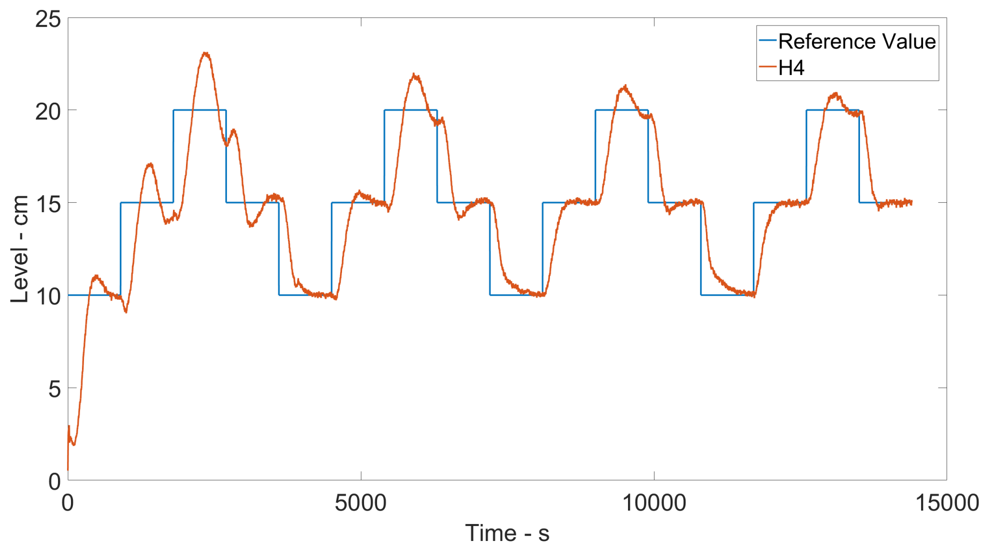
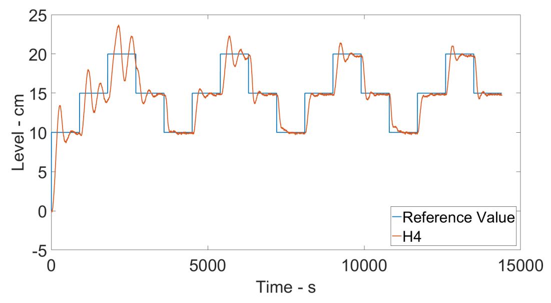
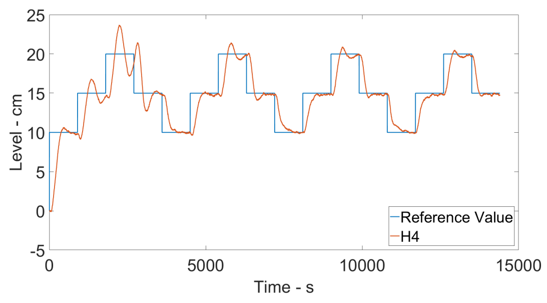


Figure 6.21: Online test with RF4. Learning rate 0.01 and sample time 1s.

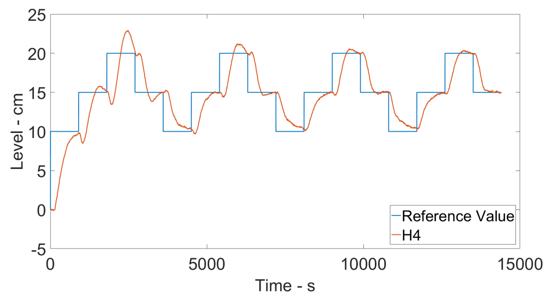
From figures above, RF3 was arbitrarily chosen to be the default (RF4 shows similar results, but since RF3 is simpler it was chosen) as the learning rate of 0.01, the reason for not choosing the learning rate of 0.025 which shows better fit will be explained further. Using the same initial conditions for the ANNs, different sample times were used. Figure 6.22 shows the controller performance with the previously defined default network with various sample times. It can be observed from the Figure that, as expected, the smaller the sample rate is (or the bigger the sample time), the slower the controller is. Just as the learning rate represents the size of the step given by the controller, the sample rate determines the amount of steps to be taken in a given amount of time. It is observed that a higher sample rate causes the system to oscillate more but also to learn faster regarding system accommodation. That being said, it should be interesting to use a higher sample rate and a matching smaller learning rate in order to give a lot of small steps instead of a few long steps. Figures 6.23 and 6.24 show the performance of the on-environment controller with different sample rates maintaining every other parameter constant. Again it can be seen that the real system response to the controller is extremely alike to the response given by the simulated system, although, for the sample time of 0.1s, the real environment oscillates a bit more.



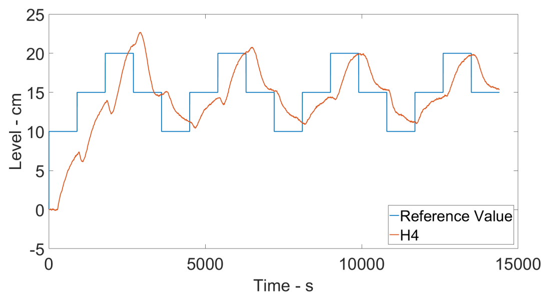
(a) RF3 with Learning rate 0.01 and sample time of 0.1s.



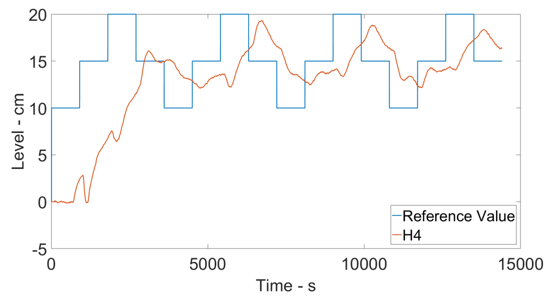
(b) RF3 with Learning rate 0.01 and sample time of 0.5s.



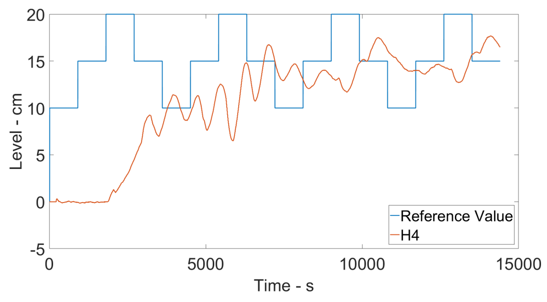
(c) RF3 with Learning rate 0.01 and sample time of 1s.



(d) RF3 with Learning rate 0.01 and sample time of 2s.



(e) RF3 with Learning rate 0.01 and sample time of 5s.



(f) RF3 with Learning rate 0.01 and sample time of 10s.

Figure 6.22: Effects of various sample rate.

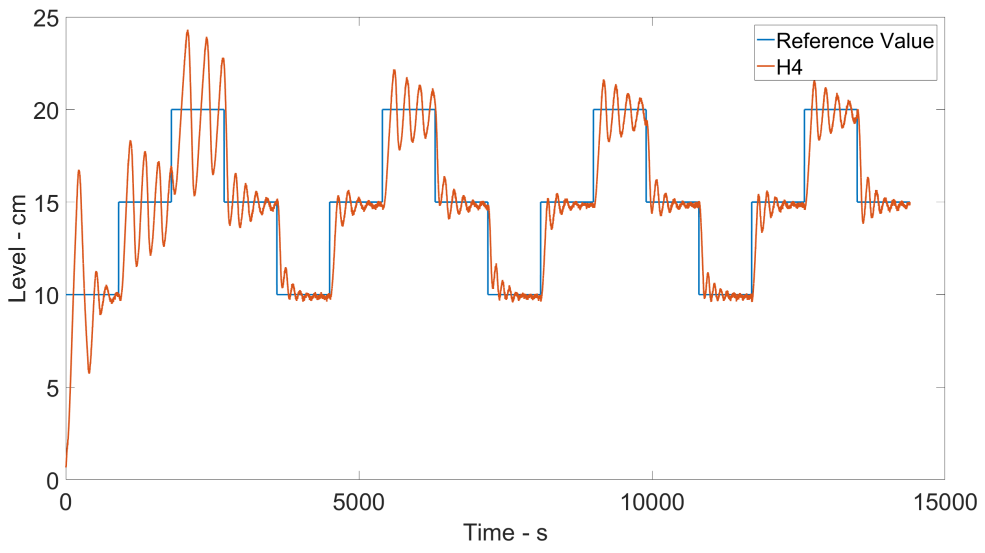


Figure 6.23: Online test with RF3. Learning rate 0.01 and sample time 0.1s.

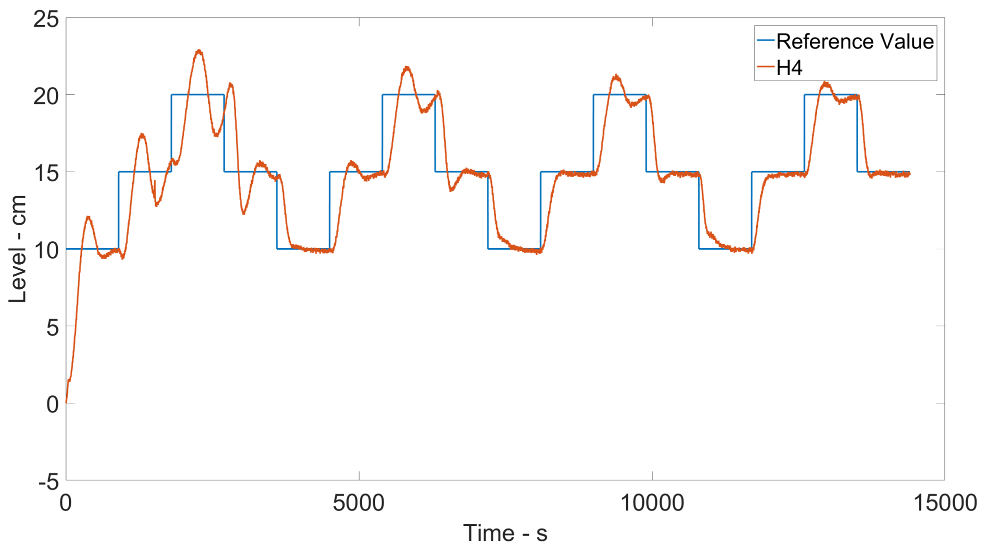
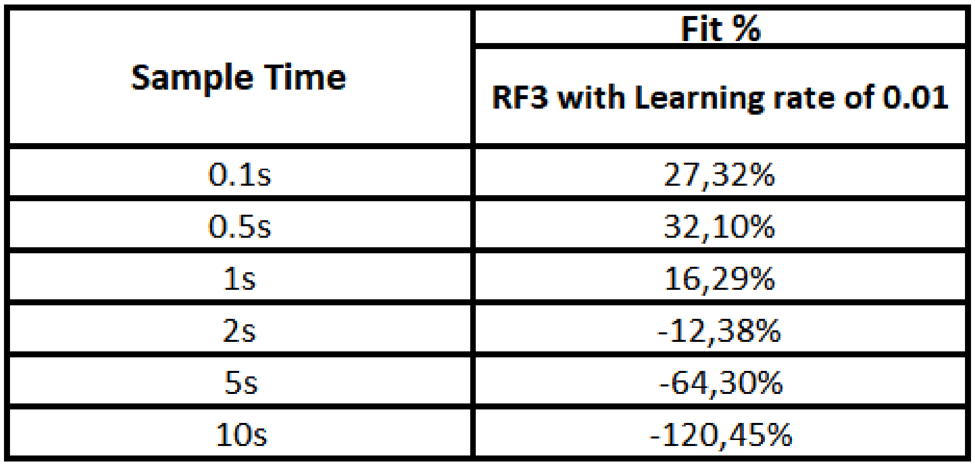


Figure 6.24: Online test with RF3. Learning rate 0.01 and sample time 0.5s.

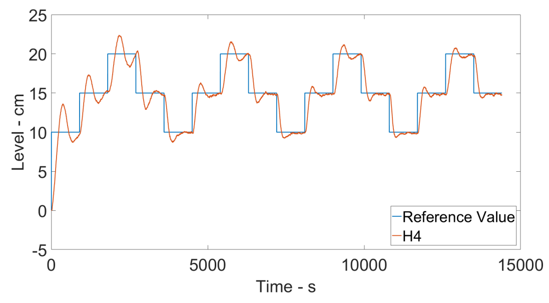
Table 6.4 shows the fit of each experiment. It can be observed from the figures and the table that the experiment with sample time of 1s and learning rate of 0.025 has similar behavior and fit as the experiment with sample time of 0.5s and learning rate of 0.01. this indicates that if inside specific margins for the learning and sample rate, maintaining the relation between them maintains the behavior. This phenomena can be observed in other relations between experiments as well. It is preferable, though, to use a higher sample rate and a lower learning rate, since the learning rate is what determines the stability of the learning algorithm, thus, instead of choosing sample time of 1s and learning rate of 0.025, the default rates were chosen to be 0.5s for the sample time and 0.01 for the learning rate, which achieves best fir among the experiments so far.

Table 6.4: Fits for each sample time used with RF3 and learning rate 0.01.

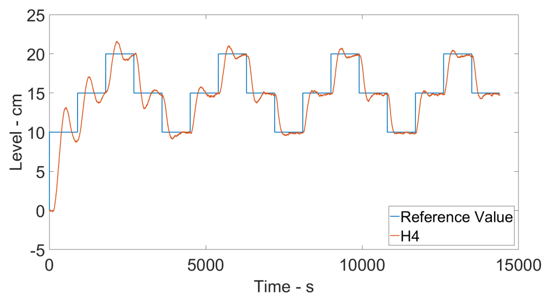


Sample Time	Fit %
	RF3 with Learning rate of 0.01
0.1s	27,32%
0.5s	32,10%
1s	16,29%
2s	-12,38%
5s	-64,30%
10s	-120,45%

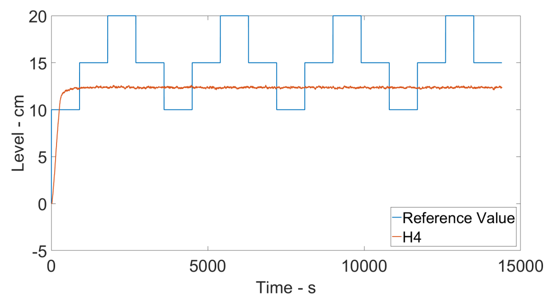
Every test so far, simulated or on real environment, had the very same initial conditions for the ANNs, those being the starting centers and weights. In order to verify the sensitivity of the controller to the initial conditions the system was subdued to eight different initial conditions and the results are shown in Figure 6.25. Although most responses are very similar some condition bring the controller to a wrong minimum while minimizing the cost function which is seen in Figure 6.25c, and some conditions, as seen in Figure 6.25e, make it harder for the controller to learn even though given enough time, it will reach the same result as it was observed in almost every test so far.



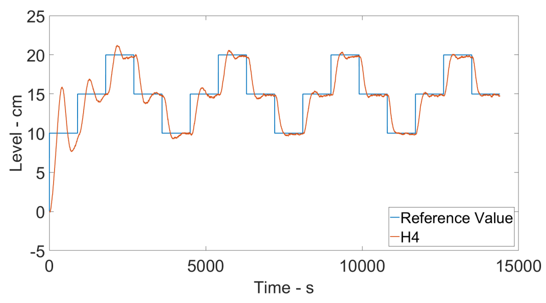
(a) Condition 1



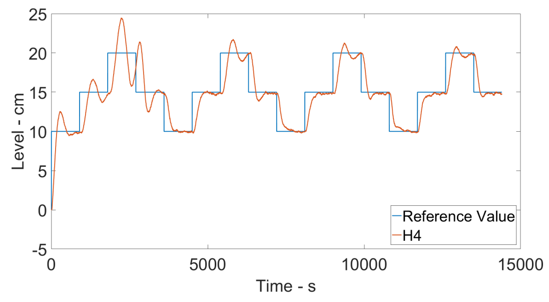
(b) Condition 2



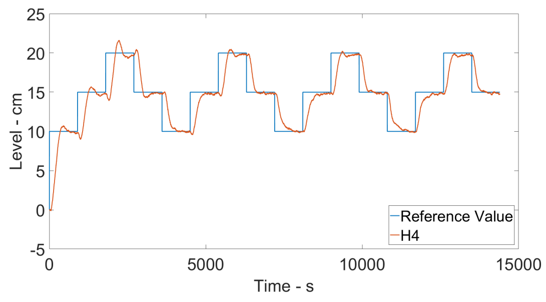
(c) Condition 3



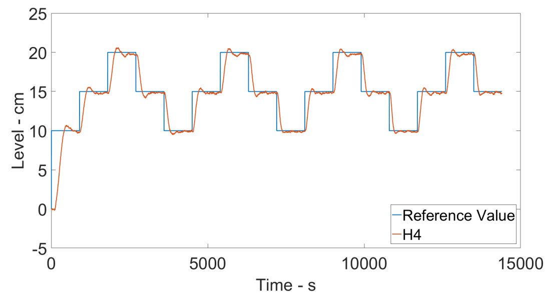
(d) Condition 4



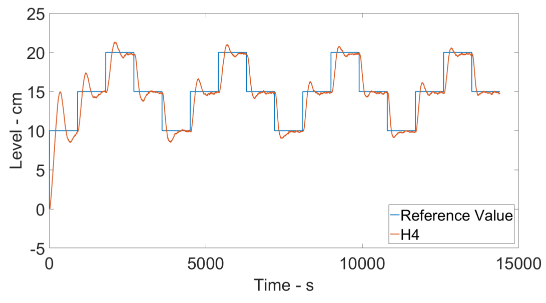
(e) Condition 5



(f) Condition 6



(g) Condition 7



(h) Condition 8

Figure 6.25: Different initial conditions of the neural networks maintaining the other parameters constant. RF3 with sample time of 0.5s and learning rate of 0.01.

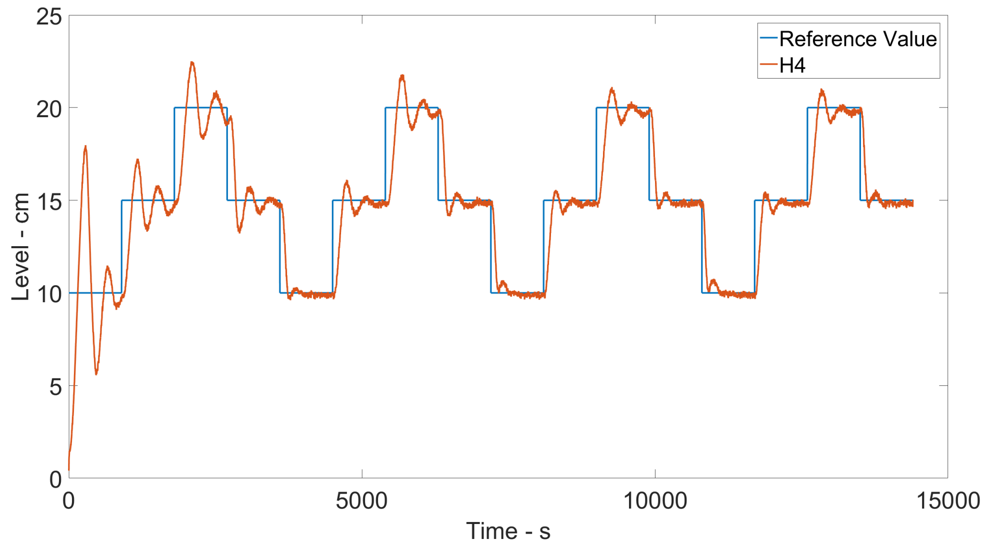


Figure 6.26: Online initial condition 1. RF3 with sample time 0.5s and learning rate 0.01.

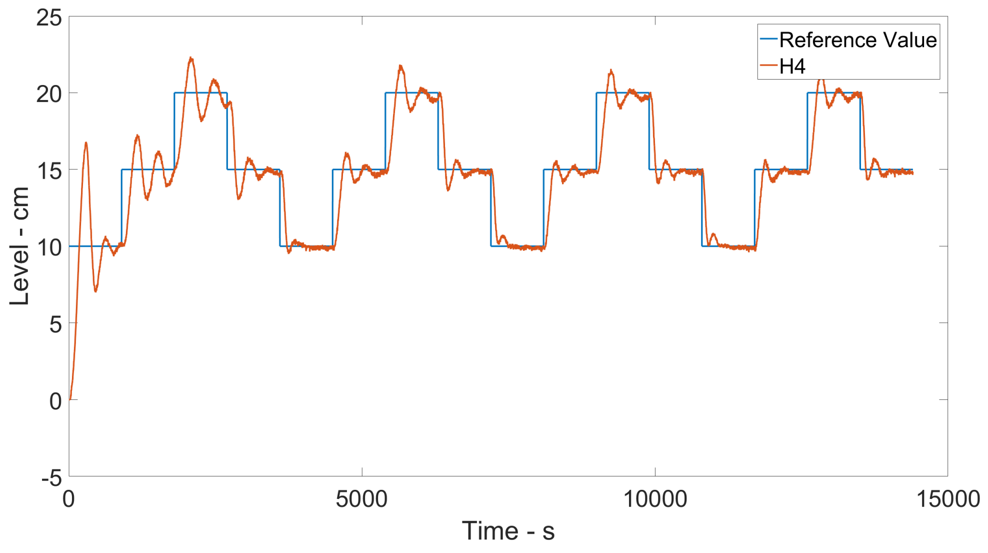


Figure 6.27: Online initial condition 2. RF3 with sample time 0.5s and learning rate 0.01.

Regarding this test, it seems that the real environment is less sensitive to different initial conditions as it shows very little variance between performances and not once has shown to be locked in a wrong local minimum as in Figure 6.25c.

The last test was to compare the RL controller performance with that of a PI projected to this very specific reference sequence that has been used so far. The comparison was made along the same reference values used in the tests so far and then with completely different values to see how both controller responds and how the compare. The RL controller used RF3, sample time of 0.5s and learning rate of 0.01.

Figures 6.28 and 6.30 show the controllers performance and Figures 6.29 and 6.31 show the respective control signal of the actuators. From Figure 6.28 it can be seen that at the beginning, the PI is superior, with faster response and lower overshoot, yet, as time passes, even with higher overshoot to some references, the RL controller stabilizes faster and demonstrates less oscillation. It is clear from the Figure and from previous tests, that in given time, maybe one or two more cycles of the reference values, the RL controller will surpass the PIs efficiency. Figure 6.30 is even more interesting because even though both controllers demonstrate similar performance, the RL controller, even without a specific cycle of reference values, depicts learning characteristics, lowering oscillation, overshoot, and steady time while the PI will perform the same over time where in many reference values, it never reached stationary value. In both cases, the RL controller performed as expected from the objectives defined for this work.

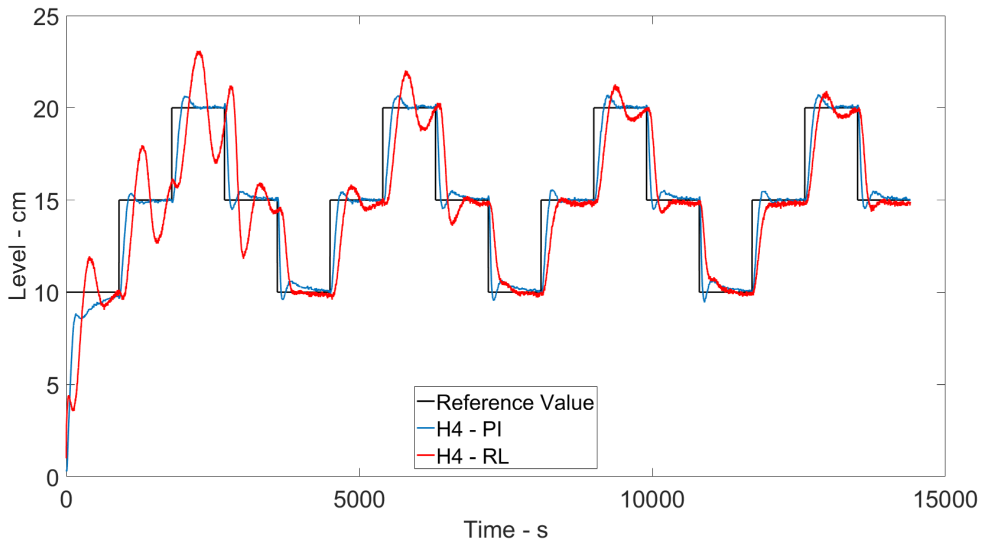


Figure 6.28: Comparison of the RL controller to a PI controller for the reference values used so far.

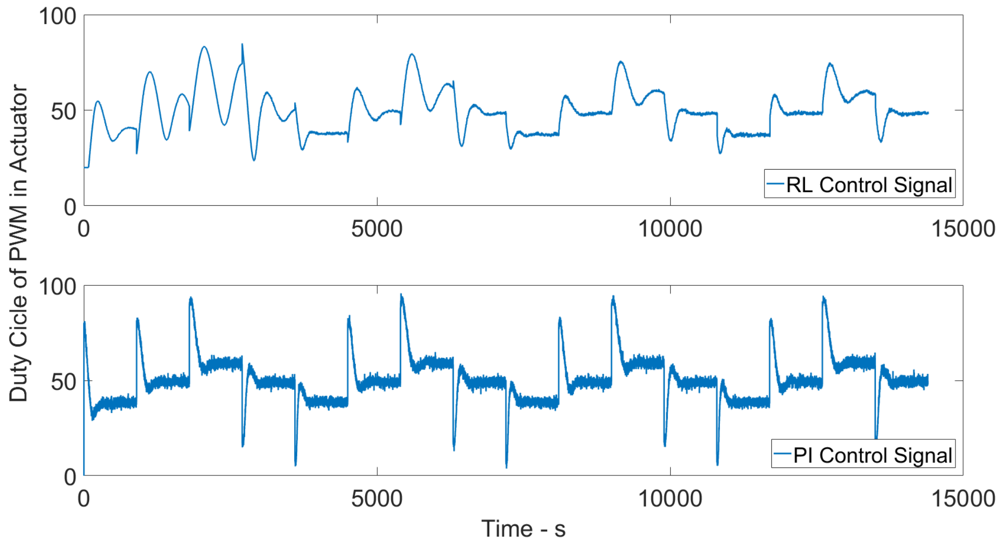


Figure 6.29: Control Signal of comparison for the reference values used so far.

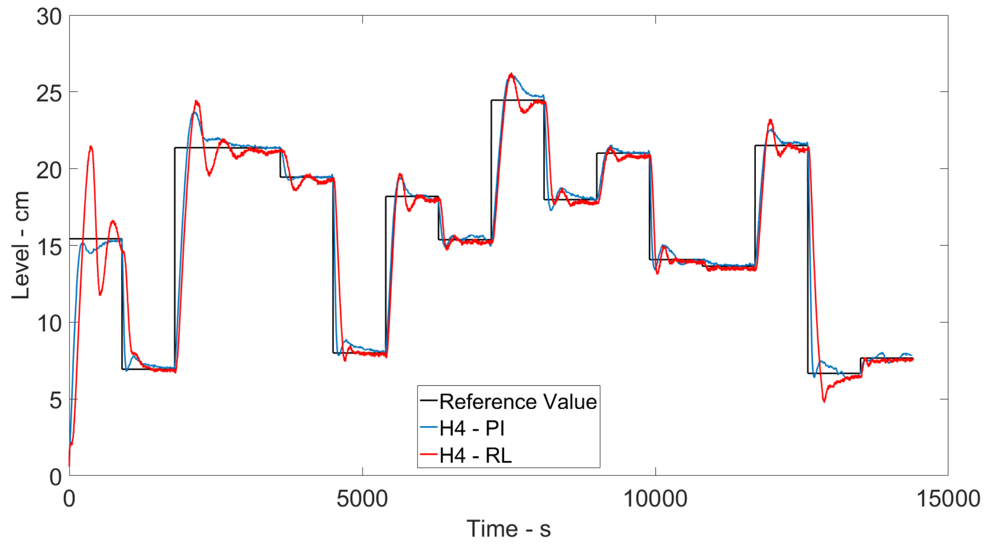


Figure 6.30: Comparison of the RL controller to a PI controller for generic reference values.

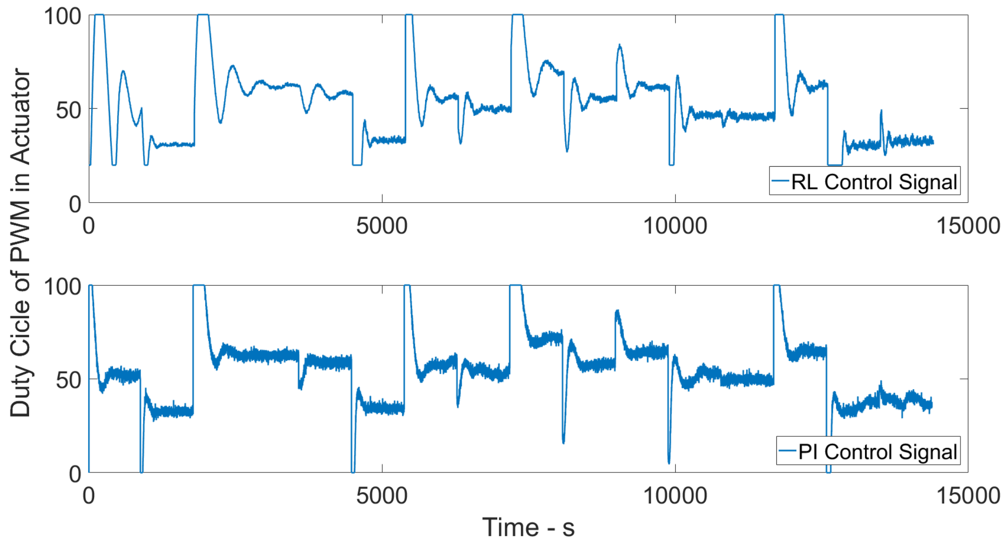


Figure 6.31: Control Signal of comparison for generic reference values.

Table 6.5 shows the fits for the RL controller and for the Pi controller for both cases. The Table shows the fit for the last cycle of the references given, meaning the last hour of the experiment. In the first case the RL surpasses the PI performance and in the second case it ties the PI performance. This can be expected since in the second case the RL didn't have enough time to learn as much as about the reference values it is visiting as it had in the first case, although, the result from the first case show that in given time, the RL will perform better for any reference value, while the PI will have the same performance.

Table 6.5: Fits of RL controller and PI Controller.

Controller	**Same References Used for Tuning**
RL	63,01%
PI	56,41%

Controller	**Generic References.**
RL	61,68%
PI	61,78%

7 CONCLUSIONS

Science is about knowing; engineering is about doing. ***Henry Petroski (1942 - Present)***

7.1 CONCLUSION

This work proposed using Artificial Neural Networks and Reinforcement Learning to control and identify a fourth order fluid level system. The controller used Radial Basis Networks and Multilayer Perceptrons as part of it structure in order to approximate pertinent functions and the same type of networks were used to perform system identification.

Regarding system identification, there was an extensive investigation on different types of networks, different layouts and different learning algorithms. Multilayer Parceptrons, with one and two hidden layers, using the backpropagation algorithm for learning were tested along several Radial Basis Networks that used backpropagation and clustering as a form of learning. These networks were compared to models obtained by classical methods, such as the ARX model, the ARMAX model and the Hammerstein and Wiener models.

Results on the identification process show great performance in comparison to classical models. The Fit to validation data was higher then these models and in some cases, surpassed 95%. The investigation on the different types of networks and layouts not only confirmed features already described in the literature, but also, the information gathered was used to define which networks to be used in the control problem regarding its features. Multilayer Parceptrons were found to be more stable, while Radial Basis Networks were found to respond faster. The Radial Basis Network that used fixed centers performed better then the other networks and so, was chosen to approximate the functions for the controller, while the Multilayer Perceptron with 2 hidden layer has shown remarkable stability, thus it was used for the system identification task.

The controller design also went through extensive investigation. Evaluation on how parameters such as sample rate, learning rate and initial conditions could affect the controller were tested and analyzed. Different reward functions were used to test the idea of the algorithm. After the parameter evaluation a final version of the designed controller performed against a PI for a set of defined reference values, which were the values used in all the tests, and then for a random set of values to verify how the designed controller behave outside known situation.

Results on the controller performance showed that nearly all the objectives set at the beginning were achieved, with exception to the ability to deal with parameter variability which was not tested. The designed controller, when using the right reward function and parameters, could perform regulatory control with the desired efficiency maintaining the stationary error inside the previously defined margin of 0.5cm and showed learning capacity as the performance clearly enhanced over time.

When compared to a PI the designed controller showed worse results when responding to the first reference values, but over time, it became more and more efficient and evened the PI performance surpassing it eventually and being able to steady the reference value faster and with least overshoot.

The variation of the position of the gate valves was not performed because different valve configuration incur in different dynamics, meaning that the average delay will differ. Since the RL algorithm is in use of this information this test was not performed because this information is not available.

7.2 PROPOSAL FOR FUTURE WORK

The main focus for future work is to develop the controller further and test it to deal with parameter variability, using the same system and changing the valve configuration online to verify how the controller reacts. To understand how the controller react to such variations might show another path to make the controller more robust and more efficient. If the controller can deal with parameter variability in a steady way, it can probably tackle a range of problems, so, the developed algorithm could be used in various other controls problems.

Another possibility is to try new networks and learning algorithms and study and detail their performance using the already tested networks as a form of comparison. For instance, Deep Learning could be tested and other learning algorithms as well.

Bibliography

Aguirre, L. A. (2015). *Introducao a Identificacao de Sistemas, Quarta Edicao*. Editora UFMG.

Bernardes, M. C., G. A. F. Melo, A. A. Freitas, G. A. Borges, and A. Bauchspiess (2006). Instrumentacao e estimacao de parametros de um sistema de nivel de liquido com quatro tanques interligados. *XII CONGRESSO BRASILEIRO DE AUTOMATICA, Salvador. 2006. pp. 3427-3432.*.

Cooper, L. N., C. Elbaum, and D. L. Reilly (1982, April 20). Self organizing general pattern class separator and identifier. US Patent 4,326,259.

Coulom, R. (2002). *Reinforcement Learning Using Neural Networks, with Applications to Motor Control*. Ph. D. thesis, Institut National Polytechnique de Grenoble - INPG.

Cui, R., C. Yang, Y. Li, and S. Sharma (2017, jun). Adaptive neural network control of AUVs with control input nonlinearities using reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics: Systems 47*(6), 1019–1029.

Deisenroth, M. P. and C. E. Rasmussen (2009). Efficient reinforcement learning for motor control. *10th International PhD Workshop on Systems and Control*.

Esfe, M. H. (2017). Designing an artificial neural network using radial basis function (rbf-ann) to model thermal conductivity of ethylene glycolâwater-based tio2 nanofluids. *J THERM ANAL CALORIM*.

Esfe, M. H., M. Afrand, W.-M. Yan, and M. Akbari (2015). Applicability of artificial neural network and nonlinear regression to predict thermal conductivity modeling of al2o3âwater nanofluids using experimental data. *International Communications in Heat and Mass Transfer*.

Fan, X., S. Li, and L. Tian (2015). Chaotic characteristic identification for carbon price and an multi-layer perceptron network prediction model. *Expert Systems with Applications*.

Fan, X., L. Wang, and S. Li (2016). Predicting chaotic coal prices using a multi-layer perceptron network model. *Resources Policy*.

Ferrari, S. and R. F. Stengel (2005). Smooth function approximation using neural networks. *IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 16, NO. 1, JANUARY 2005*.

Gomes, C. R. and J. A. C. C. Medeiros (2015). Neural network of gaussian radial basis functions applied to the problem of identification of nuclear accidents in a pwr nuclear power plant. *Annals of Nuclear Energy*.

Halalia, M. A., V. Azaria, M. Arablooa, A. H. Mohammadib, and A. Bahadori (2016). Application of a radial basis function neural network to estimate pressure gradient in waterâoil pipelines. *Journal of the Taiwan Institute of Chemical Engineers*.

Haykin, S. S. (2009). *Neural Networks and Learning Machines*. Prentice Hall.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proc. NatL Acad. Sci. USA*.

Hwangbo, J., I. Sa, R. Siegwart, and M. Hutter (2017, oct). Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters 2*(4), 2096–2103.

Kamthe, S. and M. P. Deisenroth (2017). Data-efficient reinforcement learning with probabilistic model predictive control.

Kiumarsi, B., F. L. Lewis, H. Modares, A. Karimpour, and M.-B. Naghibi-Sistani (2014, apr). Reinforcement q-learning for optimal tracking control of linear discrete-time systems with unknown dynamics. *Automatica 50*(4), 1167–1175.

Klopf, A. H. (1972). Brain function and adaptive systems: a heterostatic theory. Technical report, AIR FORCE CAMBRIDGE RESEARCH LABS HANSCOM AFB MA.

Lechappe, V., S. Rouquet, A. Gonzalez, F. Plestan, J. D. Leon, E. Moulay, and A. Glumineau (2016, sep). Delay estimation and predictive control of uncertain systems with input delay: Application to a DC motor. *IEEE Transactions on Industrial Electronics 63*(9), 5849–5857.

Lia, F., J. Qiaoa, H. Hana, and C. Yang (2016). A self-organizing cascade neural network with random weights fornonlinear system modeling. *Applied Soft Computing 42*.

Lillicrap, T. P., J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra (2016). Continuous control with deep reinforcement learning. *ICLR*.

Lin, C.-K. (2011, apr). Radial basis function neural network-based adaptive critic control of induction motors. *Applied Soft Computing 11*(3), 3066–3074.

Lin, W.-S. and C.-H. Zheng (2012). Constrained adaptive optimal control using a reinforcement learning agent. *Automatica*.

Liu, Y.-J., J. Li, S. Tong, and C. L. P. Chen (2016). Neural network control-based adaptive learning design for nonlinear systems with full-state constraints. *IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, VOL. 27, NO. 7, JULY 2016*.

Liu, Y.-J., L. Tang, S. Tong, C. L. P. Chen, and D.-J. Li (2015, jan). Reinforcement learning design-based adaptive tracking control with less learning parameters for nonlinear discrete-time MIMO systems. *IEEE Transactions on Neural Networks and Learning Systems 26*(1), 165–176.

Luo, B., H.-N. Wu, and T. Huang (2015). Off-policy reinforcement learning for hâ control design. *IEEE TRANSACTIONS ON CYBERNETICS, VOL. 45, NO. 1*.

McCulloch, W. S. and W. Pitts (1943, dec). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics 5*(4), 115–133.

Michie, D. (1961). Trial and error. *Science Survey, Part 2*, 129–145.

Michie, D. (1963, nov). Experiments on the mechanization of game-learning part i. characterization of the model and its parameters. *The Computer Journal 6*(3), 232–236.

Michie, D. and R. A. Chambers (1968). Boxes: An experiment in adaptive control. *Machine intelligence 2*(2), 137–152.

Minski, M. L. and S. A. Papert (1969). Perceptrons: an introduction to computational geometry. *MA: MIT Press, Cambridge*.

Napoli, C., G. Pappalardo, and E. Tramontana (2014). An agent-driven semantical identifier using radial basis neural networks and reinforcement learning. *Proceedings of the XV Workshop âDagli Oggetti agli Agentiâ*.

Narendra, K. and K. Parthasarathy (1990, mar). Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks 1*(1), 4–27.

Noel, M. M. and B. J. Pandian (2014, oct). Control of a nonlinear liquid level system using a new artificial neural network based reinforcement learning approach. *Applied Soft Computing 23*, 444–451.

Orr, M. J. L. (1996). Introduction to radial basis function networks.

Pan, Y. and H. Yu (2017). Biomimetic hybrid feedback feedforward neural-network learning control. *IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, VOL. 28, NO. 6, JUNE 2017*.

Papierok, S., A. Noglik, and J. Pauli (2008). Application of reinforcement learning in a real environment using an rbf network. *1st International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems*.

Pillonetto, G., F. Dinuzzo, T. Chenc, G. D. Nicolao, and L. Ljung. (2014). Kernel methods in system identification, machine learning and function estimation: A survey. *Automatica*.

Pradeep, D. J., M. M. Noel, and N. Arun (2016, jun). Nonlinear control of a boost converter using a robust regression based reinforcement learning algorithm. *Engineering Applications of Artificial Intelligence 52*, 1–9.

Ramanathan, P., K. K. Mangla, and S. Satpathy (2018, feb). Smart controller for conical tank system using reinforcement learning algorithm. *Measurement 116*, 422–428.

Rao, J., H. An, T. Zhang, Y. Chen, and H. Ma (2016, aug). Single leg operational space control of quadruped robot based on reinforcement learning. In *2016 IEEE Chinese Guidance, Navigation and Control Conference (CGNCC)*. IEEE.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review 65*(6), 386–408.

Slusny, S., R. Neruda, and P. Vidnerova (2008). Comparison of rbf learning and reinforcement learning on the maze exploration problem. *International Conference on Artificial Neural Networks - ICANN*.

Sutton, R., A. Barto, and R. J. Williams (1991). Reinforcement learning is direct adaptive optimal control. *American Control Conference*.

Sutton, R. S. (1988, aug). Learning to predict by the methods of temporal differences. *Machine Learning 3*(1), 9–44.

Sutton, R. S. and A. G. Barto (1981). Toward a modern theory of adaptive networks: Expectation and prediction. *Psychological Review 88*(2), 135–170.

Sutton, R. S. and A. G. Barto (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Syafiie, S., F. Tadeo, E. Martinez, and T. Alvarez (2011). Model-free control based on reinforcement learning for a wastewater treatment problem. *Applied Soft Computing*.

Tesauro, G. (1995). Temporal difference learning and td-gammon. *Communications of the ACM 38*(3), 58–68.

Vamvoudakis, K. G., M. F. Miranda, and J. P. Hespanha (2016, nov). Asymptotically stable adaptive–optimal control algorithm with saturating actuators and relaxed persistence of excitation. *IEEE Transactions on Neural Networks and Learning Systems 27*(11), 2386–2398.

Wang, T., H. Gao, and J. Qiu (2016, feb). A combined adaptive neural network and nonlinear model predictive control for multirate networked industrial process control. *IEEE Transactions on Neural Networks and Learning Systems 27*(2), 416–425.

Wang, X. S., Y. H. Cheng, and W. Sun (2007, mar). A proposal of adaptive PID controller based on reinforcement learning. *Journal of China University of Mining and Technology 17*(1), 40–44.

Wanga, W.-X., Y.-C. Lai, and C. Grebogi (2016). Data based identification and prediction of nonlinear and complex dynamical systems. *Physics Reports*.

Warwick, K. and R. Craddock (1996). An introduction to radial basis functions for system identification. a comparison with other neural network methods. *IEEE Conference on Decision and Control*.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. Ph. D. thesis, King's College, Cambridge.

Widrow, B., N. K. Gupta, and S. Maitra (1973, sep). Punish/reward: Learning with a critic in adaptive threshold systems. *IEEE Transactions on Systems, Man, and Cybernetics SMC-3*(5), 455–465.

Widrow, B. and M. Hoff (1960). An adaptive adaline neuron using chemical memistors. Technical report, Stanford University.

Witten, I. H. (1977, aug). An adaptive optimal controller for discrete-time markov environments. *Information and Control 34*(4), 286–295.

Xu, B., C. Yang, and Z. Shi (2014, mar). Reinforcement learning output feedback NN control using deterministic learning technique. *IEEE Transactions on Neural Networks and Learning Systems 25*(3), 635–641.

Yang, Q. and S. Jagannathan (2012, apr). Reinforcement learning controller design for affine nonlinear discrete-time systems using online approximators. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 42*(2), 377–390.

Yu, H., P. D. Reiner, T. Xie, T. Bartczak, and B. M. Wilamowski (2014). An incremental design of radial basis function networks. *IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, VOL. 25, NO. 10, OCTOBER 2014*.

Zhu, J. Z., J. X. Cao, and Y. Zhu (2014). Traffic volume forecasting based on radial basis function neural network with the consideration of traffic flows at the adjacent intersections. *Transportation Research*.