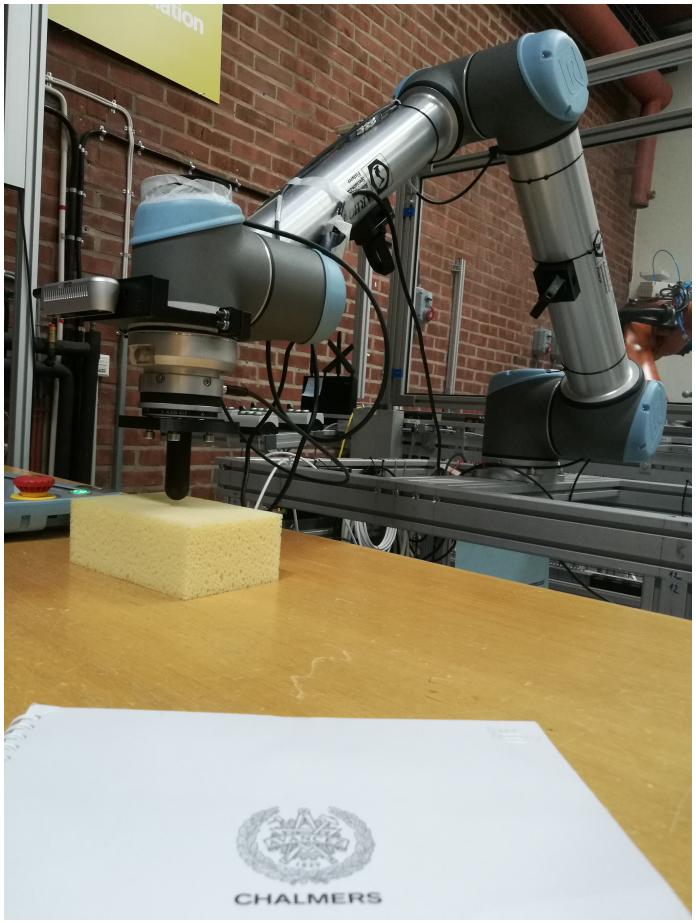
CHALMERS
UNIVERSITY OF TECHNOLOGY



Data-driven Modeling for Robotic Manipulation of Soft Objects

Master's thesis in Systems, Control and Mechatronics

Konstantinos Alfonsos

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's thesis 2020:EENX30

Data-driven Modeling for Robotic Manipulation of Soft and Deformable Objects

Konstantinos Alfonsos

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
Gothenburg, Sweden 2020

Data-driven Modeling forRobotic Manipulation of Soft and Deformable Objects
Konstantinos Alfonsos

Supervisor: Yiannis Karayiannidis, Department of Electrical Engineering
Examiner: Yiannis Karayiannidis, Department of Electrical Engineering

Master's Thesis 2020:EENX30
Department Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 73 489 4285

Data-driven Modeling forRobotic Manipulation of Soft and Deformable Objects
Konstantinos Alfonsos
Department of Electrical Engineering
Chalmers University of Technology

Abstract

The necessity for robotic manipulators to interact with deformable objects for a continuously increasing number of tasks, drives the robotics community to study ways to make this interaction as efficient and as accurate as possible. A promising approach is to use data-driven machine learning to approximate deformable object models. Machine learning makes use of neural networks in order to represent a function which maps an input to an output.
In this thesis, a neural network was designed and trained for the purpose of approximating a sponge's dynamical model. The trained model was then used in an online implementation as a feedforward term, in P and PI control schemes and its advantages and disadvantages were shown.
Furthermore, an optimal adaptive controller was designed using Reinforcement Learning, and its advantages compared to the static controllers were shown. Finally, the adaptability of the RL controller scheme was shown in different trajectories.

Keywords: Force Control, Model and Parameter Identification, Artificial Neural Networks (ANN), Deep Machine Learning, Reinforcement Learning (RL), Robot Interaction with Uncertain Environments, Deformable Objects

Acknowledgements

I would first like to thank my thesis advisor, Yiannis Karayiannidis of the [Electrical Engineering department at Chalmers University of Technology. The door to Prof. Karayiannidis office was always open whenever I ran into a trouble spot or had a question about my research or writing. He consistently allowed this paper to be my own work, but steered me in the right direction whenever he thought I needed it.

Finally, I must express my very profound gratitude to my parents and to my partner for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Konstantinos Alfonsos, Gothenburg, June 2020

Contents

List of Figures

List of Tables

1
Introduction

1.1 Background

During the last decade, the importance of robots in our lives has been increasing and their development became essential for the technology and industry evolution. The new industrial revolution provides flexible, collaborative and more independent machines, which could ultimately reduce the workload of highly skilled workforce. In today's industry, robots are being used in a lot of applications, such as production lines in factories, food service, welding, and medical purposes. An even greater number of applications, require the robot to come in contract with soft, deformable objects, like robot-assisted surgery, robotic-collection and placement of fruits and vegetables, elastic parts robotic-assembly. This creates the necessity of extending the capabilities of robotic manipulators even further, in order to cope with tasks like these.
Designing autonomous robotic systems able to interact with deformable objects without human intervention constitutes a challenging research area. Although manipulation and interaction with rigid objects is a mature field in robotics, with over three decades of work, the study and identification of soft and deformable objects has not been as extensive in the robotics community. Another important issue is that most of the techniques and strategies devised for the manipulation of rigid objects cannot be directly applied to deformable objects.
Hence, this need, has led to many diverse approaches to handle deformable objects with robots.

This thesis project aims to cover two main aspects. The first step will be to investigate the learning capabilities of modern AI tools such as deep neural networks, for the problem of learning the model of soft, deformable objects. The second aspect will be to assess the learnt models regarding their applicability for real-time force control problems.

1.2 Related Work

A vast amount of different papers have dealt with deformable object identification, as well as force control on robot manipulators. However, not a lot of research has been done combining both. There are mainly three parts in this report: modeling

of the soft/deformable object, control design, to perform force control on the object with a robotic manipulator using model-based conventional approaches and RL, and finally, the experiments, to demonstrate and compare the performance of the developed control schemes.

1.2.1 Soft Object Identification

Modeling of deformable objects and parameter estimation are active areas of research. There are different available methods to model the environment in real-time control applications. One way to represent non-rigid objects and simulate deformations is to use dynamic models like mass-spring systems, and perform real-time parameter estimation as Haddadi and Hashtrudi-Zaad [1] propose.

The problem of real-time identification of dynamic models of contact environments is treated in [1]. Real-time estimates of environment dynamics play an important role in the design of controllers for stable interaction between robotic manipulators and unknown environments. Haddadi and Hashtrudi-Zaad [1], use dynamic contact models, such as the nonlinear Hunt-Crossley (HC), and the linear Kelvin-Voigt (KV) to perform real-time parameter estimation of the environment.

While these methods are easy to implement, can be simulated efficiently and are able to handle large deformations, their major drawback as mentioned in [2], is the tedious modeling as there is no intuitive relation between spring constant and physical material properties. In [2],it is discussed how Finite Element Methods (FEMs) reflect physical properties of the objects in a more natural way [3]. This allows for more intuitive modeling since they require only a small number of parameters. The disadvantage of FEMs lies in the computational resources required to calculate deformations. A computationally more efficient approach, which is also used in [2], is the co-rotational finite element approach [4], [5] that avoids nonlinear computations.

Moreover, a lot of different approaches exist to determine the physical parameters of models. Bianchi et al. [6] learn the stiffness constants of mass-spring models by using a genetic algorithm and comparing it to a FEM reference model. The identification of mass-spring parameters is also discussed in the work of Lloyd et al. [7], where they derive an analytical formulation for the spring parameters from the linear finite element model for different mesh topologies. Another approach that estimates the stiffness properties of mass-spring models was proposed by Burion et al. [8]. They use a particle filter to obtain a posterior distribution over the stiffness parameters and evaluate the particles by comparing simulated and observed deformations [2].

A completely different approach to effectively identify unknown environments, is to employ artificial neural networks (ANNs).
In complicated real-world systems, deep neural networks (DNNs) have proven very effective for classification problems related with patterns in complicated systems such as image processing [14], speech processing [15], language models [16], handwriting recognition [17] and sequential data [18], [19]. These networks are termed

"deep" because they are constructed by stacking multiple layers of non-linear operations (such as NNs) atop one another with many hidden layers.
For real time system identification a deep dynamic neural network method was investigated in [9]. Neural networks are known to be effective function approximators. Recently, deep neural networks have proven to be very effective in pattern recognition, classification tasks and human-level control to model highly nonlinear real-world systems. This paper investigates the effectiveness of deep neural networks in the modeling of dynamical systems with complex behavior.

1.2.2 Force Control of Position/Velocity Controlled Robots

The robot force control literature reports two broad approaches for the control of robots executing constrained motion: impedance control and hybrid (force/position) control [20]. In impedance control, a prescribed static or dynamic relation is sought to be maintained between the robot end effector force and position [23]– [25]. In hybrid control, the end effector force is explicitly controlled in selected directions and the end effector position is controlled in the remaining (complementary) directions [26], [27].
Jaydeep Roy and Louis L. Whitcomb in [12] addresses the problem of achieving exact dynamic force control with manipulators possessing the low-level position and/or velocity controllers typically employed in industrial robot arms. Force control algorithms are being deployed for velocity/position controlled robot arms in contact with surfaces of unknown linear compliance.

Modelling of contact-rich tasks is challenging and cannot be entirely solved using classical control approaches due to the difficulty of constructing an analytic description of the contact dynamics.
In [21] the problem of force/position control for a robotic manipulator in compliant contact with a surface under non-parametric uncertainties is considered. A neuro-adaptive controller was used, that exploits the approximation capabilities of the linear in the weights neural networks, guaranteeing the uniform ultimate boundedness of force and position error with respect to arbitrarily small sets.
In [22] the authors present a data-driven control approach that employs a recurrent neural network to model the dynamics for a Model Predictive Controller for a manipulation task like food-cutting. This paper is an extension on previous work by incorporating Force/Torque sensor measurements and formulating the control problem so that it can be applied to the more common velocity controlled robots. The controller's performance is evaluated on objects used for training, as well as on unknown objects, by means of the cutting rates achieved and demonstrating that the method can efficiently treat different cases with only one dynamic model.

Finally, in [11] a Reinforcement Learning (RL) approach is discussed as optimal adaptive control when the dynamics are unknown. Reinforcement learning, is a sub-field of machine learning, used for optimization and control with a focus on continuous control applications. Reinforcement learning is the study of how to use past

data to enhance the future performance of a dynamical system. The goal of this paper is to find a sequence of inputs that drives a dynamical system to maximize some objective beginning with minimal knowledge of how the system responds to inputs.

1.3 Objectives and Limitations

Although manipulation and grasping of rigid objects has been studied extensively amongst the robotics community, the study of deformable objects has not been as extensive. This thesis focuses on identifying the dynamical model of a deformable object, in order to achieve exact dynamic force control under deformation, with a robotic manipulator possessing a low level velocity controller. In the initial stage, the UR 10 robotic manipulator will be used to interact with a sponge, in order to collect data based on proprioception and haptic perception. Subsequently, a deep neural network will be trained with this data, using supervised learning, and try to simulate the deformable object's dynamical behaviour. In the next stage of this thesis, this model will be used to design a model-based PI feedback controller with a feedforward term in order to track a force trajectory, while in contact with the deformable object. This thesis mainly focuses on tracking a force trajectory in the (x,y,z) plane, and thus the kinematics of the manipulator and the joint dynamics are not being investigated. Moreover, since the manipulator is moving inside a sponge, with infinite degrees of freedom (Guo et al. 2013), forces in one direction may result in small forces in other directions, diminishing the controllers credibility. For this reason, in the experiments, the controllers are evaluated while tracking a force trajectory in one direction at a time. The objectives of the thesis are to:

- Investigate the capabilities of artificial neural networks (ANNs) in identifying dynamical models of deformable objects.
- Assess the learnt object models regarding their applicability for real-time force control problems.
- Assess how the same problem can be addressed by learning-based control design such as Reinforcement Learning (RL), used for optimization and control with a focus on continuous control applications.
- Evaluate the control schemes based on experiment results.

1.4 Thesis Layout

This thesis report is structured four main Chapters. In Chapter 2 the necessary theoretical background of artificial intelligence and neural networks is presented along with some pre-processing techniques commonly used while training artificial neural networks. The basic theoretical knowledge of force control is presented in Chapter 3, along with Reinforcement learning as a method to design optimal controllers. Finally, in Chapter 4 the experimentation procedure is presented in detail, and it's results are discussed and assessed. The last Chapter 5, consists of some conclusions based on the findings during the experimental procedure along with some potential future work.

2

Data-Driven Function Approximation

In this chapter, the theory of neural networks will be discussed, as well as their application in model identification of soft/deformable objects. Since, the artificial neural network will be handling sensor data, which in most cases are noisy, also some pre-processing and filtering techniques will be discussed.

2.1 Artificial Neural Networks (ANN)

Machine learning is a branch of artificial intelligence that provides systems the ability to learn and improve from experience. The learning process begins with observations or data, examples, experience in order to look for patterns in data and make better decisions in the future based on the examples that provided. The primary aim is to allow the system to learn automatically without human intervention.
There are three types of machine learning algorithms :

- Supervised Learning (SL) algorithms, apply what has been learned in the past to new data using labeled examples to predict future events.
- Unsupervised Learning (UL) algorithms are effective when the data used to train the network are neither classified nor labeled.
- Reinforcement Learning (RL) algorithms is a learning method that interacts with its environment by taking actions and observing the rewards. This method allows machines and agents to automatically determine the optimal behaviour within a specific context in order to maximize its performance. The reward of an action is used as a feedback in order for the agent to learn which action is best. This is called the reinforcement signal.

In this thesis, machine learning is being used for two purposes. First, a deep neural network will be trained, using supervised learning, in order to approximate the deformation model of a soft object. Since we can collect data of the interaction with the real object and use them as the ground truth to train the artificial neural network, this can be classified as a supervised learning problem. Therefore, the rest of this chapter will be about supervised learning.
However, later in this thesis, reinforcement learning will be used to design an optimal artificial intelligent agent, that can on-line approximate the object's as well as the robot's model. Reinforcement learning is mostly used to design an optimal on-line controller, rather than just estimating a model offline and thus will be discussed in more detail in the next chapter.

In order to understand how this two methods can be used to train a system, it is imperative to first understand how artificial neural networks work.
What is a neural network? To understand how neural networks work, let's start from a type of artificial neuron called a perceptron. Perceptrons were developed in 1950s and 1960s by the scientist Frank Rosenblatt, inspired by earlier work by Warren McCulloch and Walter Pitts [29] [28].

2.1.1 The Perceptron

A perceptron is a linear classifier, more specifically, it is an algorithm that classifies input by separating two categories with a straight line. Usually the input to the neuron is a feature vector x, which is then multiplied by a weight vector w and added some bias b, resulting in :

$$y = wx + b \tag{2.1}$$

A perceptron produces a single output based on a vector of real-valued inputs by forming a linear combination using its input weights, and most commonly passing the output through a nonlinear activation function as follows :

$$y = g(\sum_{i=1}^{r_{k-1}} w_i x_i + b) = g(w^T x + b) \tag{2.2}$$

where w is the weight vector, x is the input vector, b is the neuron's bias and g is the nonlinear activation function. Weights show the strength of the particular node, while a bias value allows for a shift in the activation function curve up or down. By connecting more than one perceptrons together, we can build a type of artificial neural network called, multilayer perceptron.

2.1.2 The Multilayer Perceptron (MLP)

A multilayer perceptron (MLP) is a deep artificial neural network, composed of more than one perceptrons. The input layer is where the network receives the input vector, whereas the output layer makes a decision, or prediction based on the input. Between the input and output layer, there is a main computational engine of the MLP, consisting of the hidden layers.
Multilayer perceptrons are often applied to supervised learning problems.In particular they are trained on a set of input-output data in order to learn to model the correlation between the inputs and outputs. Training means adjusting the network's parameters, i.e the node weights and the neuron biases, of the model in order to minimize a cost function. An algorithm called back-propagation is used to make those weight and bias adjustments relative to the error, and the error itself can be measured in many different ways, like the root mean squared error (RMSE) or the Log-Cosh loss function.
Deep feedforward networks, also often called feedforward neural networks, or multilayer perceptrons (MLPs), are the quintessential deep learning models. The goal of a feedforward network is to approximate a function. For example, for a classifier, $y = f(x)$ maps an input x to a category y. A feedforward network defines a

mapping $y = f(x, \theta)$ and learns the value of the parameters θ that result in the best function approximation. These models are called feedforward because information flows through the function being evaluated from x, through the intermediate computations used to define f, and finally to the output y [10].

Forward Propagation

In the forward pass, the signal flow moves from the input layer through the hidden layers to the output layer, and the decision of the output layer is measured against the ground truth. For this, a loss function is used to estimate the loss (error) and to compare and measure how good or bad the networks prediction was in relation to the correct result. Since we are using supervised learning, we have the label that expresses the expected value.
In Fig. 2.3, a simple artificial neural network (ANN) is presented, with 3 inputs, and 1 output. This network will be used to establish some notation of the mathematical principles of forward and back propagation.

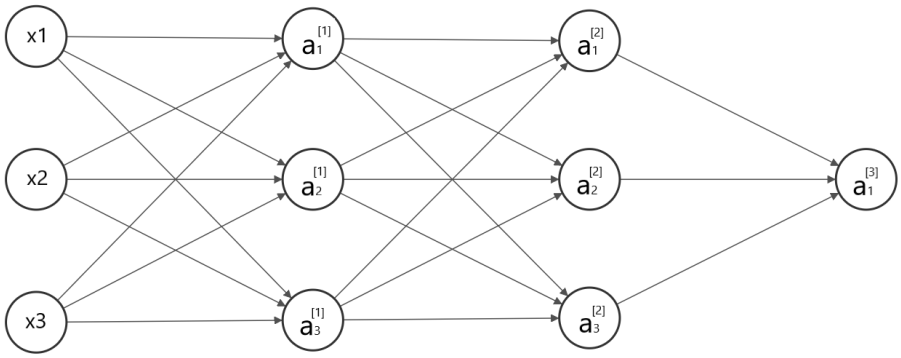


Figure 2.1: Multilayer perceptron Layout : An MLP with 3 inputs, 1 output, and 2 hidden layers

In the Network, $a_i^{[k]}$ denotes the activation of neuron number i in layer k
Assuming $x_i = a_i^{[0]}$ and $k > 0$ for each activation, of each neuron and with the help of equation (2.2) it holds that :

$$z_i^{[k]} = w_i^{[k]T} a^{[k-1]} + b_i^{[k]} = w_{i1}^{[k]} a_1^{[k-1]} + ... + w_{in^{[k]}}^{[k]} a_{n^{[k]}}^{[k-1]} + b_i^{[k]} \tag{2.3}$$

$$a_i^{[k]} = g^{[k]}(z_i^{[k]}) \tag{2.4}$$

where $g^{[k]}(\cdot)$ is the activation function for layer i, w_i^k is the weight vector and $b_i^{[k]}$ is the bias for neuron number i in layer k.

Back Propagation

In the backward pass, using back-propagation and the chain rule of calculus, the partial derivatives of the error with respect to the weights and biases are back-propagated through the MPL. By differentiating, one can obtain the gradient along which the parameters can be adjusted as they move the MLP one step closer to

minimizing the error. This can be done with any gradient-based optimization algorithm such as stochastic gradient descent or Adamax. The network keeps training until it can no longer lower the error between the expected output and the networks prediction. This state is known as convergence.
Let's assume a neural network with an input vector $\boldsymbol{x}_i$ and a vector of weights and biases $\boldsymbol{\theta} = [w_i^k,\ b_i^{[k]}]$ that produces an output $\hat{y}_i$. Also, assume a ground truth output vector y_i exists. Training a neural network with any gradient descent method requires the calculation of the gradient of an error function $E(\boldsymbol{x}_i, \boldsymbol{\theta}) = f(\hat{y}_i, y_i)$ with respect to the weights w_i^k and biases $b_i^{[k]}$. For simplification, the bias $b_i^{[k]}$ will be incorporated into the weights as w_{i0}^k with a fixed output of $a_0^{[k-1]} = 1$, [30]. With this formulation, equation (2.3) takes the form :

$$z_i^{[k]} = \sum_{j=0}^{r_{k-1}} w_{ij}^{[k]} a_j^{[k-1]} \tag{2.5}$$

The error function measures the difference between y_i, which is the target value for input-output pair $(\boldsymbol{x}_i, y_i)$ and $\hat{y}_i$ which is the computed output of the network on input $\boldsymbol{x}_i$. Back propagation algorithm attempts to minimize the error function with respect to the neural network's weights, by calculating for each weight $w_{ij}^{[k]}$, the value of $\frac{\partial E}{\partial w_{ij}^{[k]}}$ [30]. By applying the chain rule to the error function partial derivative we get:

$$\frac{\partial E}{\partial w_{ij}^{[k]}} = \frac{\partial E}{\partial z_i^{[k]}} \frac{\partial z_i^{[k]}}{\partial w_{ij}^{[k]}} \tag{2.6}$$

Moreover the partial derivative of equation (2.5), yields:

$$\frac{\partial z_i^{[k]}}{\partial w_{ij}^{[k]}} = a_j^{[k-1]} \tag{2.7}$$

Combining equations 2.6 and 2.7 together we get:

$$\frac{\partial E}{\partial w_{ij}^{[k]}} = \frac{\partial E}{\partial z_i^{[k]}} a_j^{[k-1]} \tag{2.8}$$

Starting from the final layer, and considering the network in figure 2.3, the partial derivative of the error function $E(a_j^{[k]}, y)$ with respect to the weight in the final layer $w_{ij}^{[k]}$ is given by :

$$\frac{\partial E}{\partial w_{ij}^{[k]}} = \frac{\partial E(a_j^{[k]}, y)}{\partial a_j^{[k]}} g_0'(z_j^{[k]}) a_i^{[k-1]} \tag{2.9}$$

where $a_j^{[k]}$ is the output of the node j of the final layer k, and $j = 1$, g_0 is the activation function of the final layer and E is the loss function that the algorithm is trying to minimize.
For the partial derivatives of the hidden layers, the chain rule of the multivariate functions is applied. For $1 \leq k < m$ where m is the final layer the error propagated in each hidden layer can be calculated as follows:

$$\frac{\partial E}{\partial z_j^{[k]}} = \sum_{n=1}^{r^{k+1}} \frac{\partial E}{\partial z_n^{[k+1]}} \frac{z_n^{[k+1]}}{z_j^{[k]}} \tag{2.10}$$

where n ranges from 1 to $r^{[k+1]}$, the number of nodes in the next layer. Finally, the partial derivative of the error function E with respect to the weights in the hidden layers $w_{ij}^{[k]}$ for $1 \leq k < m$ becomes :

$$\frac{\partial E}{\partial w_{ij}^{[k]}} = g'(z_j^{[k]}) a_i^{[k-1]} \sum_{n=1}^{r^{k+1}} w_{jl}^{[k+1]} \frac{\partial E}{\partial z_n^{[k+1]}} \tag{2.11}$$

The backpropagation algorithm consists of 4 steps, assuming a suitable learning rate a and a random initialization of the networks weights w_{ij}^k :

1. Calculate the forward phase for each input-output pair $(\boldsymbol{x_i}, y_i)$ and save the resulting outputs $\hat{y}_i, z_j^k$ and a_j^k for each node j in layer k by going from the input layer (layer 0), to the output layer (layer m).
2. Calculate the backward phase for every input-output pair and store the resulting error derivatives $\frac{\partial E}{\partial w_{ij}^k}$ for each node connecting neuron i in layer $k-1$ to neuron j in layer k by going (backwards), from layer m (output layer), to layer 0 (the input layer). This happens in three steps:
   (a) By using equation (2.9), the partial derivative of the error function of the final layer is being evaluated.
   (b) Backpropagate the gradient of the error for the hidden layers working backwards from the final layer $k = m - 1$, using equation (2.11).
   (c) Finally, evaluate the partial gradients of the individual errors with respect to the weights.
3. Combine the individual gradients for each input-output pair to get the total gradient for the entire set $\boldsymbol{X} = (\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_n, y_n)$, by averaging the individual gradients.
4. In the final step, the weights are updated according to the learning rate a and the total gradient, by moving in the direction of the negative gradient [30].

In the description of the basic training process of an artificial neural network, some parameters of the network were absent from the mathematical model presented. These parameters being, the activation functions of each layer (g), the loss function ($E = f(\hat{y}_i, y_i)$), which is a function of the error between the network's estimation and the ground truth, and the optimization algorithm that tries to minimize this loss function. The reason these three parameters were not included in the training description is that they depend on the application, and more specifically the model we are trying to estimate. Some functions work better than others, and this is why a clear description of the functions used in this thesis project will be presented bellow. First, let's define the activation function of a neuron, and why they are important for function estimation applications.

2.1.3 Softsign Activation Function

An activation function is used to map the input of a neuron between the required values such as $(0, 1)$ or $(-1, 1)$. The activation functions can be divided into two types:

- Linear Activation Functions
- Non Linear Activation Functions

The most used activation functions used in deep machine learning problems are the non linear activation functions. But, why do we need Non-Linearities?
If no activation function is applied, then the output signal would be just a simple linear function. Linear equations are easy to solve, but they have limited complexity and they lack in power to learn complex functional mappings from data.
Non-linear functions introduce non-linear properties to the network, making it easier to generalize or adapt with a variety of data and be able to differentiate between the output. By using nonlinear activation functions, the network can learn and represent functions of arbitrary complexity which maps inputs to outputs. Artificial neural networks are considered Universal Function Approximators, which makes them ideal for the purpose of this thesis project.
Another important attribute of an activation function is that it has to be differentiable. As shown in the previous section, in order to perform the backpropagation optimization strategy, the gradients of the loss function have to be computed, with respect to the weights and then accordingly optimize them using any optimization technique to reduce the error [34].
In this thesis project, the softsign activation function will be used. Softsign activation function is closely related to the hyperbolic tangent function, with the only difference that softsign converges polynomially, whereas tanh converges exponentially [32]. The softsign function is a nonlinear equation of the form:

$$y = \frac{x}{1 + |x|} \tag{2.12}$$

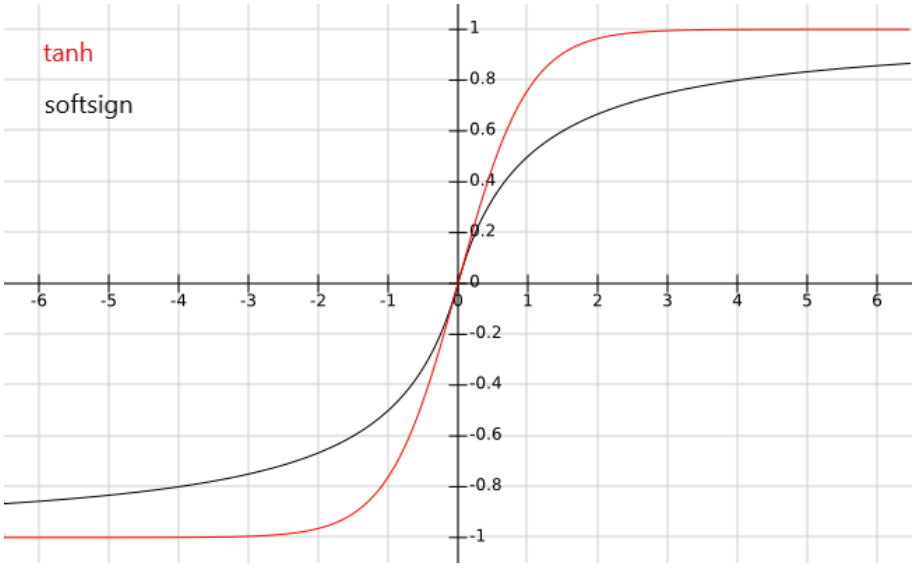


Figure 2.2: Softsign and tanh functions produce outputs between $[-1, +1]$

It can be observed that the softsign activation function is smoother than tanh. This can result in better and faster training, due to the fact that softsign activation function can prevent the neurons from being saturated, resulting in more effective training.

As stated before, the function's partial derivative needs to be computed in order to be used in the backpropagation process.

$$\frac{dy}{dx} = \frac{x'(1+|x|) - x(1+|x|)'}{(1+|x|)^2}$$

where the derivative of $|x|$ can be computed as follows :

$$|x|' = \frac{x}{|x|}$$

Combining these equations, the derivative of the activation function softsign is derived :

$$\frac{dy}{dx} = \frac{1}{(1+|x|)^2}$$

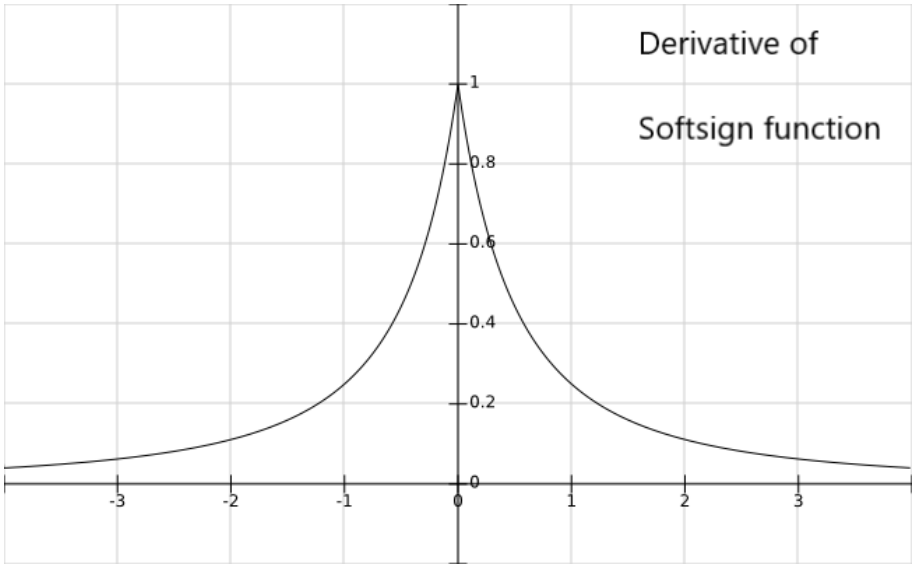


Figure 2.3: Derivative of the Softsign Activation Function

2.1.4 Log-Cosh Loss Function

All algorithms in machine learning rely on minimizing or maximizing an objective function. The function to minimize are called loss functions. A loss function measures how good a prediction model does in being able to predict the expected outcome. The most commonly used method of minimizing a function is the gradient descent optimization strategy.
The choice of a loss function, is highly dependant on the dataset the network is training on, as well as the problem the network is trying to solve. Loss functions can be categorized into two types [33] :

- Classification Loss
- Regression Loss

Classification loss functions predict a label, whereas regression loss functions predict a quantity. For the purpose of this thesis project, a network is needed that outputs

a predicted quantity, in other words, a function approximation, and thus, the focus will be on regression loss functions.
More specifically, the loss function used in this thesis project was the Log-Cosh Loss function. The Log-Cosh is a loss function used in regression tasks, and is a smoother version of the Mean Square Error (MSE) loss function.
Log-Cosh is the base e logarithm of the hyperbolic cosine of the prediction error [33].

$$E(\hat{y}_i, y_i) = \sum_{i=1}^{n} \log_e \left(\cosh\left(\hat{y}_i - y_i\right)\right)$$

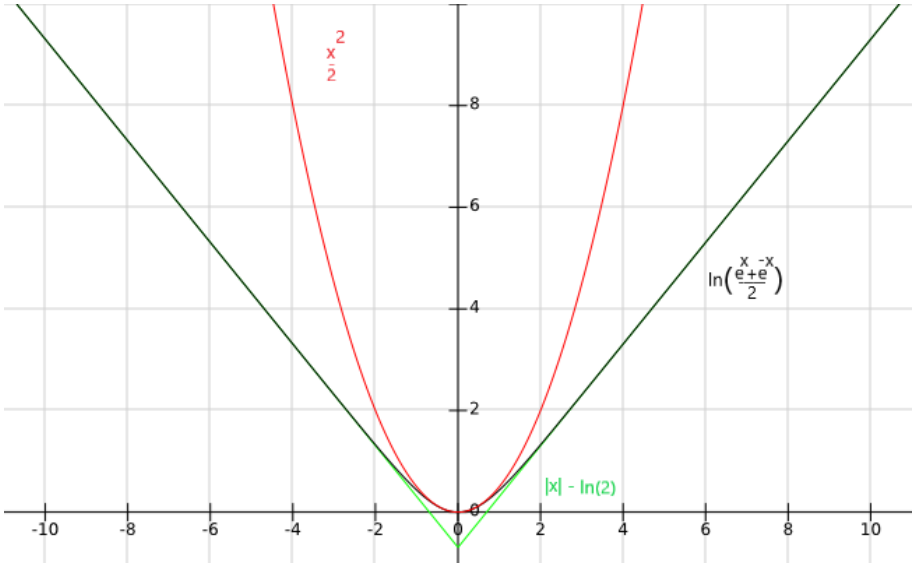


Figure 2.4: Plot of Log-Cosh Loss Function, $(\frac{x^2}{2})$ and $(|x| - ln(2))$

As it can be observed in figure 2.4, the $(\log_e(\cosh(x)))$ function is approximately equal to $(\frac{x^2}{2})$ for small x and $(|x| - ln(2))$ for large x. This means that Log-Cosh function works mostly like the Mean Squared Error, but will not be strongly affected by the occasional wildly incorrect prediction. Moreover, it is twice differentiable everywhere [33], and it's second derivative is always greater than zero. Due to this, the Log-Cosh loss function can not have local minimum, since it cannot have any maximum, which means that any gradient based optimization technique will always lead to the global minimum.
During the training process, the parameters (weights) of the model are tweaked, in order to minimize the loss function, and therefore make the model's predictions as correct as possible. But how exactly do the parameters of the model change, by how much, and when? This questions are addressed by the choice of the optimizer.

2.1.5 Adamax Optimizer

An optimizer ties together the loss function and the model parameters by updating the model in response to the output of the loss function. The loss function acts like the guide, informing the optimizer when it is moving in the right or wrong

direction. Stochastic gradient-based optimization is of practical importance in many fields of engineering and science. A lot of problems in these fields, as well as the problem being dealt in this thesis, can be described as the optimization of same scalar parameterized loss function requiring minimization with respect to its parameters. For this purpose the Adamax optimization algorithm will be used, which is a variant of Adam, based on the infinity norm [31]. The adaptive moment estimation method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. This method is designed to combine the advantages of two vastly popular methods: AdaGrad [35], which works well with sparse gradients, and RMSProp [36], which works well in on-line and non-stationary settings. Some of Adam's most important advantages are that the magnitudes of parameter updates are invariant to re-scaling of the gradient, the step sizes are approximately bounded by the step size hyper-parameter, it does not require a stationary loss function, it works with sparse gradients, and it naturally performs a form of step size soothing [31].
In Adam the update rule for individual weights is to scale their gradients inversely proportional to a scaled L^2 norm of their individual and past gradients. The Adamax variance of the algorithm, is based on a L^p norm update rule. Let, in case of L_t^p norm, the step size at time t be inversely proportional to $u_t^{1/p}$, where:

$$u_t = \beta_2^p u_{t-1} + (1-\beta_2^p)|g_t|^p = (1-\beta_2^p)\sum_{i=1}^{t}\beta_2^{p(t-i)}|g_i|^p$$

such that:

$$u_t = \lim_{p\to\infty}(u_t)^{1/p}$$

Combining these two equations yields:

$$u_t = \lim_{p\to\infty}((1-\beta_2^p)\sum_{i=1}^{t}\beta_2^{p(t-i)}|g_i|^p)^{1/p}$$

$$= \lim_{p\to\infty}(\sum_{i=1}^{t}\beta_2^{p(t-i)}|g_i|^p)^{1/p}$$

$$= max(\beta_2^{(t-1)}|g_1|, \beta_2^{(t-2)}|g_2|, ..., \beta_2|g_{t-1}|, |g_t|)$$

$$u_t = max(\beta_2 u_{t-1}, |g_t|)$$

with initial value $u_0 = 0$ and $g_t = \frac{\partial E}{\partial w_{ij}^{[k]}}$, as also shown in equation (2.7) we denote the gradient [31]. More specifically, the vector of partial derivatives of the loss function E_t, with respect to the weights $w_{ij}^{[k]}$ evaluated at time-step t, while $\beta_2 \in [0, 1)$ is the hyper-parameter that controls the exponential decay rate of the moving average.

In this section, we reviewed theory on artificial neural networks, as well as how they work and learn from data, using supervised learning. Moreover, all the parameters of the network were discussed and explained: *a*) the activation functions of the neurons, *b*) the loss function, and *c*) the optimizer. The focus was to present the parameters used for the specific task of this master thesis, which is to approximate

a function in order to model the dynamics of a deformable object. The choice of activation functions, loss function, and optimizer was decided after experimentation with different functions, in order to find the best fit for the specific task, and the specific data.

2.2 Pre-processing

It was discussed before that supervised learning allows for a set of collected data to be used to optimize an artificial neural network's future performance. Therefore, this machine learning method is able to solve various types of real-world computation problems.
However, in order to achieve these tasks there are some challenges that supervised learning algorithms may struggle with :

- Irrelevant input feature present in the training set can produce inaccurate results.
- Data preparation and pre-processing can be a challenge, but necessary.
- Accuracy may suffer when incomplete, or unlikely values have been inputted in the training set.
- Choosing the right features to train the network on, can be a challenge. A "1 to 1" mapping must exist from the input features to the output, otherwise the machine cannot approximate a function for this problem.

In order to tackle these challenges, in this chapter the pre-processing methods used in this thesis project will be presented, in other words how the data were manipulated before sending them into the artificial neural network for training.

2.2.1 Butterworth Low-pass Filter

In most supervised learning problems, the data to train the neural network (training set) needs to be collected in the real world. More specifically, for this project's task, a robotic manipulator was used to interact with the deformable object and collect the position of the end effector of the robot, as well as the force exerted to it. More detail will be given in Chapter 4.
However, it is important to note here that in order to do so, the robot's force sensor and position sensor were used. Therefore, the measurements are bound to contain at least some amount of sensor noise, which will make the features not entirely accurate, making the machine as a whole less accurate. Moreover, it was observed that the sensor data, has high frequency noise. For this reason, the inputs to the network will be filtered by a low-pass filter.
A low-pass filter is a filter that can be designed to modify, reshape or reject all unwanted high frequencies of a signal and allow low frequency component to pass. In applications that use filters to shape the frequency of a signal such as in control systems, the width or shape of the roll-off , called the "transition band", for a first order filter may be too long or wide and so active filters with more than one "order" are required. The complexity or the filter type is defined by the filter's order, which also affects the rate of roll-off and therefore the width of the transition band [37].

For the purpose of filtering the sensor data for this thesis project, the Butterworth low-pass filter will be used. The frequency response of the Butterworth filter approximation function is often referred to as "maximally flat" (no ripples) response because the pass band is designed to have a frequency response which is as flat as mathematically possible from $0Hz$ until the cut-off frequency with no ripples. However, one main disadvantage of the Butterworth filter is that it achieves this pass band flatness at the expense of a wide transition band as the filter changes from the pass band to the stop band. It also has poor phase characteristics as well [37].

The frequency response of Butterworth filters of different orders are shown in figure 2.5.

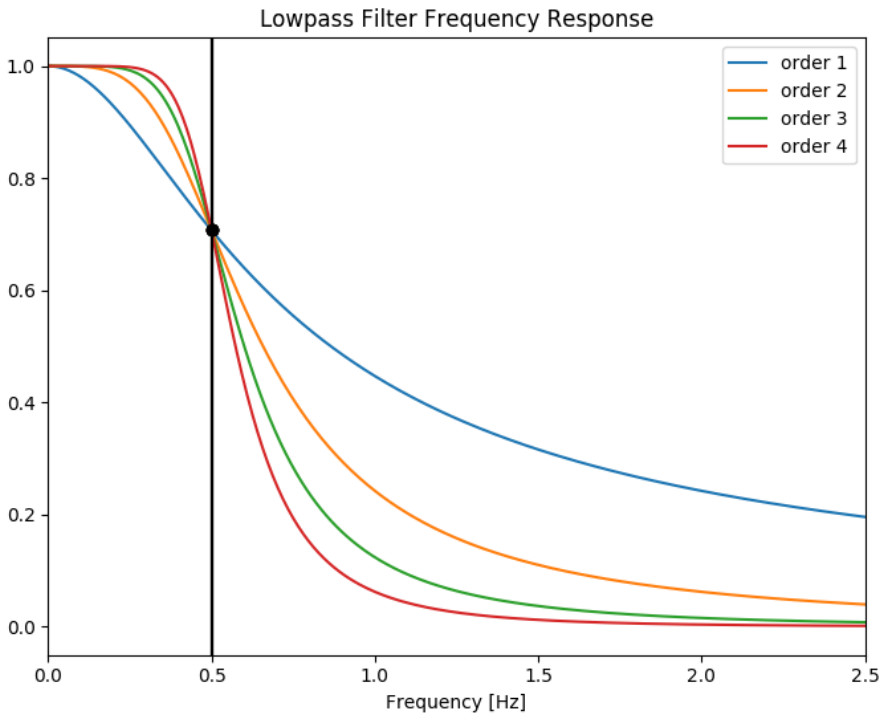


Figure 2.5: Magnitude- squared characteristic of the normalized Butterworth low-pass filter

Higher-order filters achieve better filtering characteristics, but the settling time is increased.Therefore, for on-line real life applications, a good trade-off between the transient response and the required attenuation is required. The second-order Butterworth filter offers an appropriate relation between the transient response and the required attenuation characteristic.

The first stage of the pre-processing process, was to filter the unwanted sensor noise from the inputs of the neural network. This way, the features are more accurate and consistent, which can make a huge difference in the network succeeding to estimate the intended model. The next stage of the pre-processing process will be the standardization of the features, which is needed to make sure that the features have the same units, and are of the same scale.

2.2.2 Standardization

Standardization of a dataset is a common requirement for many machine learning estimators. The estimator might behave badly if the individual features do not look like standard normally distributed data (e.g. Gaussian with 0 mean and unit variance). In practice, the shape of the distribution is often ignored and the transformation just happens such that the data are centered by removing the mean value of each feature, then scaled by dividing non-constant features by their standard deviation.
For instance, many elements used in the objective function of a learning algorithm assume that all features are centered around zero and have variance in the same order. If a feature has a variance that is orders of magnitude larger than others, it might dominate the objective function and make the estimator unable to learn from other features correctly as expected.
Data standardization is about making sure that data is internally consistent, meaning that, each data type has the same content and format. Suppose for example, that the input features to an artificial neural network have the form presented in Figure 2.6:

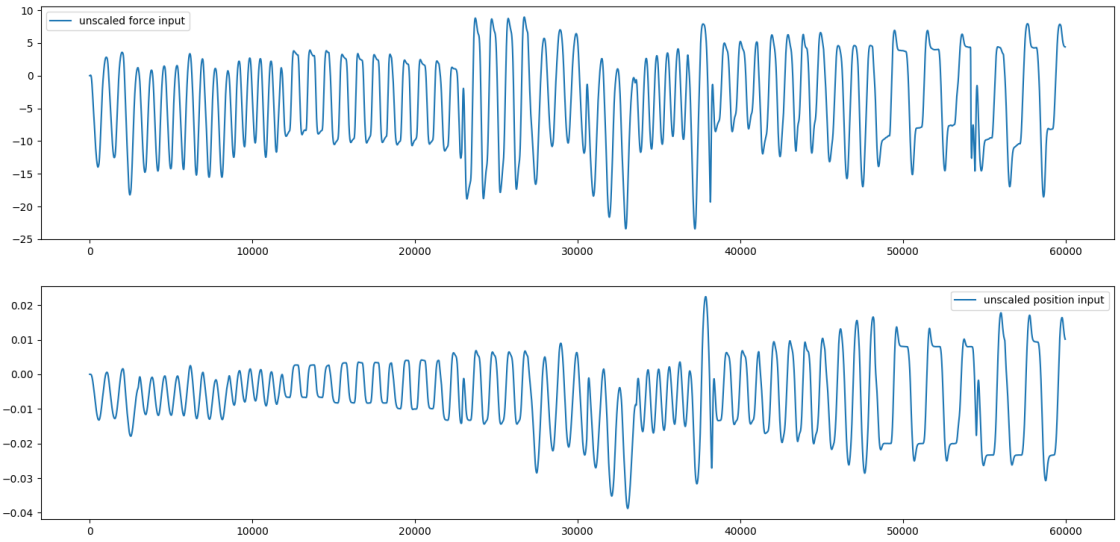


Figure 2.6: Unscaled input features to an artificial neural network

It can be observed in Figure 2.6 that the order of magnitude of the force feature is immensely larger than the position feature. If these data are used as inputs in an artificial neural network, the force feature will dominate the objective function and make the estimator unable to learn from the other features correctly as expected.
This is why standardization of the input features is required, by removing the mean and scaling to unit variance. The standard score of a sample x is calculated as:

$$z = \frac{x - u}{s}$$

where u is the mean of the training samples, and s is the standard deviation of the training samples. Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the dataset.

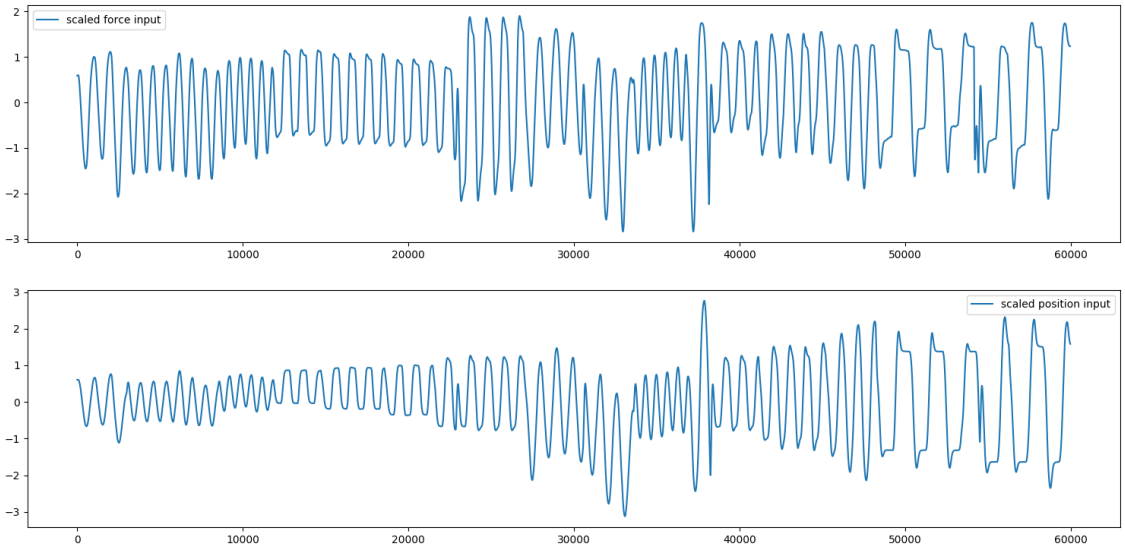


Figure 2.7: Scaled input features to an artificial neural network

The resulting scaled features can now be used as inputs to an artificial neural network, which will try and approximate a function based on these features.

3

Control System Design

In this chapter a background in control theory will be established, with some of the most commonly used control schemes, like P-Only (P) and Proportional-Integral (PI) controllers. Moreover, a background on force control, and its usefulness in robotic applications will be established. Finally, a machine learning method, called Reinforcement Learning will be analyzed, as well as its application and utilization in control applications in order to design optimal controllers.

3.1 Proportional Integral Control (PI)

The Proportional-Integral (PI) algorithm, like the P-Only controller, computes and transmits a controller output signal at every time step T. The computed control signal from the PI algorithm is influenced by the controller tuning parameters and the state error $e(t)$, as follows in (Figure 3.1).

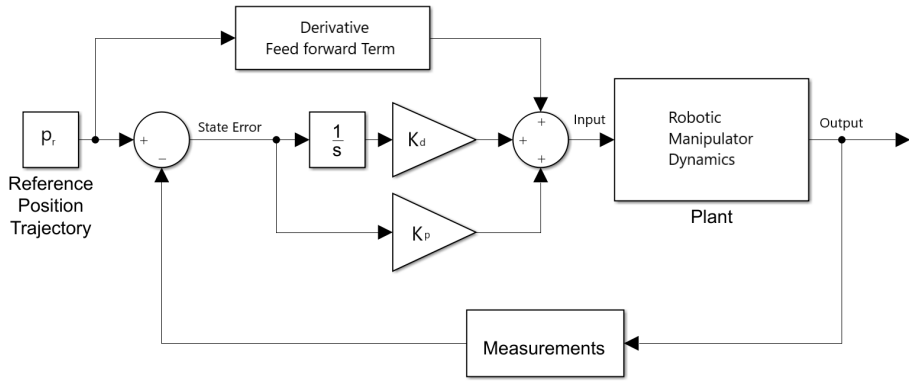


Figure 3.1: The PI controller formulation with a feed-forward term : Controller Output $= K_p e(t) + K_d \int e(t) + \dot{p_r}$

$$u = \dot{p_r} + K_p e(t) + K_d \int e(t) \tag{3.1}$$

where $\dot{p_r}$ is the feed-forward term, also referred to as controller bias, $e(t) = p_r(t) - x(t)$ is the current error between the reference ($p_r(t)$) the measured state ($x(t)$), and K_p and K_d are the controller's gains (Tuning Parameters).
The first two terms on the right side of equation (3.1) are identical to the P-Only controller. The integral term of the controller is the last term of the equation that

integrates or continually sums (in the discrete case) the controller error, $e(t)$, over time.
While the proportional term considers the current $e(t)$ only at the time of the controller calculation, the integral term considers the history of the error. Integration of the error over time means that we sum up the complete controller error history up to the present time, starting from when the controller was first switched on.
The integral action I enables the PI controller to eliminate offset, a major weakness of a P-only controller. Hence, PI controllers provide a balance of complexity and capability that makes them by far the most widely used algorithm in process control applications [44].
For the control part of this thesis project, a P-only controller as well as a PI controller scheme will be evaluated in a force control task.

3.2 Force Control

Controlling the physical interaction between a robotic manipulator and the environment is crucial for the successful execution of a plethora of practical tasks where the robot's end effector has to manipulate an object or perform some operation on an unknown object [40]. Today's industrial robots are almost always programmed using a position control scheme. Typically, the end effector follows a designated trajectory in space which has been pre-defined or "taught" before run-time. However, for some applications, it is more important to precisely control the force applied by the end-effector rather than controlling the robot's position [38]. More specifically, during contact with deformable surfaces with infinite degrees of freedom (Guo et al. 2013), there may be constraints on the paths that can be followed by the robot's end effector. In order to overcome these constraints, compliance was introduced either in a passive or in an active fashion, to accommodate the robot motion in response to interaction forces. Compliance can be classified into active and passive compliance. Active compliance mechanism is a closed-loop system, which the robot responds to the environment according to the sensory information. In contrast, passive compliance is an open-loop system, which the robot responds to the environment based on the body regulatory mechanisms, such as spring-damper mechanism, adjustable joint, and link stiffness [63].

- Active Compliance. It is entrusted to the control system, denoted interaction control or Force Control. In most cases, the measurement of the contact force and moment is required, which is fed back to the controller and used to modify or even generate online the desired motion of the robot [41]. For this purpose, vision, force, torque, and other similar type of sensors are usually used. This family of compliance methods is aimed for better flexibility in the manufacturing process [42].
- Passive Compliance. It is applied during the setup of the robotic cell and will always stay active in the background to fulfill its safety role. It may be inherent to the structure of the robot, like a torque limitation device on the end effector or a torque limitation on the joints. It can also be a form of collision detector that prevents collisions from occurring or prevents them from being harmful. These compliance devices aim at safety ensuring and

are important when human-robot collaboration are intended [42]. However, the use of passive compliance in industrial applications lacks flexibility, since for every task a single purpose compliant end-effector has to be designed and mounted. Also, it can only deal with small position and orientation deviations of the programmed trajectory. Finally, since no forces are measured, it can not guarantee that high contact forces will never occur [43].

One of the most common applications of force sensing in research is to implement Force Control. Since the 1970s, researchers have developed different ways to integrate the sense of touch into their robots. At first glance, the basic idea behind force control is simple: the output of a force/torque sensor is used to close the loop in the controller, adjusting each of the joint torque, or velocities to match the desired output. In a certain way, this is similar to position control by simply replacing the reference position/velocity (from the motor encoders) with a reference force (from the force/torque sensor). Unfortunately, it is not always as easy as that. The problems arise from the fact that the robot needs to perform a trajectory in certain directions while a precise control of the force is required in other directions. For example, when a robot is grinding a surface, a precise force is needed in the direction perpendicular to the grinded surface. In all other directions (and orientations), the robot needs to perform a standard, programmed position trajectory. Therefore, in real applications, force control is in fact a hybrid force/position control for which the joint torques are computed using two references. On top of that, the control scheme needs to be dynamically changed between two operations, for example when the robot completes grinding the surface of the object [38]. This qualifies force controllers more suitable for a variety of tasks like:

- Applying controlled force to an object (e.g pushing an object)
- Dealing with geometric uncertainty in assembly
- Avoiding high forces applied to the environment for safety reasons [39].

In general, six force components are required to provide complete contact force information. More specifically, three translational force components and three torques. Often, a force/torque sensor is mounted at the robot wrist and measures any force applied to the end effector (an example is presented in figure 3.2).

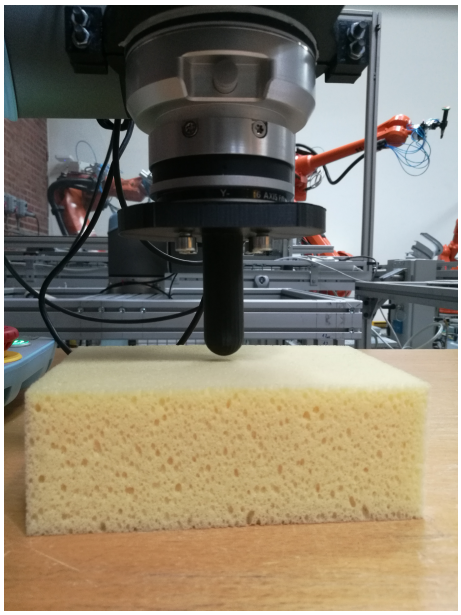


Figure 3.2: The UR10 Robot with the force/torque sensor and the deformable object from the experimental setup

The most basic type of motion control is pure position control, whilst the most basic type of force control is pure force control.

- Pure Position Control :
    1. A command to the robot's joints or end effector position to a particular position.
    2. The manipulator tries to reach that position.
    3. If it cannot reach the point (collision with something) it will continue to apply a stronger and stronger force on the environment until it does.
- Pure Force Control :
    1. A command to the robot's end effector to apply a certain force to the environment.
    2. The manipulator tries to apply that force.
    3. If the robot does not apply a force (i.e moving in free space) it will keep moving in the same direction until the target force is reached.

In practice, force control is usually integrated into robots by using a combination of motion and force control. However, for the purpose of this project, the robotic manipulator was always in contact with the deformable object, and thus a pure force control technique was used. Moreover, in order to track a force trajectory on the deformable object, an active compliance or Force Control scheme was designed. Consequently, the focus will be on force control strategies.
However, active compensation (Force control) can be grouped into two categories: *a*) Indirect force control, *b*) Direct force control. The main difference between the two categories is that indirect force control achieve force control via motion control, without explicitly closing a force feedback loop, whereas direct force control offers the possibility of controlling the contact force and moment to a desired value, by directly closing a force feedback loop [40].

An indirect force control technique was used in the early stages of the experimentation, in order to build a compliance controller to follow a position trajectory while in contact with the unknown deformable environment, and collect data to train the artificial neural network and identify the unknown environment. In the next stage, a direct force control scheme was implemented to track a force trajectory in contact with the now known environment.

3.2.1 Indirect Force Control (Compliance Control)

The first category, indirect force control methods, include impedance control, where the deviation of the end effector motion from the desired motion due to the interaction with the environment is related to the contact force through a mechanical impedance with adjustable parameters. A special case of impedance control is the compliance control, where only the static relationship between the end-effector position and orientation deviation from the desired motion and the contact force and moment is considered. Indirect force control schemes do not require, in principle, measurements of contact forces and moments. The resulting impedance is typically nonlinear and coupled. However, if a force/torque sensor is available, then force measurements can be used in the control scheme to achieve a linear and decoupled behavior [43].
There are some problems arising while the end-effector of a robot manipulator is interacting with the environment. In order to address those, it is worth analyzing the effects of a motion control strategy in the presence of a contact force and moment. Assume, that the robotic manipulator's end effector position in the (x, y, z) plane can be denoted as p, and the desired trajectory p_r. Furthermore, let $\dot{p_r}$ denote the reference velocity of the manipulator's end effector in the (x, y, z) plane and let V be the controller output, and the velocity commanded to the manipulator. Also, assume that the force exerted to the end effector from the environment and measured from the force/torque sensor is denoted as F.
Consider the following velocity control law :

$$V = \dot{p_r} + K_p(p_r - p) + aF \tag{3.2}$$

corresponding to a simple P-only controller with a feed-forward term, along with a compliance term where a is a gain that maps the external forces to the robot end effector position.
Equation (3.2) shows that in the steady state the end effector, under a proportional P control action on the position error, behaves as a spring in respect of the external force F. Thus, the gain K_p plays the role of an active stiffness, meaning that it is possible to act so as to ensure a suitable elastic behavior of the end effector during the interaction. The last term of the equation represents a compliance relationship, where the gain a is called the sensitivity gain and maps the external forces to the robot end effector position. This approach, consisting of assigning a desired position and orientation and a suitable static relationship between the deviation of the end effector position from the desired force exerted from the environment, is known as stiffness control, or compliance control.
The selection of the stiffness/compliance (K_p, a) parameters is not easy, and strongly depends on the task to be executed [43].

3.2.2 Direct Force Control

In the previous section, an indirect control of the contact force has been achieved by suitably controlling the end effector position. In this way, it is possible to ensure limited values of the contact force for a given rough estimate of the environment stiffness. It has also been emphasized that even compliance control, which does not aim at achieving a desired force, needs contact force measurements to obtain the linear and decoupled end effector dynamics during the interaction. This is desirable in order to realize a compliant behavior only along those task directions that are actually constrained by the presence of the environment. Certain interaction tasks, however, do require the fulfillment of a precise value of the contact force. This would be possible, in theory, by tuning the active compliance control action ($\frac{K_p}{a}$), and selecting a proper desired location for the end effector. This strategy would be effective only on the assumption that an accurate estimate of the contact stiffness is available. Therefore, contact force measurements are fully exploited hereafter to design direct force control [45].
Direct force control approach, operates on a force error between the desired and the measured values. The implementation of a force control scheme can be ensured by closing an outer force control loop, that generates the reference input to the velocity controlled robotic manipulator. Let F_d denote the desired contact force and let the force error for each time step t be:

$$\Delta f = F_d - F$$

where F is the force acted on the end effector from the environment, measured by the force/torque sensor and F_d the desired contact force. Then, the velocity input to the robotic manipulator in order to control the exerted force can be chosen as a Proportional-Integral (PI) control with a feed-forward term, on the force error :

$$V = V_{ref} + K_p \Delta f + K_d \int_0^t \Delta f dt \tag{3.3}$$

where K_p and K_d are positive gains, and the tuning parameters of the PI controller. It is important to note that $K_p(F - F_d)$ is also known in the bibliography as inverse damping, and it is strongly related to the implementation of direct force control on velocity controlled robots. The tuning of the controller's parameters in crucial, and highly dependent on the application, and the force trajectory the manipulator has to track. Assuming that the desired contact force is assigned such that it has non null components only along the constrained task direction, the PI based control scheme ensures:

$$\lim_{t \to \infty} F(t) = F_d$$

thanks to the use of the integral control action on the force error, considering that V_{ref} is appropriately defined. Moreover, the proportional actions on the force error aims at improving the transient behaviour during the interaction.

This thesis project's objective is to evaluate control methods on tracking force trajectories on deformable objects, and thus mostly direct force control will be used.

However, direct force control is highly dependent on the gain tuning of the controller parameters. This means that in order for the robot to track a different force trajectory, the controller needs to be tuned once again. To overcome this issue, an online "learning" algorithm will be introduced that uses past experience to decide the optimal control action at every time step.

Most optimal control approaches are tuned offline and require complete model knowledge. Even for linear systems, where the linear quadratic regulator (LQR) gives the closed-form analytical solution to the optimal control problem, the algebraic Riccati equation (ARE) is solved offline and requires the exact knowledge of the system dynamics. Adaptive control provides an approach to design controllers which can adapt online when the system dynamics are unknown, based on minimization of the output error, like using gradient or least squares methods. However, classical adaptive control methods do not maximize a long-term performance function, and hence are not optimal. Adaptive optimal control refers to methods which learn the optimal solution online for uncertain or unknown systems. Reinforcement Learning (RL) methods have been successfully used in Markov decision processes (MDPs) to learn optimal polices in uncertain environments, e.g TD-based Value function learning is an online model-free RL method for learning optimal policies. Due to the discrete nature of RL algorithms, many methods have been proposed for adaptive optimal control of discrete-time systems. Unfortunately, a RL formulation for continuous-time systems, like the force control of a robotic manipulator is not as straightforward as in the discrete-time case. Other issues concerning RL-based controllers are: closed-loop stability, convergence to the optimal control, function approximation, and tradeoff between exploitation and exploration [61].

A model-free solution will be investigated by using an actor-critic architecture, for a continuous time system.

3.3 Reinforcement Learning

Reinforcement learning is the study of how to use past data to enhance the future performance of a dynamical system. How does this differ from ordinary machine learning? The main use of reinforcement learning in this project is to design an optimal controller when the system (environment) dynamics are unknown.

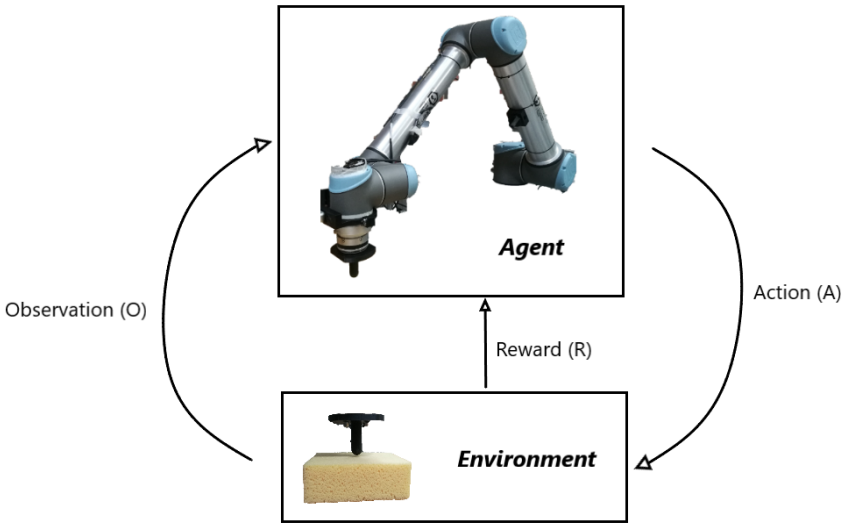


Figure 3.3: Reinforcement Learning Setup : The Agent is the 6 DoF UR10 robotic Manipulator, and the environment is a deformable object.

The goal of Reinforcement Learning (RL) algorithms is to find a sequence of inputs that drives a dynamical system to maximize some objective function, beginning with minimal knowledge of how the system responds to inputs. In any classic optimal control problem, the dynamical system is usually governed by a difference equation:

$$x_{t+1} = f_t(x_t, u_t, e_t)$$

where x_t is the state of the system, u_t is the control action, e_t is a disturbance, and f_t is the rule that maps the current state, control action, and disturbance at time t to a new state. Assume that at every time step t, corresponds some reward $\rho(x_t, u_t) = r_t$ for the current x_t and u_t. The main purpose of the reinforcement learning algorithm is to maximize/minimize this reward, depending on the problem formulation. In terms of mathematical optimization, the objective is to solve the problem :

$$\begin{aligned} \underset{u_t}{max/min} \quad & \mathbb{E}_{e_t}[\sum_{t=0}^{N} \rho[x_t, u_t]] \\ subject\ to \quad & x_{t+1} = f_t(x_t, u_t, e_t) \end{aligned} \tag{3.4}$$

That is, maximize/minimize a reward function over N time steps with respect to the control sequence u_t, subject to the dynamics specified by the state-transition rule f_t. The expected value is over the disturbance e_t, and assumes that u_t is to be chosen having seen only the states x_0 through x_t and previous inputs u_0 through u_{t-1}. r_t is the reward gained at each time step and is determined by the state and control action. Note that x_t is not really a decision variable in the optimization problem, instead it is determined entirely by the previous state, control action, and disturbance. As a trajectory, τ_t, will be considered a sequence of control actions and states generated by a dynamical system [11].

$$\tau_t = (u_1, ..., u_{t-1}, x_0, ..., x_t)$$

However, in some applications the state-transition rule f is unknown as for example, when a robotic manipulator is interacting with a deformable object of unknown dynamics. In this case, there is no known model that connects the next state of the system x_{t+1} with the current state x_t and control action u_t. Therefore, it is important first to acquire knowledge about the dynamical system and subsequently choose the best policy. To achieve this, a policy π and an horizon of length N is decided. Then this policy is used as a control input into a real physical system (e.g UR10 Robotic Manipulator), which results in a trajectory τ_N and a sequence of rewards $R(x_t, u_t)$. The intention is to find a policy that maximizes/minimizes the reward with the fewest total number of samples computed by the agent [11].
In the case where the state-transition function (model) is unknown, there are two widely used methods.

- Model Based Reinforcement Learning methods fit a model to previously observer data and then uses this model to approximate a solution to the problem expressed in (3.4).
- Model Free Reinforcement Learning directly seeks a mapping from observations to actions. Model-free methods, aim to solve optimal control problems only by taking an action, observing the result of that action and improving the strategies based on that observation [11].

In the rest of this chapter, the focus will be in model free reinforcement learning methods, and some different approaches will be introduced.

3.3.1 Model-Free Reinforcement Learning

Model Free Reinforcement learning is a machine learning method, where an agent (controller) is trying to optimize its behaviour only by interacting with its environment. For each action from a specific state, the agent receives a reward from the environment, which is an indication of the quality of that action. The function that specifies which action to take from a certain state is called the policy. The objective of the agent (controller) is to find the policy that maximizes the total accumulated reward, also called the return. While following a certain policy and processing the rewards, the agent is building estimates of the return. These estimates are condensed into a value function, which is defined as the expected return. The agent uses the value function to acquire information about past experiences and use them to decide on future actions from a certain state [47]. Over the years, several Reinforcement Learning algorithms have been introduced and can be classified into three groups:

1. Actor-only (Policy Based Methods): These methods optimize directly the policy without using a value function. This is extremely useful in continuous or stochastic action space environments. The main problem is that the optimization methods used are typically the policy gradient, which suffer from high variance in the estimate of the gradient, resulting in slow learning.
2. Critic-only (Value Function Based Methods): These methods are trying to learn the value function that maps a specific value to each action from each state. This way, the best action from each state can be found, given a finite set of actions. This can be problematic and computationally expensive if the action space is continuous. Due to this, critic-only methods discretize the

continuous action space making it impossible to find the true optimum.

3. Actor-critic methods: These methods combine the advantages of actor-only and critic-only methods. The critic measures how good the action the agent took was (value based), while the actor controls how the agent behaves (policy based). The actor brings the advantage of calculating continuous actions without the need of optimizing a value function, while the critic supplies the actor with a low variance knowledge of the performance [47], [48].

Reinforcement learning methods can be modelled using a Markov Decision Process (MDP) [49], represented as a tuple $\langle X, U, f, \rho \rangle$. Here X is the state space, U is the action space, $f : X \times U \to X$ is the state transition function and $\rho : X \times U \to \mathbb{R}$ is the reward function which gives the instantaneous reward r_t. The state transition function in Model Free methods is unknown while the reward function is a design parameter.

Assuming a discrete time environment, at each time instant t, the agent applies an action u_t that depends on the current state of the system x_t. The action taken results in a new system state, x_{t+1}, and the agent receives a reward, r_{t+1}, of how good the action taken from the current state was. This is repeated for all time steps N in a trial, which can possibly be infinite. The purpose is to find the optimal policy π that optimizes the cumulative discounted sum of rewards:

$$R^{\pi} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... \tag{3.5}$$

where $0 < \gamma < 1$ is the discount factor [50].
Given that the agent follows a specific policy π the value function $V^{\pi}(x_t)$, is a measure of how good it is to be in a certain state x_t, and then continue following policy π from that state. It is equal to the expected discounted sum of rewards for an agent starting at state x. The value function depends on the policy by which the agent picks actions to perform, resulting in:

$$V^{\pi}(x_t) = \mathbb{E}[\sum_{i=1}^{N} \gamma^{i-1} r_i] = \mathbb{E}[R^{\pi} \mid x = x_t] \tag{3.6}$$

The value function can be decomposed, using the Bellman expectation equation, into immediate reward plus the discounted value of successor state:

$$V^{\pi}(x_t) = \mathbb{E}[\rho(x_t, u_t) + \gamma V^{\pi}(x_{t+1}) \mid x = x_t] \tag{3.7}$$

Among all possible value functions, there exists an optimal value function that has higher value than other functions for all states. By using equation (3.7) the optimal value function is derived:

$$V^{*}(x_t) = \max_{\pi}(\mathbb{E}[\rho(x_t, u_t) + \gamma V^{\pi}(x_{t+1}) \mid x = x_t]) \tag{3.8}$$

The optimal policy π^* is the policy that corresponds to the optimal value function.

$$\pi^{*} = arg \max_{\pi} V^{\pi}(x_t) \tag{3.9}$$

In addition to the value function, RL algorithms introduce another function called the state-action pair Q Function. The optimal Q function means that the expected

total reward received by an agent starting at state x and picking action u, will be optimal. Therefore $Q(x_t, u_t)$ is an indication of how good it is for an agent to pick action u while being is state x and then follow policy π [50].

$$Q^{\pi}(x_t, u_t) = \mathbb{E}[R^{\pi} \mid x = x_t, u = u_t] \tag{3.10}$$

Similar to the value function, the Q function can also be decomposed, using the Bellman expectation equation into immediate reward plus the discounted value of the successor state:

$$Q^{\pi}(x_t, u_t) = \mathbb{E}[\rho(x_t, u_t) + \gamma Q^{\pi}(x_{t+1}, u_{t+1}) \mid x = x_t, u = u_t] \tag{3.11}$$

Since the optimal value function is the maximum total expected reward when starting from state x, the optimal Q function, will be the maximum Q function over all policies π.

$$Q^{*}(x_t, u_t) = \max_{\pi}(\mathbb{E}[\rho(x_t, u_t) + \gamma Q^{\pi}(x_{t+1}, u_{t+1}) \mid x = x_t, u = u_t]) \tag{3.12}$$

Therefore the relationship of the value function and the Q function is derived as:

$$V^{*}(x_t) = \max_{u} Q^{*}(x_t, u_t) \tag{3.13}$$

This results in:

$$Q^{*}(x_t, u_t) = \rho(x_t, u_t) + \gamma \mathbb{E}_{t+1}[V^{*}(x_{t+1}) \mid x = x_t]) \tag{3.14}$$

Finally, combining equations (3.13) and (3.14), the optimal value function is computed as [50]:

$$V^{*}(x_t) = \max_{u_t}[\rho(x_t, u_t) + \gamma \mathbb{E}_{t+1}[V^{*}(x_{t+1}) \mid x = x_t])] \tag{3.15}$$

When attempting to solve a Reinforcement Learning problem, there are two main approaches to choose from: calculating the Value Function V or the Q Function Q of each state and choosing actions according to those, or directly compute a policy which defines the probabilities each action should be taken depending on the current state, and act according to it. Actor-Critic algorithms combine these two methods in order to create a more robust method [51].

3.3.2 Actor Critic

The Actor-Critic method, uses artificial neural networks to learn a policy π and a value function V separately and simultaneously. The policy is called the *Actor* and the value function is called the *Critic*. The role of the critic is to evaluate the current policy determined by the actor [47].

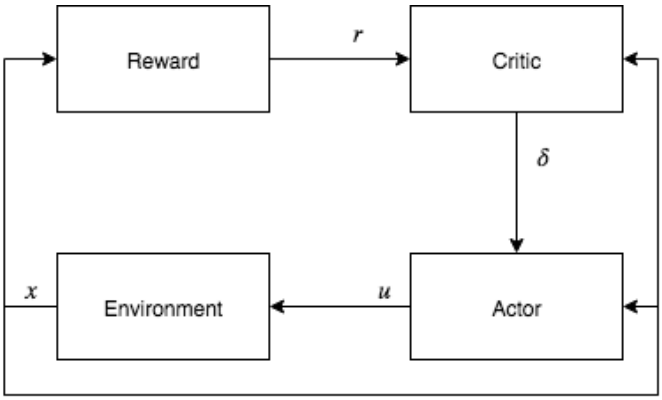


Figure 3.4: The Actor Critic architecture

The structure of an actor-critic algorithm can be seen in Figure 3.4. The learning agent has been split into two entities, the actor (policy) and the critic (value function). The actor's responsibility is to generate a control input u, given the current state x. The critic's responsibility is to evaluate the quality of the current policy by updating the value function's estimate. The actor is then updated using information from the critic. However, in problems where the state and/or the action space are continues it is necessary to approximate the policy and the value function in order to avoid enormous tables of possible states and actions.

Radial Basis Functions (RBF)

For the purpose of this thesis project, a linear function approximation method was used. Let us assume some feature extraction function $\phi : S \rightarrow \Phi$, that maps the states from S into features in the feature space Φ. Often, the dimentionality of the feature space is immensely smaller than the full state space, and this is the reason for using function approximations in continuous space problems where actions and/or states can be infinite. A linear function approximation is a simple parametric function that depends on the feature vector ϕ [58]. A generic function approximator can be defined as:

$$F(x, \theta) = \theta^{\top} \phi(x) \tag{3.16}$$

where θ is the parameter vector. As a feature extraction function ϕ a radial basis function (RBF) network was chosen.

The RBF network architecture that was used for this problem can be seen in figure 3.5.

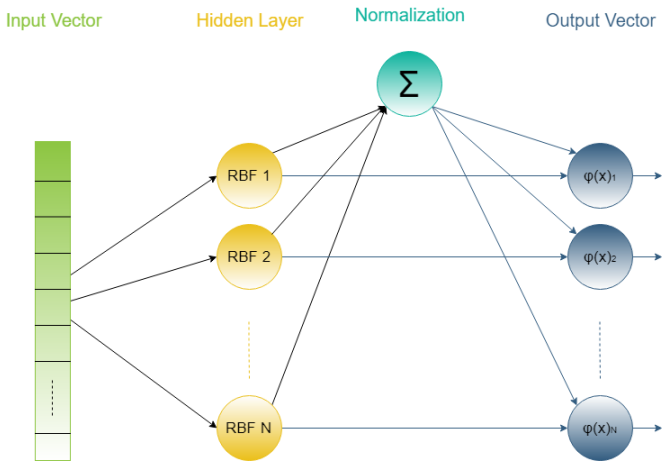


Figure 3.5: The RBF Network Architecture

A Radial Basis Function (RBF) network can be seen as a shallow feed forward neural network, with one hidden layer and one output layer. Each node in the hidden layer uses an RBF as a nonilear activation function, denoted $\tilde{\phi}$. The hidden layer performs a nonlinear transformation of the input, whereas the output layer is a linear combination mapping the nonlinearity into a new space. Each RBF neuron compares the input to a "center", and outputs a value between 0 and 1, which is a measure of similarity. The closer the input is to the "center", the closer the RBF neuron's output is to 1. The shape of the hidden layer's activation function, was chosen to be a Gaussian, and the "center" is the mean of the distribution [60].

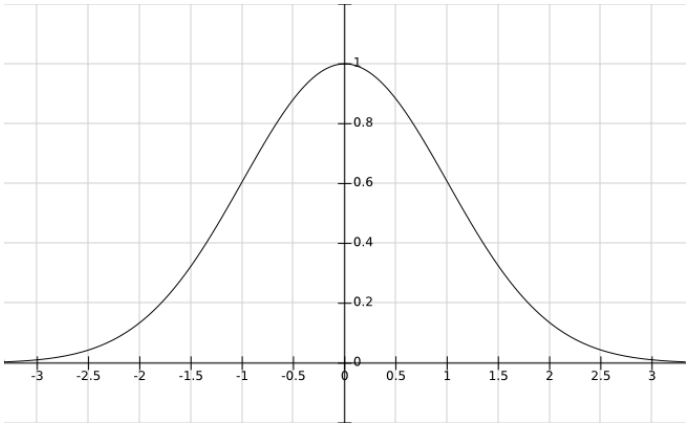


Figure 3.6: The Gaussian Distribution

Each neuron's activation function is defined as follows:

$$\tilde{\phi} = e^{-\frac{1}{2}(x-c)^\top B^{-1}(x-c)} \tag{3.17}$$

where x is the state vector, c is the "center" of the RBF and B is the covariance matrix of the RBF. The distribution of the centers, also called *kernels*, are linearly placed throughout the state space in a range of states which the application is expected to visit. It is possible to have a wide range, with a lot of parameters,

although this comes at a high computational cost. The number of kernels and their width in the state space is a design parameter. As with most regression problems, using too many kernels may cause overfitting and will have high computational cost, while having too few kernels will cause underfitting, resulting in poor performance. Moreover, since the application of the algorithm is online, it is necessary to lower the computational cost as much as possible, while keeping the performance as high as possible.

Despite the non linearity of the RBF network, Equation (3.16) is linear in the parameters, and therefore its gradient w.r.t parameters θ can be obtained as follows:

$$\frac{\partial F(x,\theta)}{\partial \theta} = \phi(x) \tag{3.18}$$

For the purposes of this thesis project, two RBF networks were used to transform the state and action continue spaces into a feature space that can then be fed into the linear function approximation equation (3.16) to get linear separability. The policy and value functions are then approximated by the actor and critic:

$$\hat{\pi}(x,\psi) = \psi^\top \phi_a(x) \tag{3.19}$$

$$\hat{V}(x,\vartheta) = \vartheta^\top \phi_c(x) \tag{3.20}$$

where ψ and ϑ are the parameter vectors of the actor and critic respectively, and ϕ_a and ϕ_c the corresponding RBF functions.

Temporal Difference Method (TD)

The Actor-Critic model is a combination of Monte Carlo ideas and dynamic programming (DP) ideas. Like Monte Carlo methods, Actor-Critic can learn directly from experience without a model of the environment's dynamics. However, instead of waiting until the end of an episode to update the agent's parameters, like in the Monte Carlo learning method, the Actor Critic updates the parameter vectors at each time step like in DP methods. This is also known as Temporal Difference learning (TD). Because the update is done at each time step, the TD learning method cannot use the accumulated discounted sum of rewards R^π in the policy gradient, since the future is not known. Instead, it needs to train a Critic model that approximates the Q Function [48].
An even better approach, that reduces the high variability of the Q function estimation is the usage of the Advantage Function. The Advantage Function is defined as:

$$A(x_t, u_t) = Q(x_t, u_t) - V(x_t) \tag{3.21}$$

where, Q expresses the value of action u_t in state x_t, whereas V is the average value of that state. The Advantage Function, represents the improvement of the action taken at that state compared to the average. In other words, this determines the extra reward collected if a specific action is taken. The extra reward is that beyond the expected value of that state [48].

- If $A(x_t, u_t) > 0$: the gradient is pushed in that direction, because it means the action taken, gave better reward than the average.

- On the other hand, if $A(x_t, u_t) < 0$: then the gradient moves in the opposite direction.

The complication of using the advantage function is that it requires both the Q as well as the value function [48]. However, by using equation (3.14), the advantage function can be rewritten as :

$$\delta(x_t, u_t) = \underbrace{\rho(x_t, u_t) + \gamma V(x_{t+1})}_{Q(x_t, u_t)} - V(x_t) \tag{3.22}$$

where the value $\rho(x_t, u_t) + \gamma V(x_{t+1})$ is also known as the TD target. Therefore, only the value function needs to be estimated. This function is known as the *TD - error* and is the cornerstone for the critic's update.

Different methods exist for evaluating the policy. The TD(0) method is very useful for online learning [52], although as it only has one-step backup, information in the past is not taken into account when updating the critic. The Monte Carlo methods on the other hand, does an update step after completing a full episode. These methods will therefore only know what reward a series of actions gave, and not knowing which actions had a positive or negative impact. Using the TD(λ) method which adds an eligibility trace ζ, it is possible to assign credit to states visited previously [47] which solves the issue of the return R^π depending on future values [53]. In a comparative study by Sutton et al. [53] it is shown that this inclusion often yields better performance in applications with continuous action-spaces. The eligibility trace that is used is called an *accumulative eligibility trace* and is updated through

$$\zeta_{t+1} = \lambda\gamma\zeta_t + \left.\frac{\partial \hat{V}(x, \vartheta)}{\partial \vartheta}\right|_{x=x_t, \vartheta=\vartheta_t} \tag{3.23}$$

where γ is the *discount factor* and λ is the *trace decay rate*. The trace decay rate is introduced to make information a long time in the past affect the update less. Using $\lambda = 0$ would imply TD(0) method, where updates of the policy are made according to the gradient of the estimated value function. On the other hand $\lambda = 1$ implies TD(1) method, which is similar to using the Monte Carlo method, where updates are made based on the gradient of the return [54]. When updating the eligibility trace, states that are visited frequently or have been visited recently will have larger values, while states that have never been visited or visited many time steps earlier will have lower values. Which λ is more appropriate depends on the application and how the reward function is formulated (in other words it is another tuning parameter).

The TD-error (3.22) used when updating the critic will then be scaled by the eligibility trace as:

$$\vartheta_{t+1} = \vartheta_t + \alpha_c \delta_t \zeta_{t+1} \tag{3.24}$$

where α_c is the learning rate of the critic and ϑ_t is the critics parameter vector in the current time step. Note that this equation updates the entire critic parameter vector at every time step, based on how good the action taken from the agent was. The critic uses the information of the TD-error to approximate and update the value function in every time step. The value function is then used to update the actor's

policy in the direction that improves the performance:

$$\psi_{t+1} = \psi_t + \alpha_a \delta_t \Delta u_t \left. \frac{\partial \hat{\pi}(x, \psi)}{\partial \psi} \right|_{x=x_t, \psi=\psi_t} \tag{3.25}$$

where ψ_t is the actors parameter vector in the current time step and α_a is the learning rate of the actor. An action that results in a positive TD error is better than expected, and hence the actor is updated to have a higher probability of taking that action the next time the agent arrives at this state. A negative TD error will result in an actor update that does not favour taking this action from the specific state in the future [56]. The actor is updated in the policy gradient direction, scaled by α_a. By using a small step size, α_a, the changes in the value function will only result in a small change in the policy, which leads to just a slight or no oscillatory behavior in the policy [55].

Exploration VS Exploitation

Reinforcement learning tasks, as discussed before, have no pre-generated training sets to learn from. Therefore, they create their own experience on-line, by acting, and evaluating that action. In order to do so, the agent needs to try many different actions in many different states in order to try and learn all available possibilities and find the path which will maximize its overall reward. This is known in the literature as exploration, as the agent explores the environment. However, if all the agent does is explore, it will never maximize the overall reward. Therefore, the agent must use the information it learned and exploit that knowledge to maximize the rewards it receives. The trade-off between exploration and exploitation is one of the greatest challenges of reinforcement learning problems. The two needs to be balanced, such that the agent both explores the environment enough, but also exploits what it learned and repeats the most rewarding path it found [51].
For this reason, every reinforcement learning algorithm needs an *exploration term*, Δu_t, in order to ensure that the algorithm does not get trapped in a local optimum. In continuous action spaces, the exploration term is often a random action combined with the policy action, which is forcing the algorithm to explore the entire state space before calculating the optimal policy. In [57], it is stated that exploration could be achieved by adding a zero mean white-noise with variance, σ_e^2, equal to one. A scaling factor β is then added to the exploration such that the sample drawn will fit in the range of suitable actions. The scaling factor β, is a tuning parameter that is highly dependant on the problem formulation and the policy values.

$$\Delta u_t \sim \beta \mathcal{N}(0, \sigma_e^2) \tag{3.26}$$

The complete actor-critic algorithm used for the purposes of this thesis project is presented in Algorithm 1:

Algorithm 1 Actor-Critic Algorithm

Initialize λ, γ, α_a, α_c
Initialize ϑ_0, ψ_0
for each trial **do**
Initialize x_0
Generate an initial action u_0
Initialize eligibility trace $\zeta_0 = 0$
$t \leftarrow 0$
repeat
Generate exploration Δu_t
Calculate current action $u_t = \hat{\pi}(x_t, \psi_t) + \Delta u_t$
Apply u_t, measure x_{t+1}
Receive reward $r_{t+1} = \rho(x_t, u_t)$
$\delta_t = r_{t+1} + \gamma \hat{V}(x_{t+1}, \vartheta_t) - \hat{V}(x_t, \vartheta_t)$
$\zeta_{t+1} = \lambda\gamma\zeta_t + \left.\frac{\partial \hat{V}(x,\vartheta)}{\partial \vartheta}\right|_{x=x_t, \vartheta=\vartheta_t}$
$\vartheta_{t+1} = \vartheta_t + \alpha_c \delta_t \zeta_{t+1}$
$\psi_{t+1} = \psi_t + \alpha_a \delta_t \Delta u_t \left.\frac{\partial \hat{\pi}(x,\psi)}{\partial \psi}\right|_{x=x_t, \psi=\psi_t}$
$t \leftarrow t + 1$
until $t =$ maximum number of samples T_s
end for

4

Experiments And Results

In this chapter an overview of the complete problem formulation and experiments will be presented. The two main components of this project will be presented in detail:

1. How an artificial neural network was used to estimate the dynamics of a deformable object
2. The control methods used in order to track a force trajectory on the deformable object

4.1 Problem Formulation

The final objective of this thesis project is for a robotic manipulator to perform force tracking tasks on a soft-deformable object. Since the environment (object) is unknown, the main idea is to estimate the object's deformation model in order to use it in the control scheme.
For this phase, instead of using the existing mathematical models to estimate the dynamics, a data driven method was adopted and assessed in its capability of estimating any function. As a soft deformable object, a sponge was used, due to its elastic capabilities. The intention for the purpose of the experiments was to use a soft deformable object that can return to its original form if no force is applied, speeding up the procedure significantly, since we do not have to change the object after every single experiment. However, the sponge like many elastic materials follow one curve (path) on stretching when the stress is increased and follows another path while regaining its original configuration, in the form of returning to its initial position at a slower rate, the longer the robot is "poking" it.
When a deforming force is applied on a body, then the strain does not always change simultaneously with stress, rather it lags behind the stress. The lagging of strain behind the stress is defined as elastic hysteresis. This is the reason why the values of strain for some stress are different while increasing the load and while decreasing the load.
The effects of these characteristics will be clear in the results.
In the next phase of the project, four different controllers were designed and evaluated, in tracking 2 different force trajectories on the deformable object. During this phase, the advantage of using the model is also assessed, in both trajectories.
It is essential here to note, that both trajectories were designed on the $z - Axis$, which means that the control problem is 1-Dimensional. The reason behind this is that deformable objects, as stated before, consist of infinite degrees of freedom (Guo

et al. 2013). Therefore, moving in x or y direction, would implicate things and that would result in less reliable results from the controllers. Nonetheless, the controllers designed would also work on a $3D$ movement of the manipulator into the object, but it would require more tuning and possibly even then due to the highly non linear dynamics the response would not be as good. While presenting the method and results no subscript z will be used, since it is implied that every variable refers to the $z - Axis$.
Finally, a new controller was designed to achieve adaptive optimal control. The Actor Critic controller does not require the learned model, but learns it on-line by itself and can adapt to small environment changes. But first, lets get familiar with the robotic manipulator used for the experimental process.

4.2 The UR10 Robotic Arm

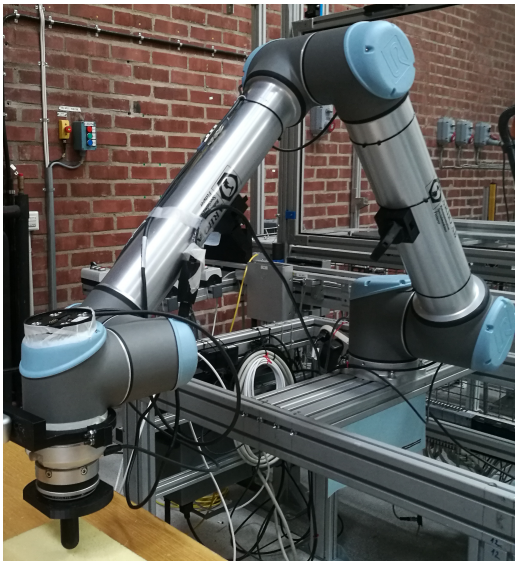


Figure 4.1: The UR10 Robot. Joints from the bottom to the top: base, shoulder, elbow, wrist 1, wrist 2 and wrist 3.

The UR10 is a 6 degree of freedom (DoF) robotic manipulator, designed by universal robots, for tasks where precision and reliability are important. The UR10 is a velocity/position controlled robotic manipulator, which means that irregardless of the type of control we want to implement, the robot's inputs must always be velocity/position, either in the joint space, or in the operational (task) space. The manipulator is shown in figure 4.1. For the purpose of exchanging data with the robotic manipulator, the Robot Operating System (ROS) framework was used. ROS is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms.
On the end effector of the robotic manipulator an Optoforce $6 - Axis$ F/T sensor is equipped in order to measure the forces exerted on the end effector from the

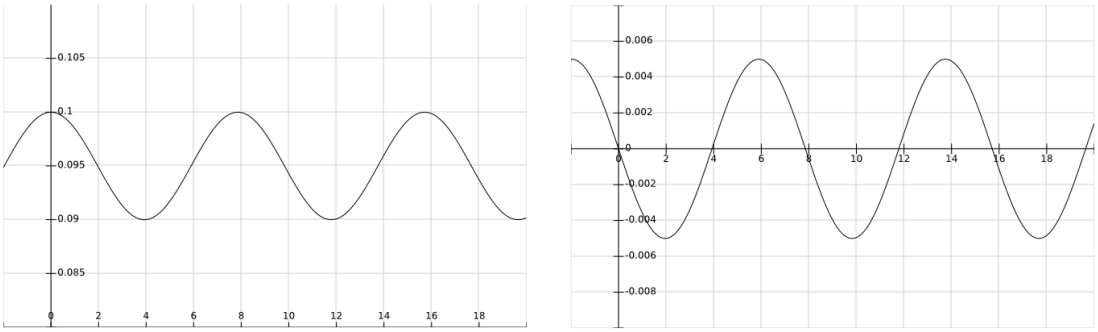
environment. However, due to the formulation of the problem, only the force on the z Axis will be used. Moreover, a $3D$ printed poking device was used for two main reasons. a) would be best to avoid direct contact of the sensor with the environment, for safety reasons, and b) the surface of the sensor itself is $70mm$ wide, which makes the forces applied to the sponge higher than intended, and the arm's motors may reach their limit and cease working for safety reasons. This tool that was $3D$ printed with the help of the Chalmers staff, can also be seen in figure 4.1 mounted onto to F/T sensor.

For the purpose of this thesis project, all the controller outputs (robot's inputs), will be in the operational (task) space, in the Cartesian space. Because of this the dynamical model of the robotic manipulator will be treated as a black box, with one input: the velocities of the end effector in the Cartesian space, and the output, or the measurements, will be the force exerted to the end effector from the environment as it is measured from the force/torque sensor, and the position of the end effector in the Cartesian space.

Position and Force Trajectories

In both the object identification, as well as the force control section the same two trajectories will be used. In the object identification phase, it is a position trajectory, whereas in the force tracking task it is a force trajectory. For this reason only some magnitude values will be different since the units are different (mm and N).

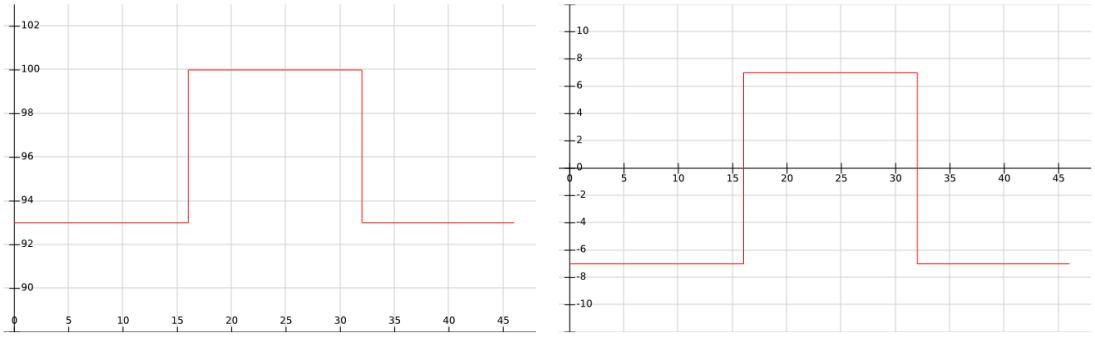
- Sinusoidal Trajectory : The first trajectory that was designed for this experiment for the data collection phase was a sinusoidal trajectory with a magnitude of 5 mm, since the width of the sponge was 70 mm. The magnitude was selected adequately small in order not to lose contact with the material. Note that in order for the data to be accurate the manipulator should not, at any point lose contact from the material, because in that case the data will not represent the object's dynamics. For the force tracking task, the magnitude was selected, after consideration of the forces acted on the sponge to be $5N$. The frequency of the sinusoidal for both tasks had to be defined while accounting for the ROS node's update rate. The ROS node frequency was set to 125 Hz, which is the default update frequency, which means the trajectory is updated every $0.008s$. Taking this into consideration and after experimentation with different frequencies, the sinusoidal's frequency rate was decided as 0.099 Hz, which means 10.048 s and this translates into 1256 samples. However, in the data collection phase, a lot of different frequencies were used to collect more extensive data of the environment's behaviour. The reference position p_r as well as the reference velocity $\dot{p}_r$ for the sinusoidal trajectory are presented in Figure 4.2.



a) Position Trajectory. b) Velocity Trajectory.

Figure 4.2: Sinusoidal Trajectories

- Step Trajectory : The second trajectory was a step function with a magnitude of 7 mm in the data collection phase and 5 N in the force tracking phase, and a frequency of 0.0625 Hz, which means 2000 samples, or 16 s. The reference position p_r as well as the reference velocity $\dot{p}_r$ for the step trajectory are presented below.



a) Position Trajectory. b) Velocity Trajectory.

Figure 4.3: Step Trajectories

It can be observed, that the position was designed in such a way, that the reference trajectory always starts from the robot's current position at the beginning of each trial. This was done in order to avoid big "movement spikes" at the beginning of each trial. In the left figure in 4.2 and 4.3, it was assumed that the manipulator's initial position was 0.1 m or 100 mm respectively, and then starts oscillating from there, always staying in contact with the sponge.

4.3 Object Identification

The first phase of the experimental procedure was to identify the model of the sponge (environment). For this task an artificial neural network was designed and

trained using supervised learning. Since the robotic manipulator used in this thesis project is a velocity controlled robot, the model identification task is to identify a dynamical model that, for some inputs, calculates the robot's velocity, or a way to calculate that velocity. This velocity can be used afterwords in the control schemes as a reference velocity to achieve the desired force. As in every supervised learning problem, in order to train the network, a ground truth value is needed. In other words data had to be collected from the interaction of the real robotic arm with the sponge, and use those data to train the neural network.

Inputs-Outputs

However, the input-output set of the network had to be defined carefully in order for the network to be able to approximate the object's model. Many different input-output sets were tested, one of them being (Inputs : Current Force, Current Position, Outputs : Current Velocity). The main problem with this input-output set is the fact that a 1 on 1 mapping between the inputs and the output does not exist. This means, that for a specific set of inputs, more than one output can be defined (e.g in a current position with a current force, the manipulator can be moving upwords, or downwards, positive or negative velocity).
It is important here to note, that in order to calculate the ground truth velocity (network's output), the derivative of the measured position has to be used, since the way the forward kinematics calculate the velocity contains a significant amount of noise and has many more errors due to the online calculation and use of the Jacobian. That being said, another set of input-outputs that was tested was: (Inputs: Current Position, Current Force, Outputs: Next Position). The network was able to adequately identify a model for this set of input outputs. However, even the smallest error in the estimated position will result in a huge error in the calculated velocity when we use the derivative.
For all these reasons the inputs and output of the network was defined as such :

- Inputs: Current Force, Previous Position, Current Position
- Output: Current Velocity

For this set of inputs-outputs the current velocity had to be computed by differentiating the measured position. This is feasible off-line because the data contain the next position, and such the use of the second equation of motion can be used to calculate the current velocity :

$$v = \frac{\Delta p}{\Delta t} = \frac{p_{t+1} - p_t}{\Delta t} \tag{4.1}$$

where $\Delta t = 0.008$ since as stated before the ROS node's frequency is 125 Hz. The identified model, can then be used in an online implementation, and calculate the current velocity, based on current and previous data only.

4.3.1 Data Collection

Controller

Data collection is an important part in every supervised learning problem. It is important that the data represent all the "states" of the model that the network is

trying to identify, or as many as possible. For this purpose a controller had to be designed, with the objective to move the robotic arm on the sponge, covering as much states of the model as possible. The controller's purpose is not to accurately track a position trajectory. The objective is to adequately track a trajectory, without damaging the equipment and without losing contact with the sponge, and thus a Compliant Control scheme was used.

$$V_z = \dot{p}_r - aF + K_p(p_r - p) \tag{4.2}$$

The purpose of this controller was solely to track a position trajectory on the $z-Axis$ while the robotic arm is in contact with the sponge, and the compliant term was making sure no huge forces would be applied to the sponge. With this scheme, data where collected as the robotic arm was tracking the two different trajectories on the sponge, in order to try and include as much information about the object as possible. Moreover, as stated before, in the sinusoidal trajectory different frequencies were also used with the intention of getting as much information about the environment's behaviour as possible.
It is important here to note, that the controller was not tuned because its response is not important for this phase. The objective here is only to get data when the manipulator is moving while in contact with the material, and the controller is just there to prevent high forces, and keep the manipulator in contact with the sponge at all times.

Sensors

As the robotic arm is moving, at each time step the end effector position is calculated, using the manipulator's translation matrices. The position of the end effector is calculated in relation to the base frame, and is saved in a *.csv* file. In order to generalize the collected data, at the start of each trial the initial position was measured, and then every other position was subtracted from the initial. This way, irregardless of the position the manipulator is starting from, the measured data was being treated as if the end effector started from zero.
In order to receive the force data from the F/T sensor, the Oproforce Etherdaq driver for ROS was used. This driver allows the *wrench* node of ROS to receive the data from the Optoforce sensor and then transmit it, in order to save the force measurements in a *.csv* file. It is important to note, that the force sensor's sensitivity is 0.1 N, and this will play a big part in the controllers performance, since this is the reason we can not achieve higher accuracy than that. Furthermore, the driver itself allows for a normalization of the force data, by saving the current (usually initial) force, and subtracting every other measurement from the initial. In this way every time a trial was initialized with the end effector already inside the sponge, the force was being set to zero.

Filtering

The calculations for the end effector position are being performed online, at every time step, while receiving the angles of each joint of the manipulator and using the forward kinematics of the UR10 robotic arm to compute the end effector task

space position. Therefore, sometimes huge sensor errors were observed in the form of "spikes", which can affect the validity of the received position data. The force sensor, on the other hand, doesn't perform any calculations so the measurements are more stable. However, in both position and force measurements, due to the sensitivity of the sensors, high frequency noise was also present. In order to avoid those faulty measurements (high frequency noise), a second order butterworth low-pass filter was used with cut-off frequency equal to 0.5 Hz, that was decided after careful observation of the frequency response of the data. As is was explained in chapter 2.2, filtering of the measurements is imperative, in order to ensure the success of the learning process of the neural network.

A plot of the difference between the filtered and unfiltered input data collected is presented bellow :

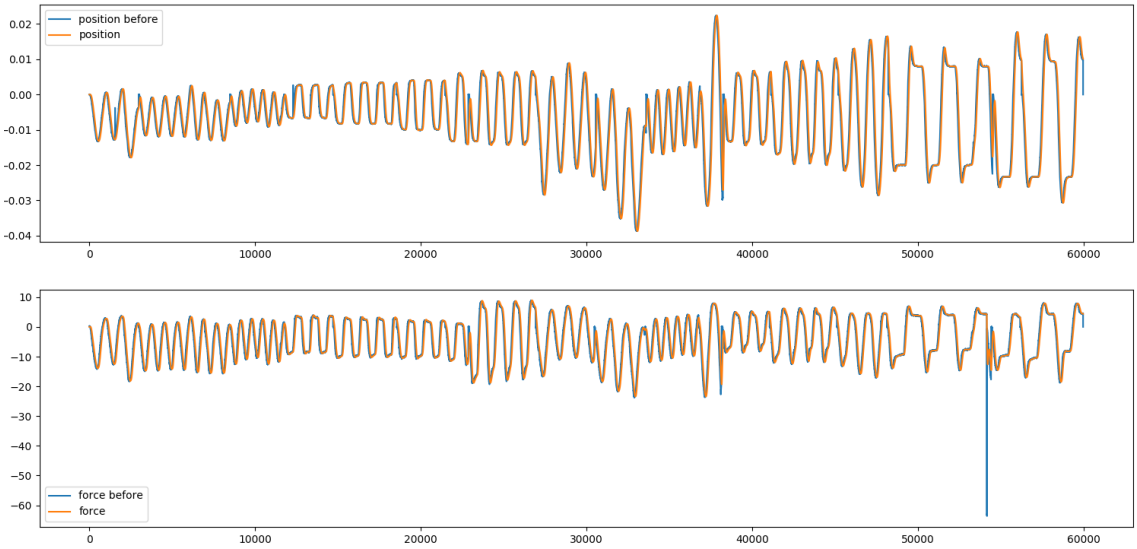


Figure 4.4: Position and Force data Before and After Filtering.

However, the most important impact of the position filtering is the fact that the ground truth output of the network, i.e the velocity is also calculated from the filtered position signal. This reduces the noise significantly as can be seen in figure 4.5, making a huge impact in the networks identification capabilities.

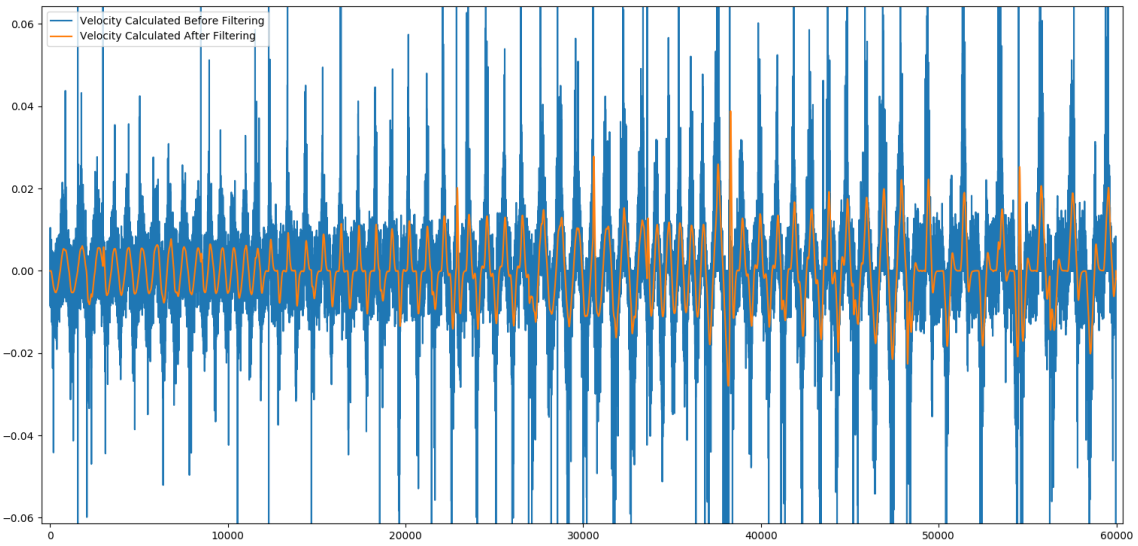


Figure 4.5: Unfiltered and Filtered Velocity.

4.3.2 Training Process

The dataset have been normalized and filtered, but it is not ready yet for the neural network. The data set has to be split into three sets, the training set, the validation set and the test set. Consequently, the standardization process will be used, to improve the network's performance and stability. These two steps are extremely important before starting the actual training of the neural network, and can greatly improve the approximated model.

Train-Validation-Test Split

- Training Dataset: The sample of the data that is used to fit the model. This is the actual dataset that is used to train the model (tune the weights and biases). The model learns from this data.
- Validation Dataset: The sample of the data used to provide an unbiased evaluation of a model fit on the Training dataset while tuning the model hyperparameters. The Validation set is used to evaluate a given model, but this is for frequent evaluation. Hence the model occasionally sees this data, but it does never "Learn" from it. So the validation set in a way affects a model, but indirectly.
- Test Dataset: The sample of the data used to provide an unbiased evaluation of a final model fit on the training dataset. The Test dataset provides the gold standard used to evaluate the model. It is only used once a model is completely trained (using the train and validation sets). The test set is generally what is used to evaluate competing models. It contains carefully sampled data that spans the various classes that the model would face, when used in the real implementation [62].

The first step after the normalization and filtering of the dataset is to split it into the "Training" and Test sets. For this step, 20 % of the entire dataset, at random,

is chosen to be the Test set. The rest is used in the training process, and is being split afterwords into the Training and Validation sets. For this split again 20% of the set is chosen to be the validation set. The test split and the validation split is being decided by the engineer, and depends on the model the network is trying to learn as well as on the size of the dataset.

Standardization

By observing figure 4.4, one can deduct that the force measurements are approximately two orders of magnitude larger than the position data. As discussed in section 2.2, standardization of the input data is extremely important for the network's performance. Therefore, the *StandardScaler* function of the *sklearn* python library was used in order to standardize the input features by removing the mean and scaling to unit variance.

One important detail is, that when training the network, the data set is being split into training data, and test data (unseen data). The standardization function is applied to the training set only, and this is because it is the only knowledge the network has. Subsequently, the scaler's parameters are being saved such that they can be used in any other set of inputs the network gets (Test set), as well as in the online implementation. This means that when the network is used online, at every time step, the features will be scaled before getting into the network. The effect of the standardization on the data set used for this project can be seen in figures 4.6 and 4.7. Note that, in the figures the whole data set is presented standardized, since the Train-Validation-Test split is performed at random, and thus the data would be mixed. In the implementation, the training set defines the standardization, and then based on this predefined scaler, the validation and test sets are being scaled. So the whole data set is presented for visual purposes.

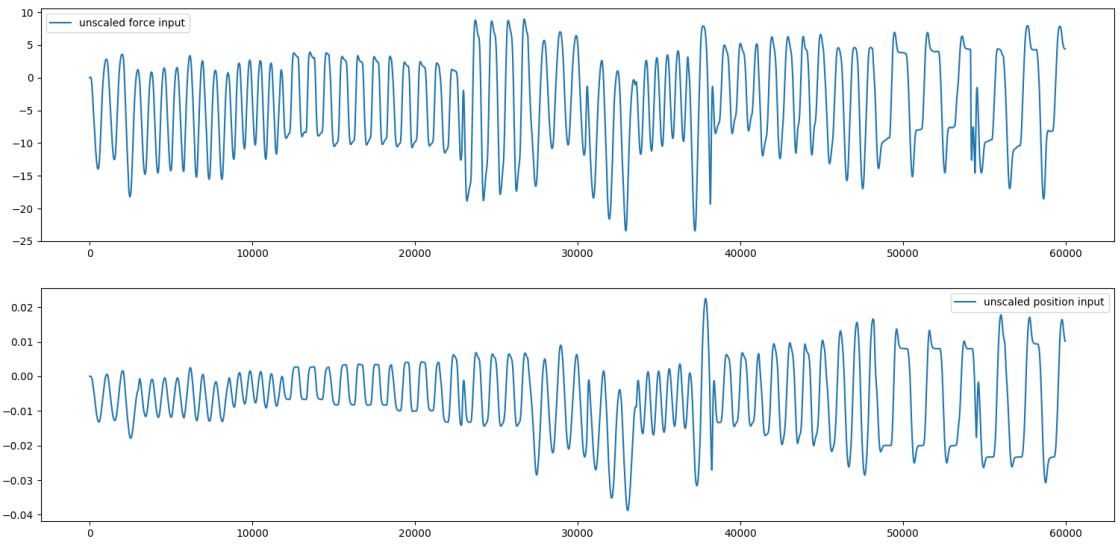


Figure 4.6: Unscaled input dataset features

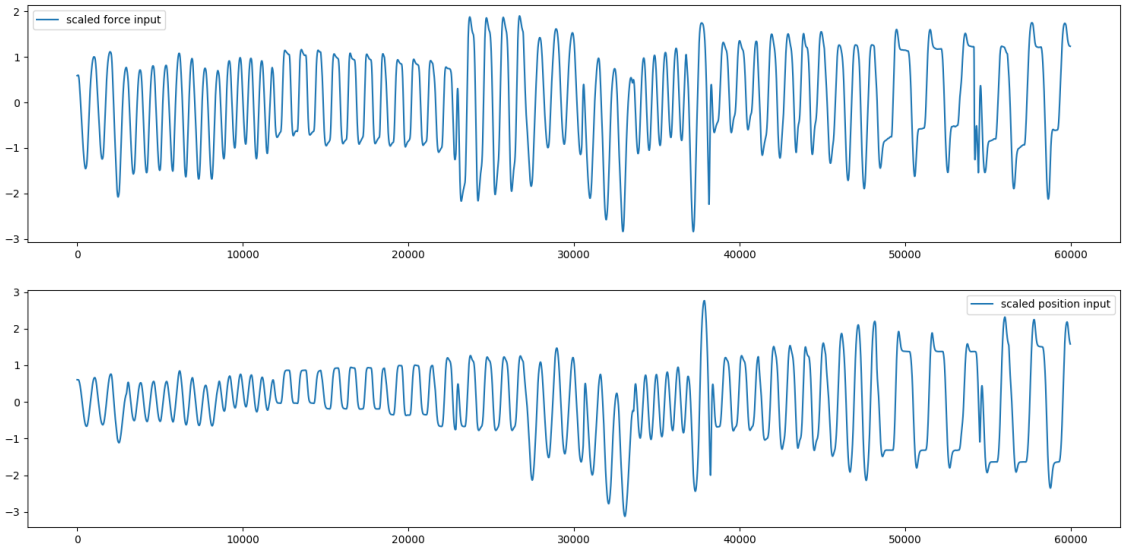


Figure 4.7: Scaled input dataset features

Neural Network Architecture

Now that all the pre-processing steps have been performed, the dataset is used to train the artificial neural network. In order to build the neural network in python, the keras library of python was used, with $Tensorflow$ as back-end. The architecture of the neural network for this specific application is presented below.

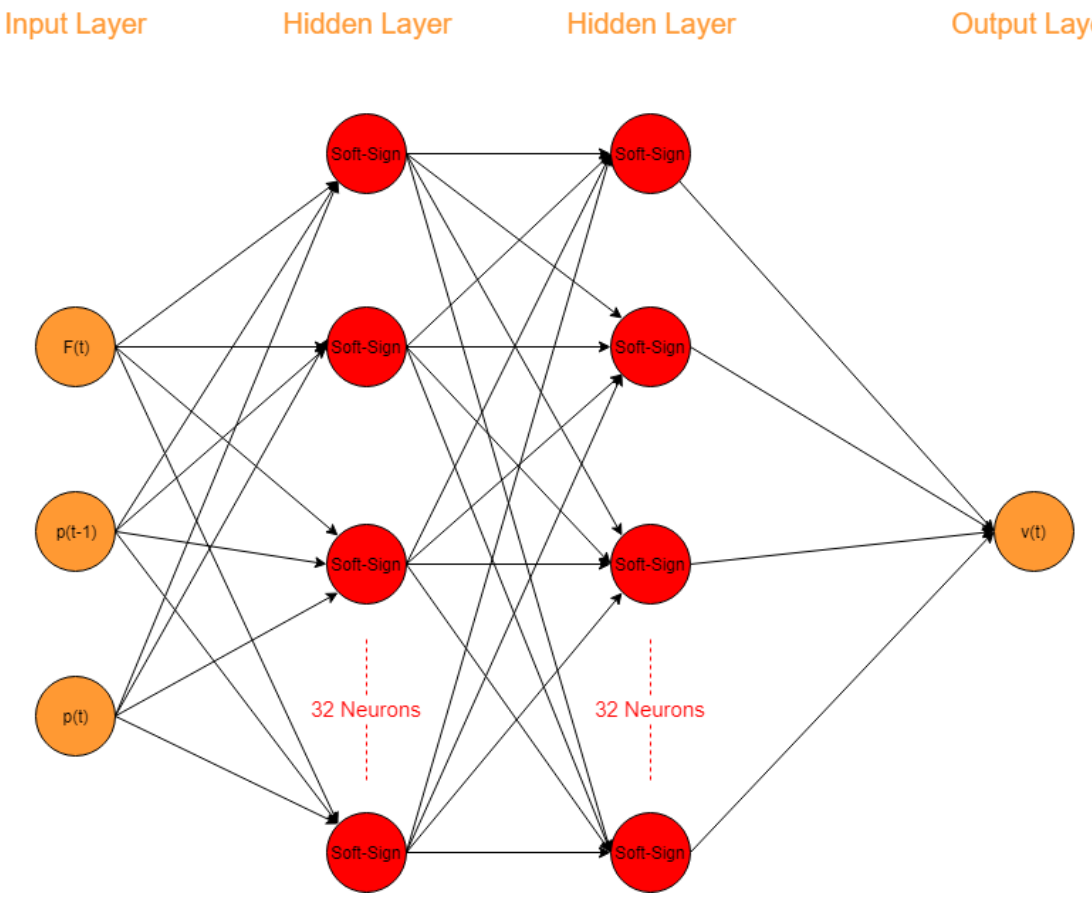


Figure 4.8: Neural Network Architecture

As discussed before, the network architecture is highly dependent on the nature of the problem, and the function the network is trying to approximate. This architecture was decided after experimentation with different architectures (number of layers, number of neurons per layer, activation functions, loss function, etc). The nature of the problem, requires the network to be used online, and thus the computational cost was an important factor in the architecture decision process. Finally, a simple MLP with 2 hidden layers and 32 neurons per layer with a $SoftSign$ activation function was enough to estimate the dynamical model of the sponge. Since the network is trying to approximate a function the output layer did not use any activation function. As a Loss Function, the $Log - Cosh$ function was used, since it was observed that it yielded the best results, and the $Adamax$ optimizer was updating the model in response to the output of the loss function. The parameters of the $Adamax$ optimizer were tuned in order to get the best possible results ($Learning\ Rate = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, L_r\ Decay = 0.0001$).
The artificial neural network was trained for 300 epochs with a batch size of 128. The batch size, defines how many samples will be propagated through the network before updating the weights and biases. With this choice, in every epoch the entire training set will be split into $(60000 \cdot 60\%)/128 = 281$ mini batches and update the weights and biases 281 times in every epoch. The training and validation loss functions during the course of the entire training process are presented below.

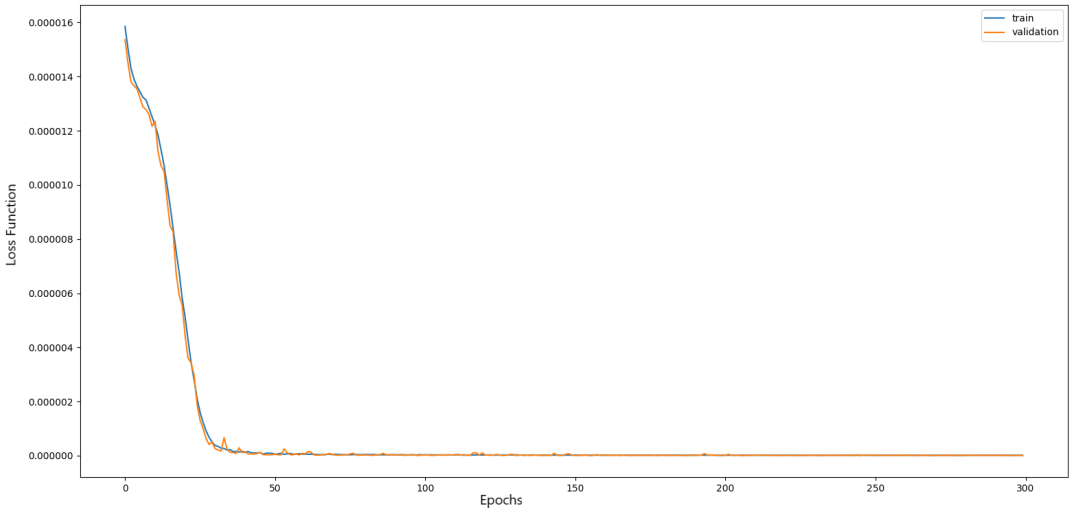


Figure 4.9: Training and Validation Loss Functions

From figure 4.9, it can be observed that after the first 100 epochs the improvement in both loss functions seems insignificant. This means that training the network for 100 epochs would be enough to minimize the loss function. However, since the training is being performed off-line and the computational cost or time is not an issue, it was decided to let the network train for 300 epochs, so that the learning rate decay can take effect and improve the estimation as much as possible.
This can be observed in figure 4.10, where the absolute error between the predictions and the ground truth is presented. As can be seen, the validation set is constantly improving and reducing it's oscillations, while the training set, is similarly improving constantly, even after 100 epochs.

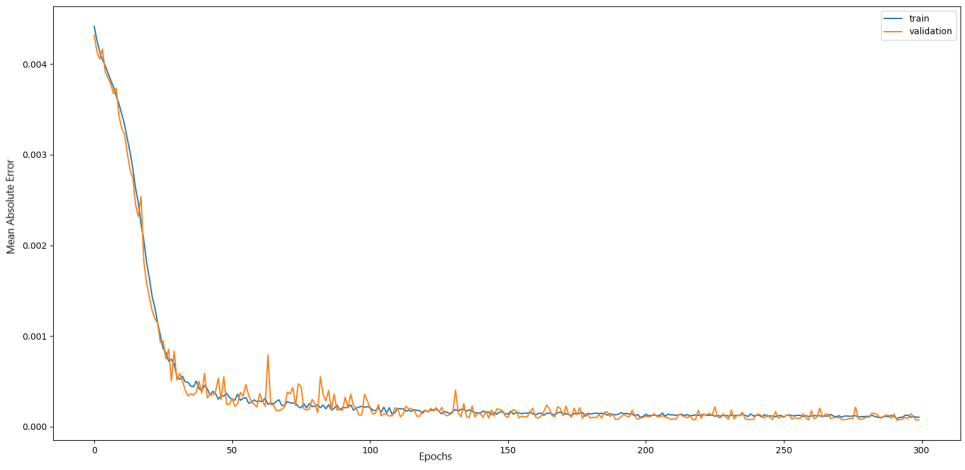


Figure 4.10: Mean Absolute Error between the Prediction and the Ground Truth

In figure 4.11, the prediction of the trained network is displayed, in the training set, the test set, and the entire dataset. Note, that because the test split, and

the validation split was performed at random, the first two plots, were sorted by magnitude for visual purposes. The last plot, is the entire sequence, as if it was performed online through time.

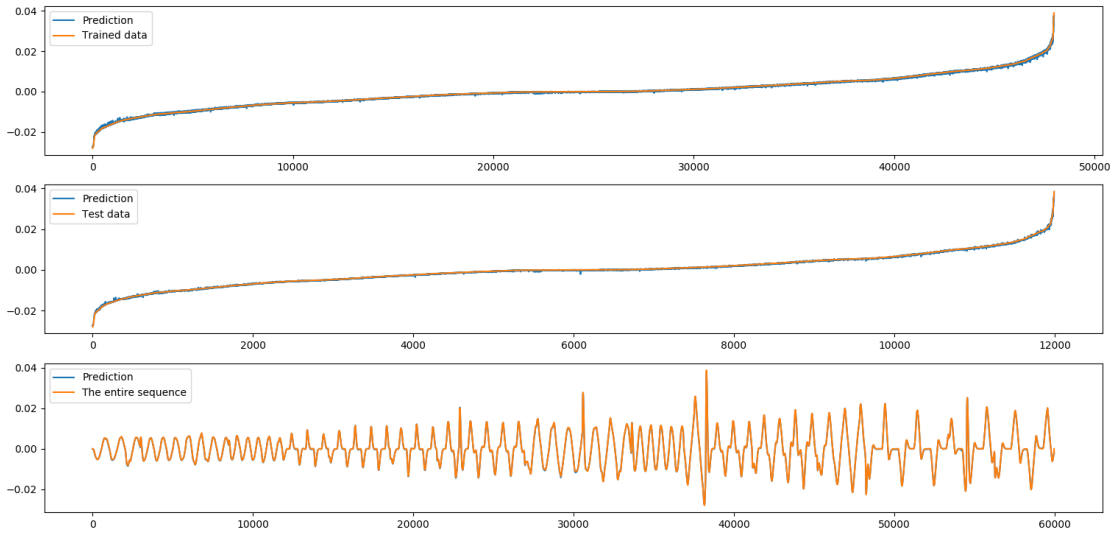


Figure 4.11: 1: Training Set, 2: Test Set (Unseen Data), 3: Entire Dataset

The trained network as can be observed from the plots has an accuracy of $1.04e-04$. The physical meaning of this, because the network is predicting velocities, is that the network is predicting the manipulator's velocity with a maximum error of $1.04e-04\ \frac{m}{s}$. To understand how significant this is, it is enough to observe that the velocity of the manipulator during it's movement across the sponge is : $[-0.05,\ 0.05]$.

Now that the network is trained, it is saved into a file so that is can be used later in the online implementation. The trained model will be a static function that receives the three indicated inputs, and provide one output. In other words this is the deformable object's dynamical model, it's a function that describes the manipulators velocity based on the object's deformation. This model will be then used to design controllers, to track force trajectories with the robotic manipulator on the deformable object.

4.4 Force Control

The second phase of the experimental process was to track force trajectories while the UR10 robotic manipulator, is moving while in contact with the deformable object. The objective is, to assess how the trained neural network can be used to improve the tracking performance of the controllers. In this section, 5 different controllers will be tested (P-only, PI, PI with feedforward, P-only with feedforward, Reinforcement Learning).

4.4.1 The Use of the Neural Network

In the previous section, it was discussed how a neural network is trained to estimate a manipulator's linear velocity, as a function of the manipulator's current and previous position, as well as the current force acted to the environment. Assuming the manipulator is required to track a specific force trajectory $F_r(t)$, this sequence can be the force input to the neural network. This way the network will output the reference velocity the manipulator requires in order to achieve that force. The objective is only to track a force trajectory, and thus the position of the manipulator is not relevant in this implementation.

4.4.2 The On-Line Implementation

This task is being performed online using the real real manipulator, and thus it requires the use of ROS, as in the data collection part. The difference is that at every trial, the trained neural network model will be loaded, as well as the scaler, to scale the inputs to the network.
Moreover, the inputs to the neural network require filtering and this has to performed online. The difference with the off-line filtering implementation is that in this case, while initializing the filter there is a transient period until the filter reaches the steady state. This transient period can greatly affect the neural network's estimations, and this would drastically deteriorate the controller's performance. For this reason, at the start of every trial a 300 samples period was given to the system, in order for the filter to stabilize. If the system is working endlessly, this only needs to happen once, when the system starts and thus it does not affect the rest of the implementation.
Finally, it is important to note, that for this task the reference force is used as an input to the neural network. This reference force is designed and sent into the system by the engineer and thus it does not require filtering, hence significantly reducing the computational cost of the implementation.

4.4.3 Controllers

The controllers used for this experiment will be presented and assessed on their ability to track a force trajectory on the deformable object, as well as their adaptability to different trajectories, without further tuning. Tuning a controller is an important procedure, and it is highly dependent on the application and the trajectory the controller is tracking. This means that for most of the controller schemes, in order to adequately follow a different trajectory without deteriorating performance, the controller has to be tuned anew.
In order to demonstrate this, 2 new trajectories were defined, one sinusoidal with different frequency and magnitude than the original one, and one step with a different magnitude. The new sinusoidal trajectory has a frequency of 0.159 Hz, or a period of 6.28 s, which translates into 785 samples, and a magnitude of 8 N. The step trajectory on the other hand, has a magnitude of 10 N. These two new trajectories will be shown in the Force tracking plots.
The first two controllers to be discussed will be the P-only and the PI controller without the use of the Neural Network as a feedforward term. Afterwards, the im-

provements of the feedforward term will be presented, as well as an optimal adaptive controller that was designed using reinforcement learning. It is important to note, that each controller was tuned separately from the others, and for each trajectory. However, the tuning was done only for the original 2 trajectories, and the same weights were used for the new trajectories.

The performance of the controllers will be assessed differently for each trajectory. For the sinusoidal, the assessment will be based on the mean square error of the measured force and the reference force signal. On the other hand, for the step trajectory the comparison will be split into the transition period and the steady state. It was decided that for all controllers, the steady state is when the error of the measured force and the reference force is less than 1 N.

P-only

The first controller that was tested in this experiment was a simple P-only controller, that only takes into account the force error, between the reference and the current force.

$$V_p = K_p(F_r(t) - F(t)) \tag{4.3}$$

The K_p parameter was tuned differently for each one of the two trajectories, to achieve the best possible performance for this controller scheme.

The P controller's performance is presented in Figure 4.12:

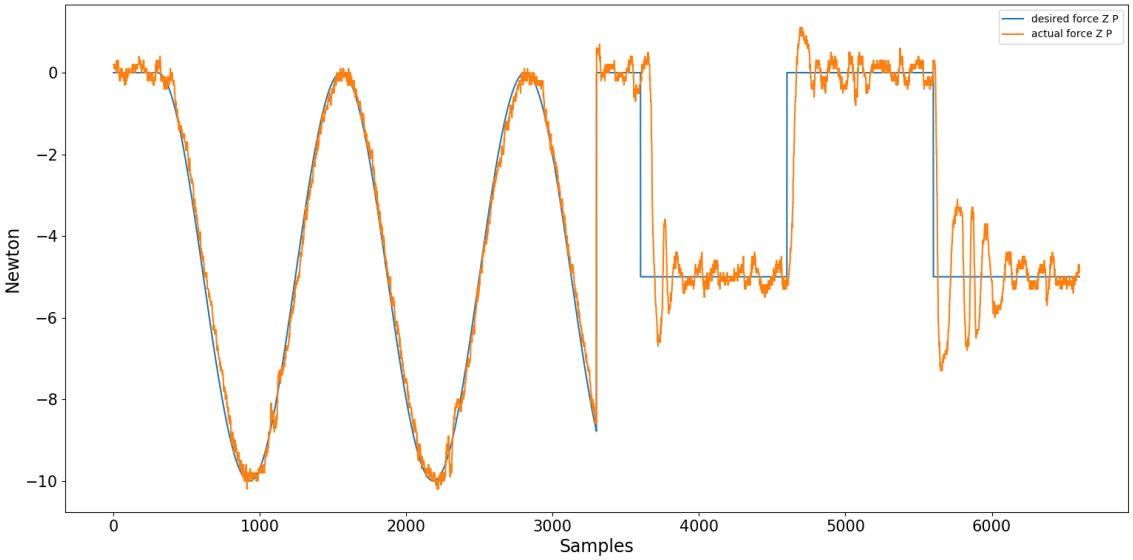


Figure 4.12: P controller: Force Tracking Performance

In figure 4.13, the force error is presented for both trajectories.

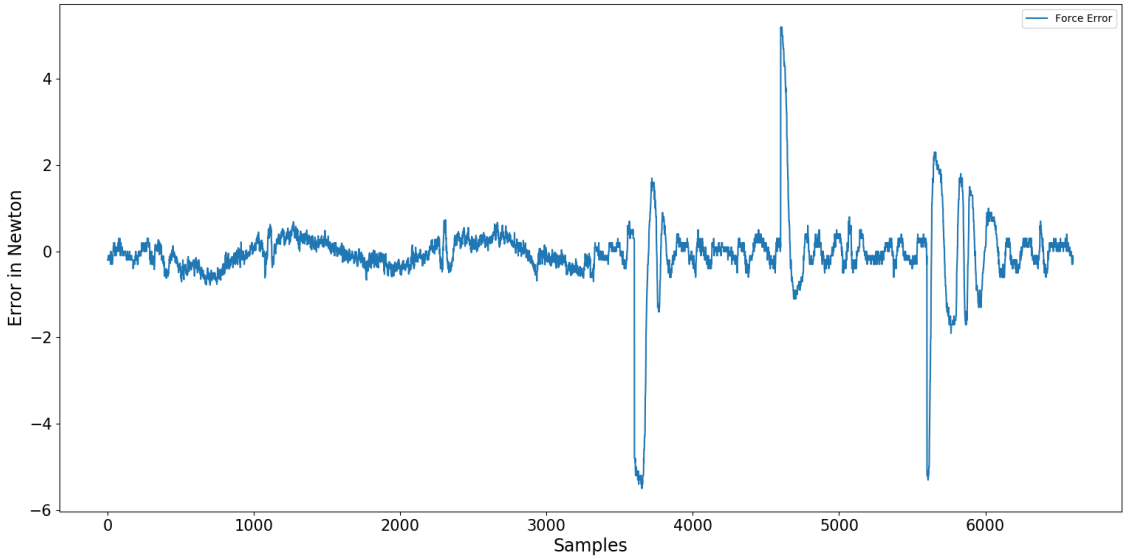


Figure 4.13: P controller: Force Tracking Error

Finally, this controller was tested in it's adaptability by tracking a different trajectory without re-tuning the K_p parameter. The controller's performance is presented in Figure 4.14:

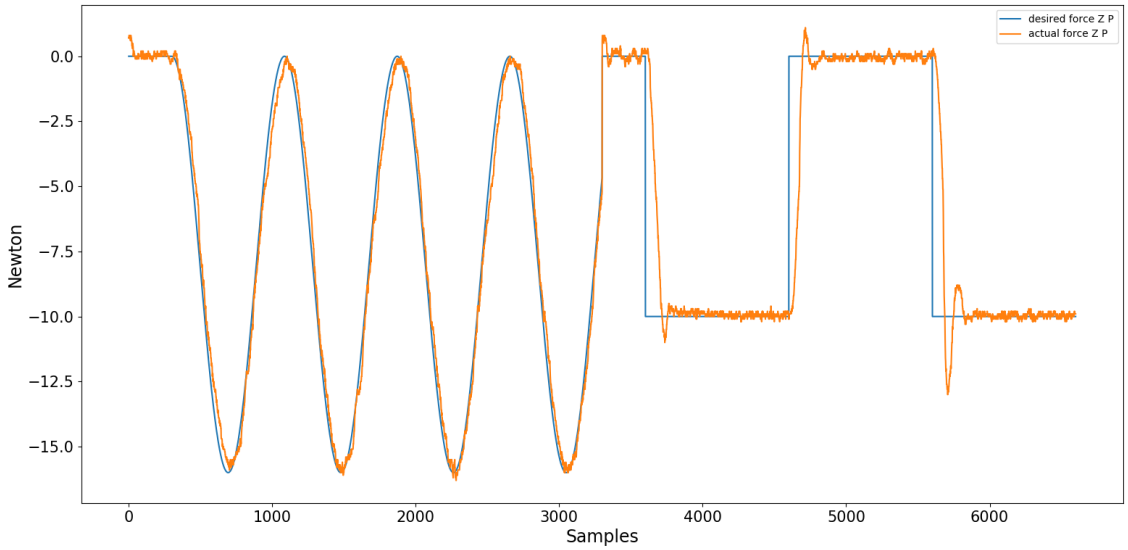


Figure 4.14: P controller: Force Tracking Performance for the new trajectories without tuning

In figure 4.15, the force error is presented for both new trajectories (Sinusoidal, Step).

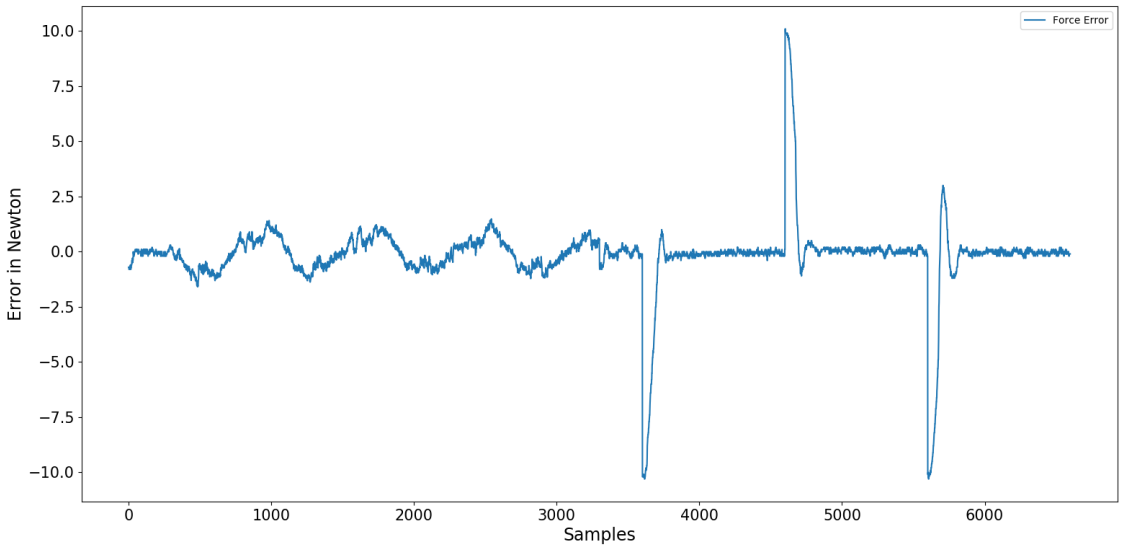


Figure 4.15: P controller: Force Tracking Error for the new trajectories

PI

In order to try and improve the first controller's performance, an integral term was added, resulting in a PI controller, that only takes into account the force error, between the reference and the current force.

$$V_p = K_p(F_r(t) - F(t)) + K_I \int (F_r(t) - F(t))dt \tag{4.4}$$

This controller contains two tuning parameters, K_p and K_I that were tuned differently for each one of the two trajectories, to achieve the best possible performance for this controller scheme.
The PI controller's performance is presented in Figure 4.16 :

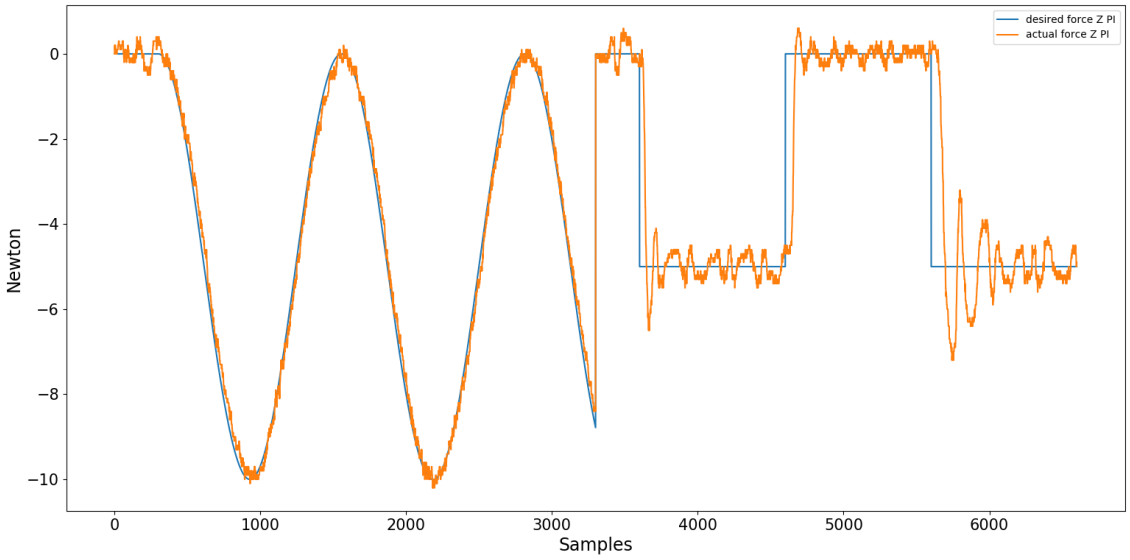


Figure 4.16: PI controller: Force Tracking Performance

In figure 4.17, the force error is presented for both trajectories.

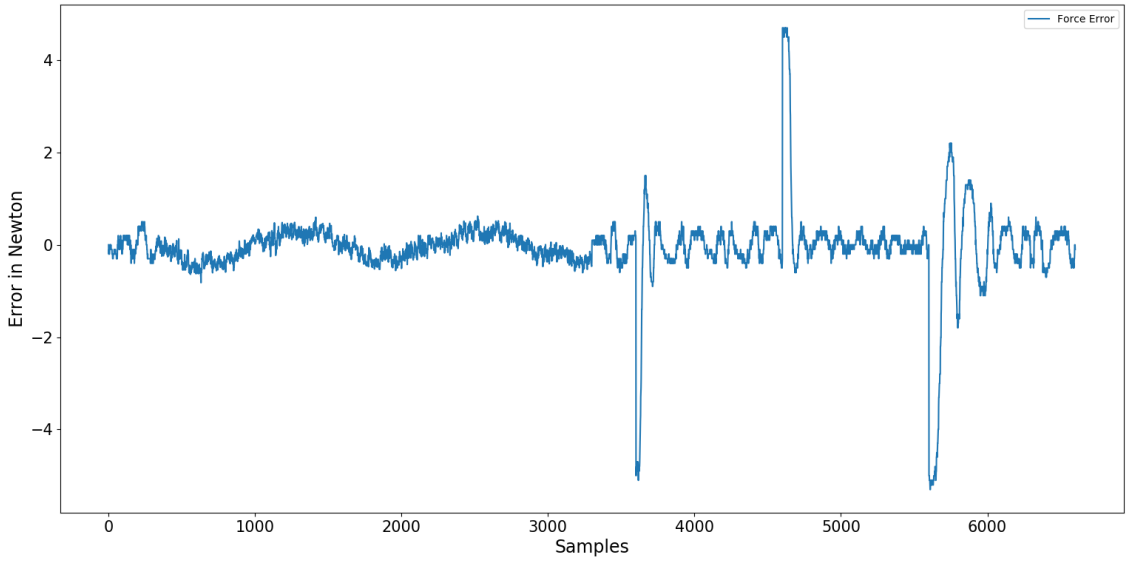


Figure 4.17: *PI* controller : Force Tracking Error

Finally, this controller as well, was tested in it's adaptability by tracking a different trajectory without re-tuning the K_p and K_I parameters. The controller's performance is presented in Figure 4.18:

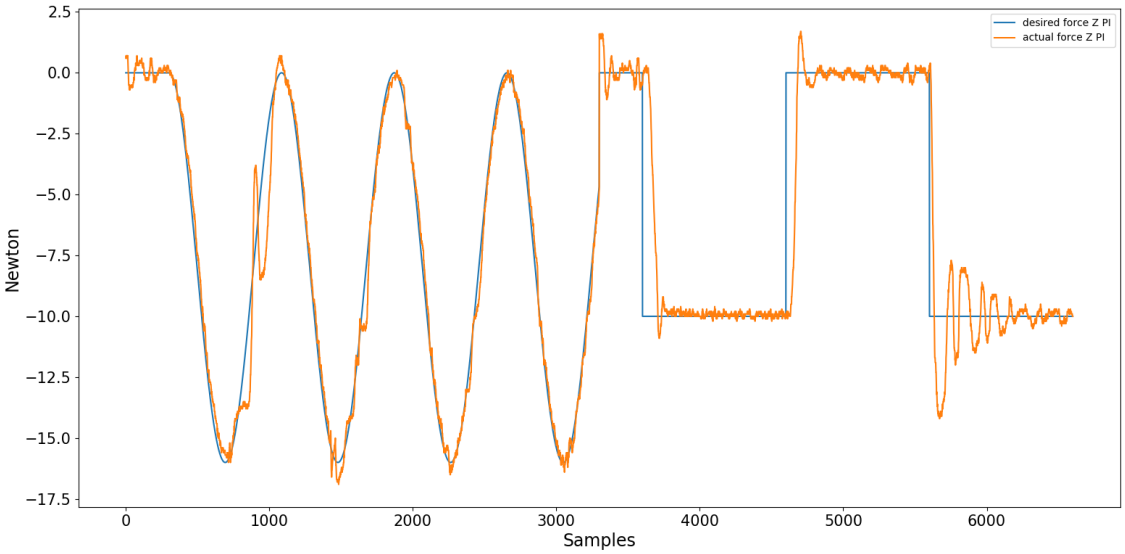


Figure 4.18: *PI* controller: Force Tracking Performance for the new trajectories without tuning

In figure 4.19, the force error is presented for both new trajectories. (Sinusoidal, Step).

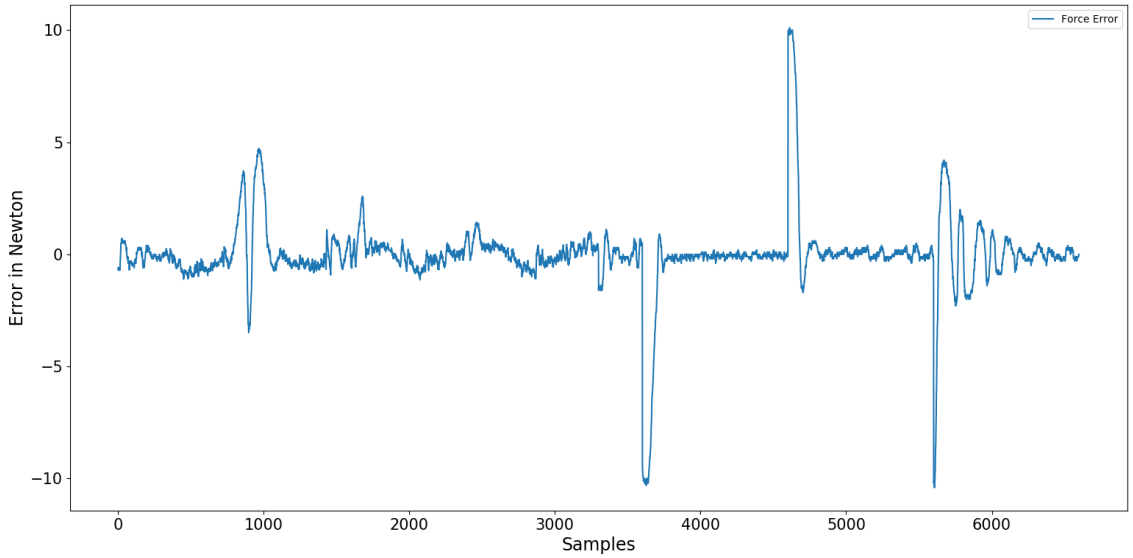


Figure 4.19: PI controller: Force Tracking Error for the new trajectories

P-only with feedforward

Now, the feedforward term will be introduced to the previous controllers in order to improve their performance. Note, that the feedforward term, is the velocity that the neural network estimates. The P-only controller with a feedforward term, is given by :

$$V_p = \dot{p}(t) + K_p(F_r(t) - F(t)) \tag{4.5}$$

The K_p parameter of this control scheme was tuned differently for each one of the two trajectories, to achieve the best possible performance and $\dot{p}(t)$ is the neural network's velocity output.
The controller's performance is presented in Figure 4.20 :

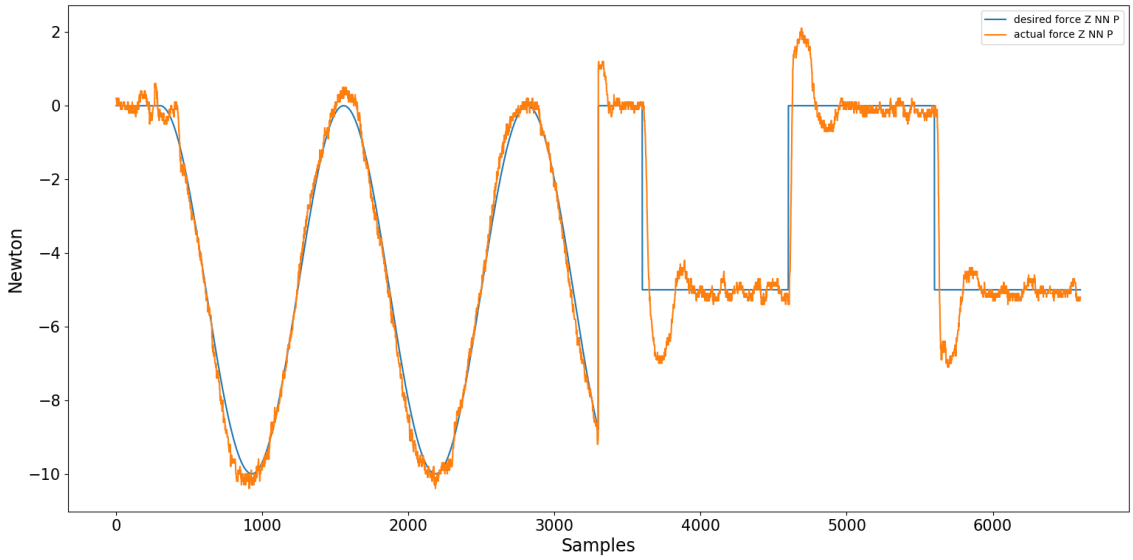


Figure 4.20: P with feedforward controller: Force Tracking Performance

In figure 4.21, the force error is presented for both trajectories.

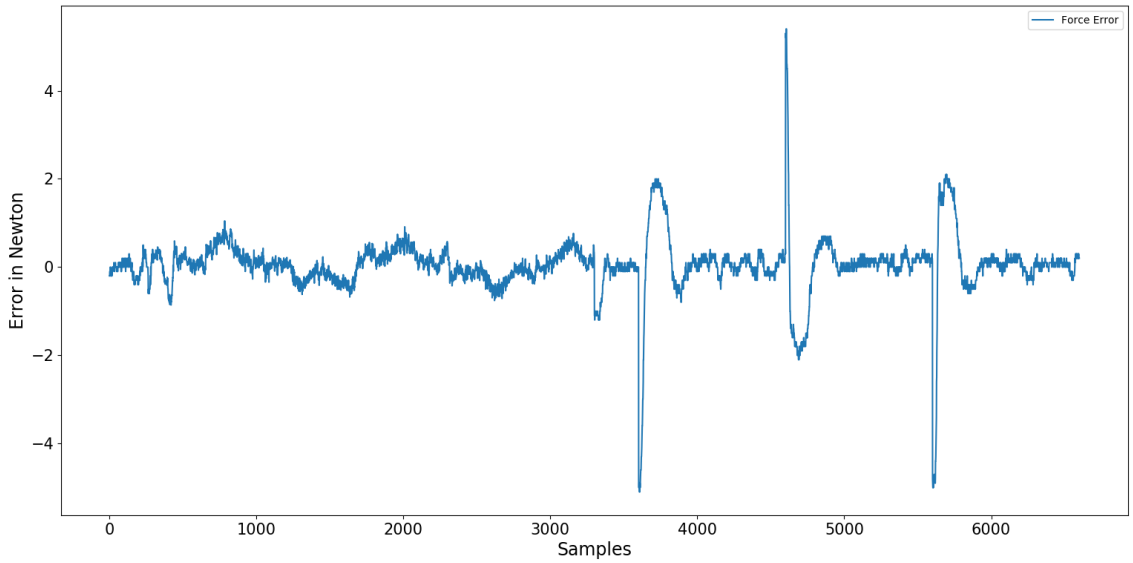


Figure 4.21: P with feedforward controller: Force Tracking Error

Finally, this controller was tested in it's adaptability by tracking a different trajectory without re-tuning the K_p parameter. The controller's performance is presented in Figure 4.22 :

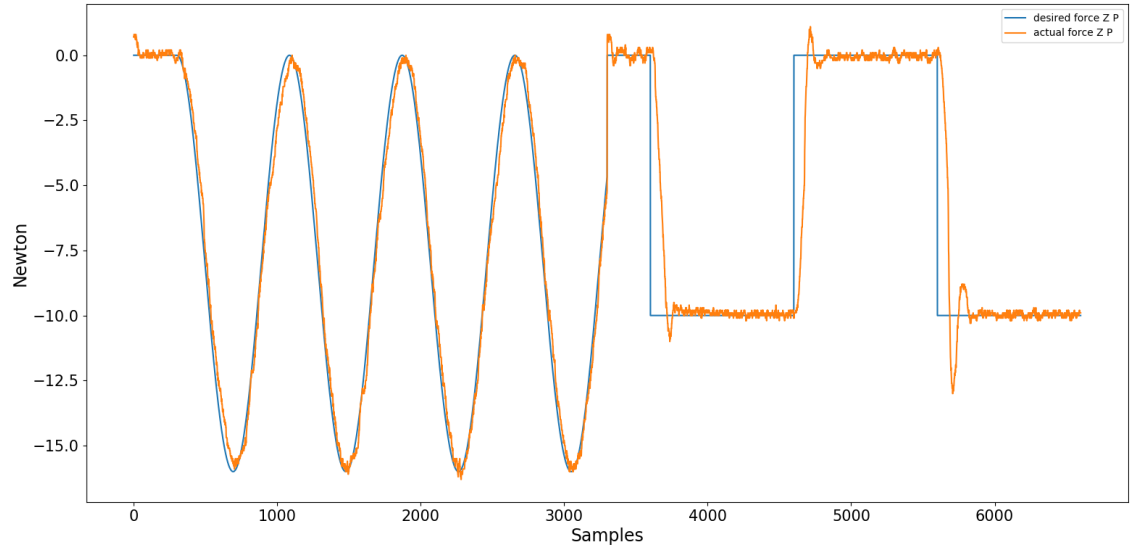


Figure 4.22: P with feedforward controller: Force Tracking Performance for the new trajectories without tuning

In figure 4.23, the force error is presented for both new trajectories (Sinusoidal, Step).

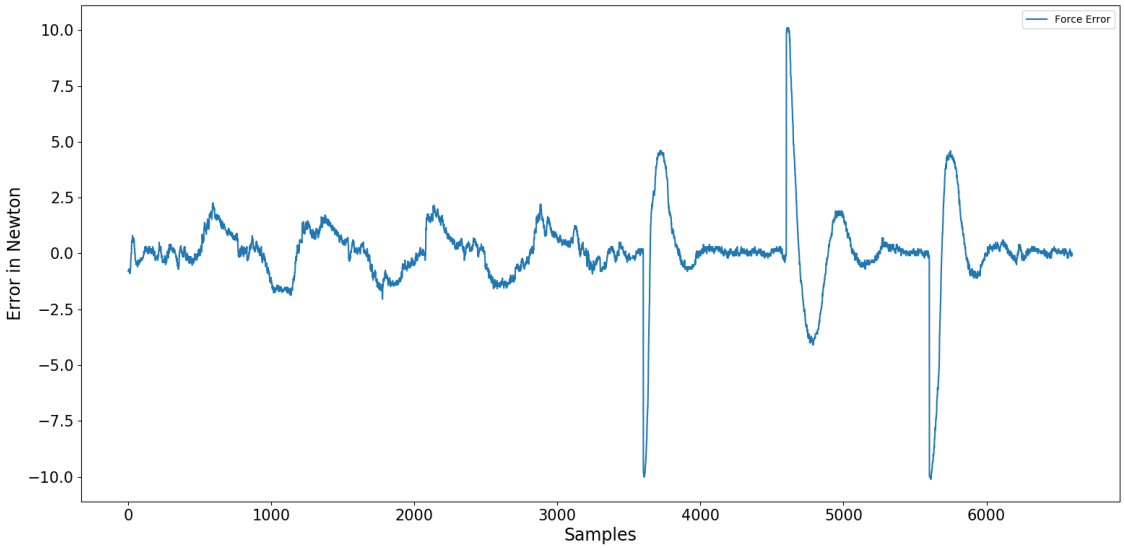


Figure 4.23: P with feedforward controller: Force Tracking Error for the new trajectories

PI with feedforward

Now, the integral term is integrated into the previous P controller with the feedforward term, resulting in a PI controller with feedforward.

$$V_p = \dot{p}(t) + K_p(F_r(t) - F(t)) + K_I \int (F_r(t) - F(t))dt \tag{4.6}$$

The K_p and K_I parameters were tuned differently for each one of the two trajectories, as before, to achieve the best possible performance for this controller scheme.
This controller's performance is presented in Figure 4.24:

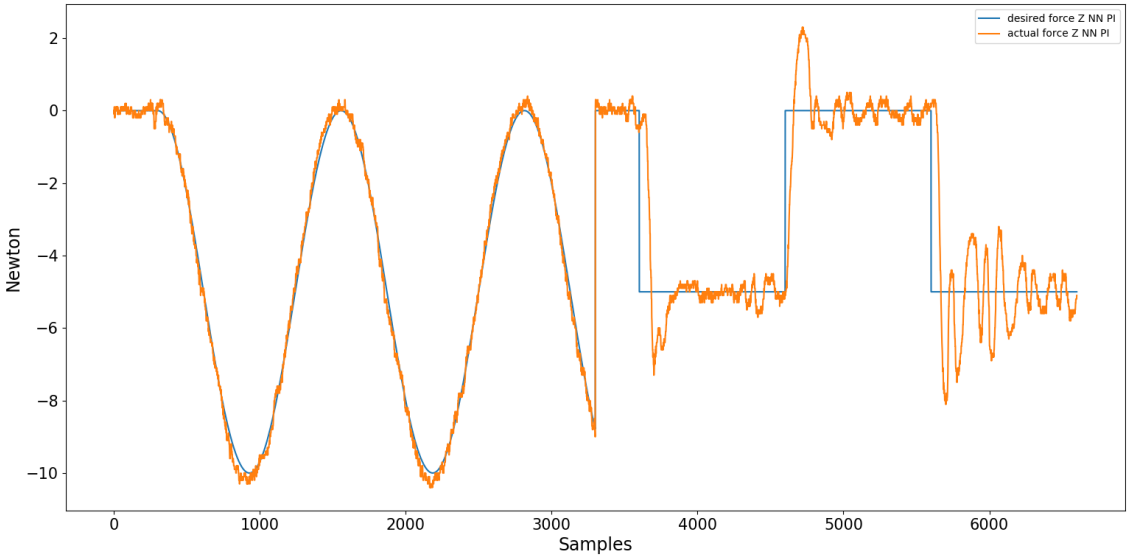


Figure 4.24: PI with feedforward controller: Force Tracking Performance

In figure 4.25, the force error is presented for both trajectories.

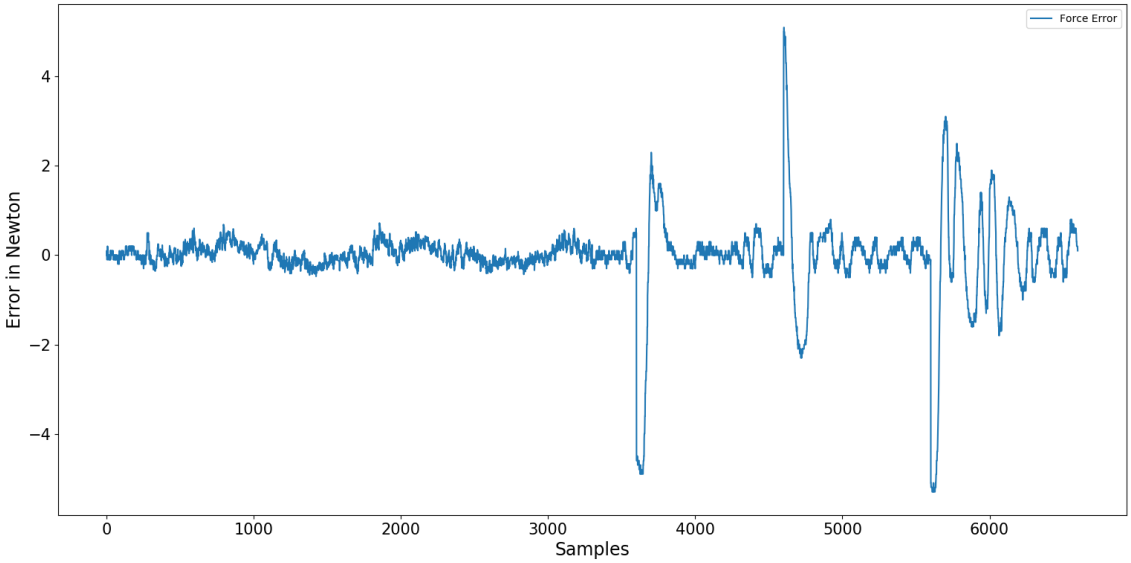


Figure 4.25: *PI* with feedforward controller: Force Tracking Error

Finally, this controller was tested in it's adaptability by tracking a different trajectory without re-tuning the K_p and K_I parameters. The controller's performance is presented in Figure 4.26:

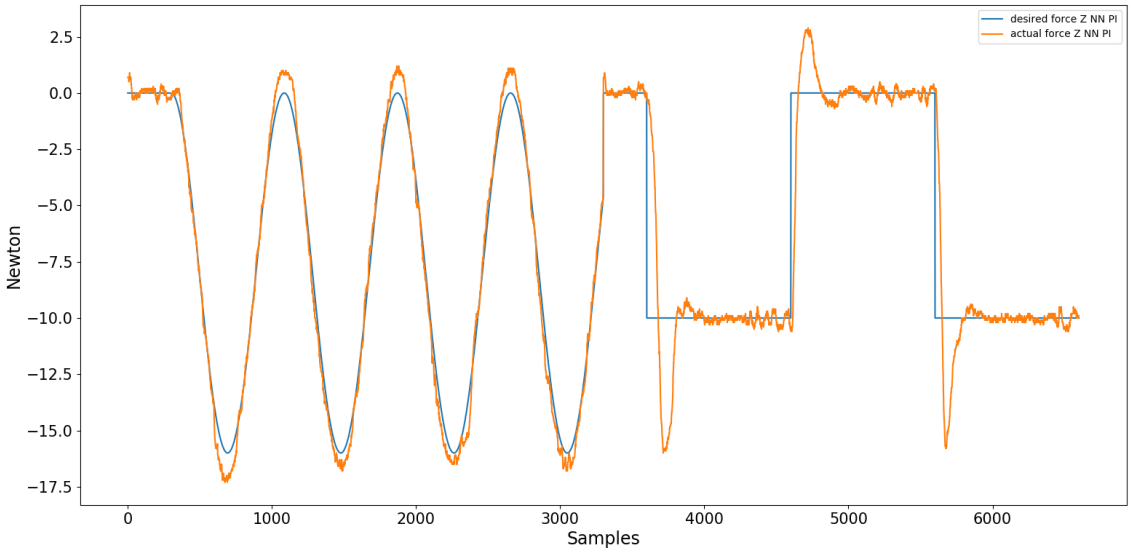


Figure 4.26: *PI* with feedforward controller: Force Tracking Performance for the new trajectories without tuning

In figure 4.27, the force error is presented for both new trajectories (Sinusoidal, Step).

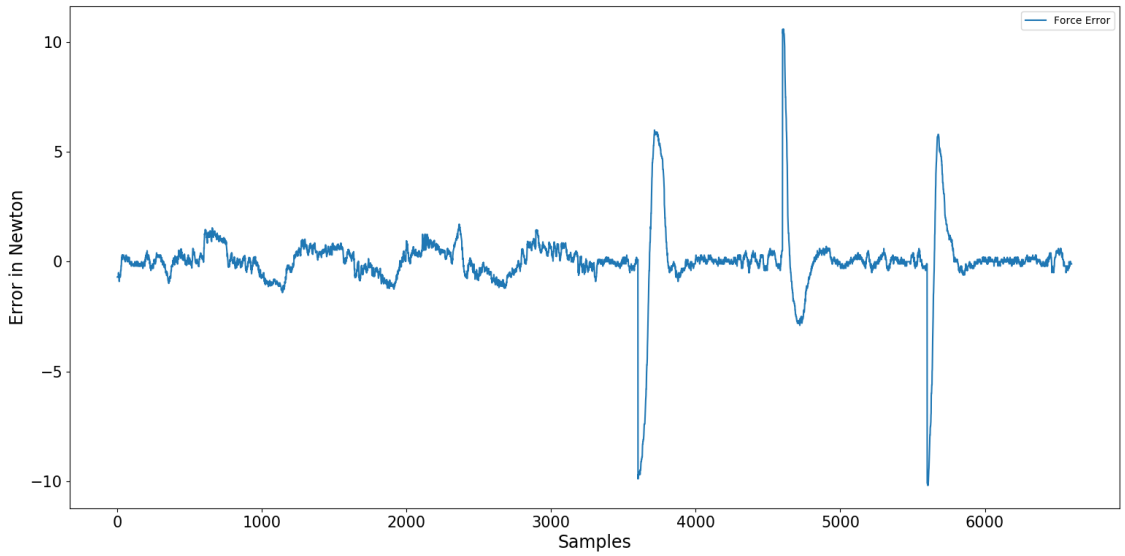


Figure 4.27: *PI* with feedforward controller: Force Tracking Error for the new trajectories

Reinforcement Learning

Finally, an optimal adaptive controller was designed, using reinforcement learning. This controller scheme, tries to minimize a reward function that contains as parameters, both the force error and integral force error ($\Delta F, \int \Delta F dt$), and the controller's previous output. However, the controller's output was only used for the step trajectory, because only in that case it yielded better results (which is logical if one considers that in the step trajectory a velocity (policy) close to zero is required, when tracking a constant force). The reward function, as discussed before, is highly dependant on the problem formulation and the task at hand, thus two slightly different Reward functions were used for the two trajectories. The reward functions that the algorithm was trying to minimize are presented below:

- For the Sinusoidal :

$$R(t) = \begin{bmatrix} \Delta F & \int \Delta F dt \end{bmatrix} \cdot \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} \cdot \begin{bmatrix} \Delta F \\ \int \Delta F dt \end{bmatrix} \tag{4.7}$$

- For the Step trajectory :

$$R(t) = \begin{bmatrix} \Delta F & \int \Delta F dt \end{bmatrix} \cdot \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} \cdot \begin{bmatrix} \Delta F \\ \int \Delta F dt \end{bmatrix} + u * R * u \tag{4.8}$$

The tuning parameters for the Actor Critic that was designed for this problem are presented below in Table 4.1 :

	Actor LR	Critic LR	Q_{11}	Q_{22}	Exploration	R	RBF Centers
Sinusoidal	1e-4	1e-4	-1e7	-1e7	5e-7	0.0	41
Step	1e-5	1e-4	-1e6	-1e7	1e-7	-1e7	41

Table 4.1: Actor Critic Controller Parameters

In table 4.1, the Actor and Critic Learning Rates were initialized higher in the first phases of training, and were gradually decreased. The learning rate can be translated as, how "big steps" the algorithm is taking in the gradient's negative direction, and thus we want "high steps" at early stages for fast convergence, and in the end, "small steps" for accuracy.

A similar strategy was also used while choosing the exploration factor. The exploration's purpose is for the algorithm to explore the entirety of the state space and assign weights to the states. Therefore, in the early training stages, the explorations needs to be noticeable and thus in both trajectories it was set to be a random normal distribution with zero mean and standard deviation: $5e-5$. Although, this is still a small exploration factor, it was enough to explore the entire state space, while preventing the manipulator from losing contact with the object, as well as not damaging the equipment.

The Q and R weights were tuned after a considerable amount of trial and error, until weights that yield adequate results were found. It can also be observed, that only the diagonal weights of Q matrix were tuned, and the rest were set to zero. The reasoning behind this is the fact that we don't want the reward function to correlate the force error, with the force error's integral. Recall that the RL agent uses the discounted cumulative reward as the learning criterion and thus as the integral keeps increasing over time, the integral action would overpower the error itself. Correlating the two "states" would yield in a less effective reward function, and therefore it was decided to use a diagonal Q matrix.

Finally, the RBF parameters were chosen to be 41, which was a adequate split of the state space that the parameters were moving, as well as computationally acceptable for an online implementation. Note that with more RBF parameters, it could be possible to achieve better accuracy, but with considerable computational cost. Therefore, for the Force Error, the center were split into 41 equal spaces between $[-2, 2]$, whereas for the integral error, the space was $[-1, 1]$. One more reasoning behind the chosen number of parameters was the force censor's accuracy, which was 0.1 N.

The controller was trained differently for each one of the two trajectories, to achieve the best possible performance for this controller scheme. For both trajectories the training process was similar, in the weight changes as well as the time required for the algorithm to learn the model. More specifically, the sample time required for the algorithm to be trained in each trajectory was around 6600 samples. This can be translated in 2 Episodes, due to the way the trajectories were built (each trajectory consists of 3300 samples). However, in order to avoid any potential damage to the equipment, the training process was being constantly monitored and stopped when the exploration of the algorithm.

The trajectory tracking results for the Actor Critic controller are shown in Figure 4.28:

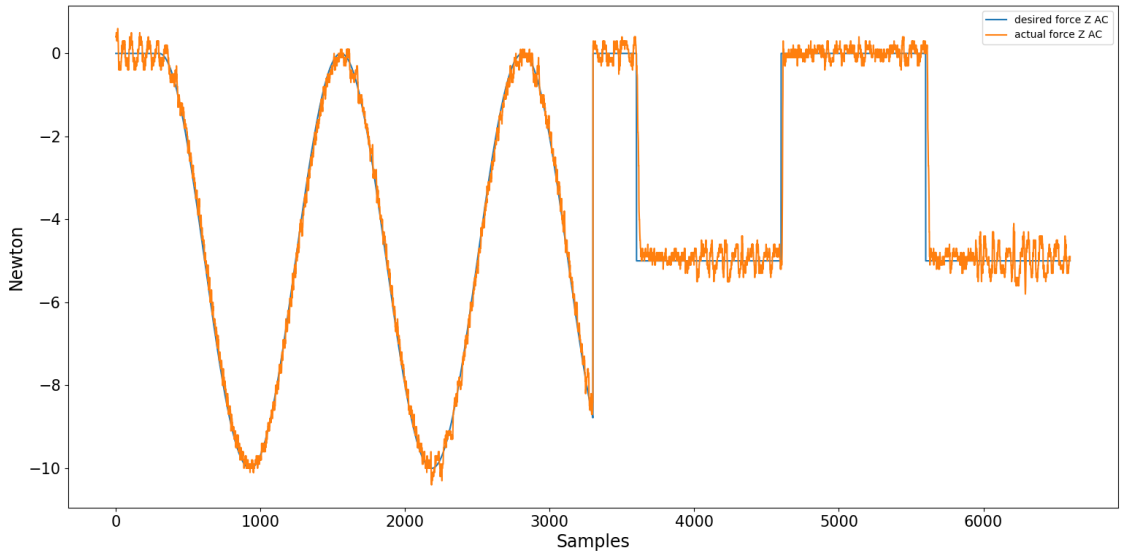


Figure 4.28: AC controller: Force Tracking Performance

In figure 4.29, the force error is presented for both trajectories.

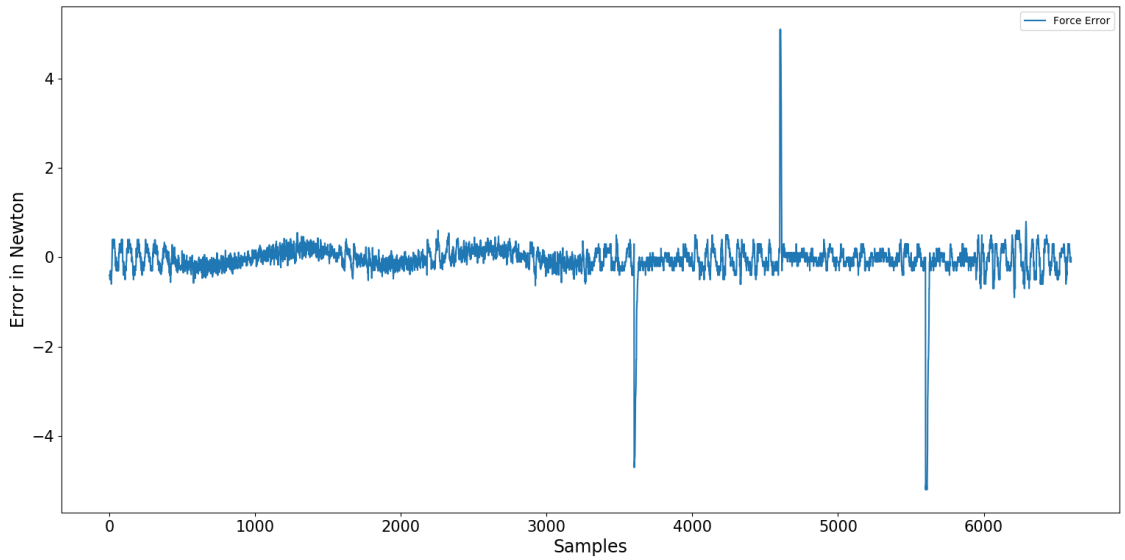


Figure 4.29: AC controller: Force Tracking Error

Finally, this controller was tested in its adaptability by tracking a different trajectory without re-tuning any of the parameters. It was discussed before, that the controller was tuned differently for the 2 different trajectories. This means that, even this controller, is case sensitive. However, it was observed in the trials that it can still adapt after a certain amount of time, even in different trajectories. The controller's performance in the 2 new trajectories, without retraining the algorithm, is presented below in Figure 4.30 :

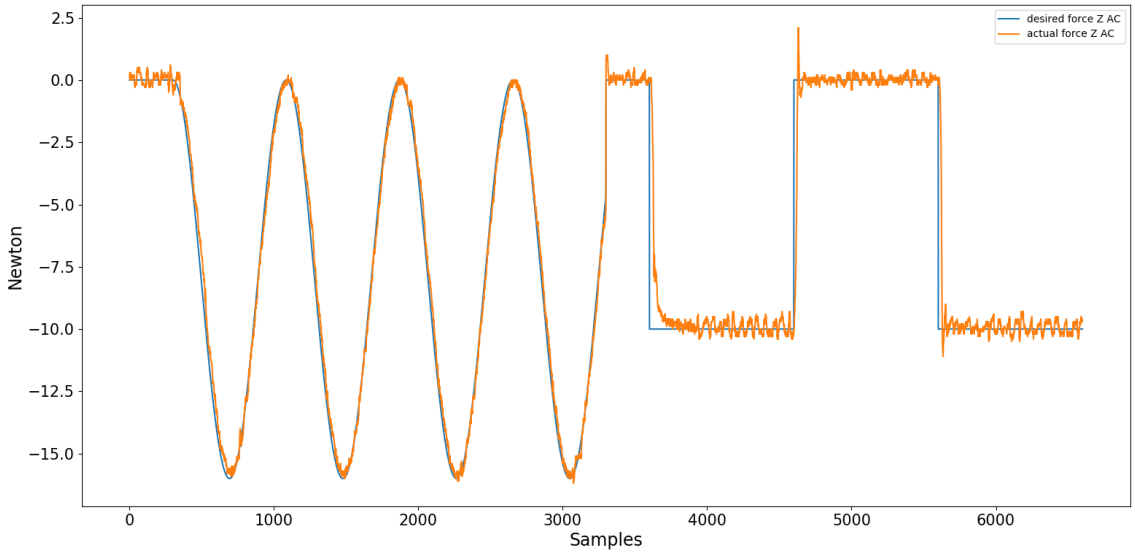


Figure 4.30: AC controller: Force Tracking Performance for the new trajectories without tuning

In figure 4.31, the force error is presented for both new trajectories (Sinusoidal, Step).

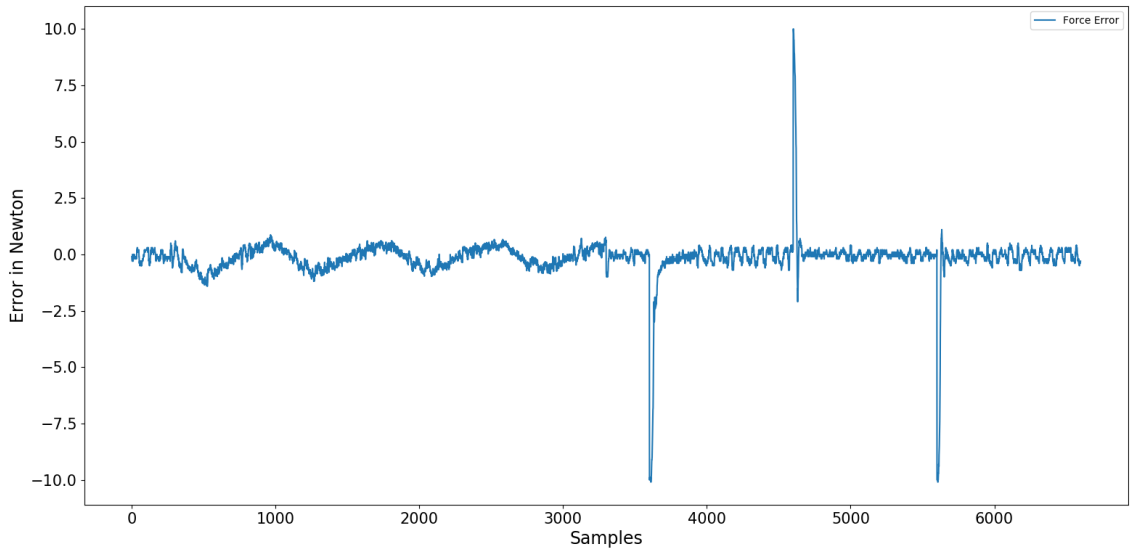


Figure 4.31: AC controller : Force Tracking Error for the new trajectories

A really important remark has to be pointed out here. As discussed in section 4.1, the longer the manipulator is interacting with the sponge, the more the elastic hysterestic behaviour of the sponge is revealed making it slower to return to its initial position. This means that sometimes, the manipulator is losing contact with the material. This is crucial, especially for the controllers that make use of the neural network, because if the manipulator loses contact, the system's model is changing, and the network has not been trained in that model, like in the last part of plot 4.24. However, this can also happen without the existence of the neural network and

result in oscillations as in the final part of plot 4.16. Therefore, it was considered important to present this behaviour in the final results and account for it when discussing each controller's overall performance.

4.4.4 Results

By observing each controller's individual performance it is not clear how well each one is performing compared to the others. Therefore, the mean square error of each controller for the sinusoidal trajectory will be presented, and for the step trajectory, the transition period as well as the the mean squared error of the steady state will be assessed. These metrics will clearly define each controller's performance and help clarify the difference in the performance of the control schemes. For the calculation of the transition period of each controller in the step trajectory, the mean of the transitions of all 3 steps was considered, and thus the samples of each will be presented, to highlight the elastic hysterestic properties of the sponge. Moreover, in the results presented below, the trajectory for which the controllers were tuned fore, will be denoted as "old", and the trajectories that were used to assess each controller's adaptability will be denoted as "new".

	Trajectory	Mean Squared Error
AC	old	0.038
	new	0.19
PI feedforward	old	0.042
	new	0.37
P feedforward	old	0.095
	new	0.849
PI	old	0.071
	new	0.846
P	old	0.088
	new	0.4

Table 4.2: MSE for both the Old and New Sinusoidal Trajectories

From table 4.2, it can be observed that, for the "old" trajectory, the use of the neural network improved the performance of the simple P and PI controllers considerably. This makes the use of the neural network to estimate the sponge's model imperative, if accuracy is important. However, it can also be observed that the performance of all the controllers deteriorate when the "new" trajectory is introduced. However, the Actor Critic controller, apart from being exceptionally better in the "old" trajectory than all other controllers, it has the least amount of deterioration when the "new" trajectory is introduced. The reason behind this is the adapting capabilities of the

Reinforcement Learning based control scheme. It is also important to note, that in time the AC controller will become even better, as it will keep adjusting its gains to achieve the best performance.

	Trajectory	Transition Period In Samples	Steady State In Seconds
AC	old	(18,12,23) : 17.66	0.05
	new	(55,34,33) : 40.6	0.06
PI feedforward	old	(183,165,568) : 305.3	0.068
	new	(194,174,171) : 179.6	0.053
P feedforward	old	(199,158,167) : 174.6	0.051
	new	(211,425,335) : 323.6	0.055
PI	old	(76,60,384) : 173	0.066
	new	(102,114,407) : 207	0.051
P	old	(172,102,420) : 232	0.068
	new	(135,115,195) : 148	0.02

Table 4.3: Transition Period and Steady State MSE for the Step Trajectory, and both the Old and New Trajectories

For the step trajectory, however, the feedforward term of the neural network does not seem to have a significant improvement in trajectory tracking, more specifically in the transition period. This happens due to the fact that the manipulator is developing high velocities to achieve a fast transition, which are hard to reduce, and thus an overshoot occurs. In general, if the manipulator does not lose contact with the material it can achieve an adequate performance and when the force remains constant even outperforms the simple P and PI controllers. However, one needs to keep in mind that this can happen any time and thus cannot neglect this possibility. Finally, as with the sinusoidal trajectory, in the step trajectory as well the AC's performance as it can be observed is exceptionally better than all other controllers, as well as it's adaptability to a different trajectory. Moreover, this control scheme has the fastest response out of all, and the least overshoot between all the controller schemes.

5

Conclusion And Future Work

5.1 Conclusion

The purpose of this master thesis project, was to investigate data driven modeling for robotic manipulation of deformable objects. More specifically, how a neural network can be trained, using supervised learning, in order to be used in an online force tracking application. The neural network model was trained using data collected from the real "robot-sponge" system, and was approximating a function, with inputs: a) Force (exerted on the end-effector), b) Current end-effector Position, c) Previous end-effector Position, and one output: end-effector linear velocity. This model was then used as a feedforward term of a P-only, and a PI control schemes. The P-only and PI control scheme were also tested without the feedforward term from the neural network, in order to showcase how the neural network model approximation can improve performance. These four (4) different control schemes were tested and assessed on two different trajectories (Sinusoidal, Step). It was observed that the feedforward term, was significantly improving the manipulator's capabilities of tracking an altering force trajectory, (e.g Sinusoidal). However, when the system needs to follow a constant force, or a discontinuous force trajectory like a Step function, it was observed that the control schemes without the feedforward term were outperforming the others.

This raised the question, if a controller exists that performs equally well in both kinds of trajectories. A reinforcement learning based controller was designed in order to answer this question. This controller, still had to be trained for each trajectory differently, but due to it's online learning capabilities, it is always adapting to any changes of the trajectory the robot is tracking, or any small changes to the system (e.g wear and tear of the robot links, deterioration of the object etc). As a result this controller scheme outperforms all the others in both trajectories, making this an optimal control scheme, as it was observed in the results chapter.

Finally, the online learning capabilities of the Actor-Critic controller, were assessed alongside the static control schemes. The P-only and PI controllers, with or without the feedforward term has to be re trained for every new trajectory that is introduced to the system. The dynamic feature of the AC control scheme make it able to adapt in any new trajectory without further tuning. This is an extremely important feature of the AC control scheme, since it can save maintenance time and be more efficient than other controllers.

5.2 Future Work

As a continuation of this project, the controllers that was designed and tested can be tuned and assessed in all three dimensions, in movements that are in the $3D$ space. Moreover, this study was focused in the state space, and therefore the robotic manipulator was commanded in the Cartesian space. The studies can be extended in order for the control scheme to be directly in the joint space, thus reducing the computational power that is required for the inverse kinematics. Finally, one can use this study as a baseline to identify, using a neural network, different object models and then apply this knowledge in force control tasks, while concurrently identifying which object it is.

Bibliography

[1] K. H.-Z. Amir Haddadi, “Real-Time Identification of HuntCrossley Dynamic Models of Contact Environments,”IEEE Transactions on Robotics, vol. 28,no. 3, pp. 555–566, 2012.

[2] B. Frank, R. Schmedding, C. Stachniss, M. Teschner and W. Burgard, "Learning the elasticity parameters of deformable objects with a manipulation robot," 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, Taipei, 2010, pp. 1877-1883. doi: 10.1109/IROS.2010.5653949

[3] K.-J. Bathe. Finite Element Procedures. Prentice Hall, 2 edition, 1995.

[4] M. Hauth and W. Strasser. Corotational Simulation of Deformable Solids. In Proc. of WSCG, 2004.

[5] M. Mueller and M. Gross. Interactive Virtual Materials. In Graphics Interface, 2004.

[6] G. Bianchi, B. Solenthaler, G. Szekely, and M. Harders. Simultaneous Topology and Stiffness Identification for Mass-Spring Models based on FEM Reference Deformations. In Medical Image Computing and Computer-Assisted Intervention (MICCAI), 2004.

[7]] B. Lloyd, G. Szekely, and M. Harders. Identification of Spring Parameters for Deformable Object Simulation. IEEE Trans. on Visualization and Computer Graphics, 13(5):1081–1094, 2007.

[8] S. Burion, F. Conti, A. Petrovskaya, C. Baur, and O. Khatib. Identifying Physical Properties of Deformable Objects by Using Particle Filters. In Proc. of the Int. Conf. on Robotics Automation (ICRA), 2008.

[9] S. J. N. G. Olalekan Ogunmolu, Xuejun Gu, “Nonlinear Systems Identification Using Deep Dynamic Neural Networks,”American Control Conference,2017, 2016.

[10] Tushar Gupta Deep Learning: Feedforward Neural Network. Jan 5, 2017 from https://towardsdatascience.com/deep-learning-feedforward-neural-network-

26a6705dbdc7

[11] B. Recht, “A Tour of Reinforcement Learning: The View from Continuous Control,”Optimization and Control (math.OC); Machine Learning (cs.LG);Machine Learning (stat.ML), 2018.

[12] J. R. . L. Whitcomb, “Adaptive Force Control of Position/Velocity Controlled Robots: Theory and Experiment,”IEEE Transactions on Robotics and Automation, vol. 18, no. 2, pp. 121–137, 2002.

[13] José Sanchez, Juan Antonio Corrales Ramon, Belhassen-Chedli Bouzgarrou, Youcef Mezouar. Robotic Manipulation and Sensing of Deformable Objects in Domestic and Industrial Applications: A Survey. International Journal of Robotics Research, SAGE Publications, In press, <10.1177/0278364918779698>. <hal-01816189>

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in Advances in neural information processing systems, 2012, pp. 1097–1105.

[15] T. J. Sejnowski and C. R. Rosenberg, “Parallel networks that learn to pronounce English text,” Complex systems, vol. 1, pp. 145–168, 1987.

[16] W. Zaremba, I. Sutskever, and O. Vinyals, “Recurrent neural network regularization,” CoRR, vol. abs/1409.2329, 2014.

[17] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2323, 1998.

[18] I. Sutskever, “Training Recurrent neural Networks,” PhD thesis, p. 101, 2013.

[19] S. Hochreiter and J. Schmidhuber, “Long short-term memory.” Neural computation, vol. 9, no. 8, pp. 1735–80, 1997.

[20] T. Yoshikawa, “Force control of robot manipulators,” in Proc. IEEE Int. Conf. Robotics and Automation, vol. 1, 2000, pp. 220–226.

[21] Karayiannidis, Yiannis Rovithakis, George Doulgeri, Zoe. (2006). Force/-Position Tracking for a Robotic Finger in Compliant Contact with a Surface using Neuro-Adaptive Control. IEEE International Symposium on Intelligent Control - Proceedings. 10.1109/CACSD-CCA-ISIC.2006.4776881.

[22] Mitsioni, Ioanna Karayiannidis, Yiannis A. Stork, Johannes Kragic, Danica. (2019). Data-Driven Model Predictive Control for Food-Cutting.

[23] N. Hogan, “Impedence control: An approach to manipulation: Part I—Theory,” J. Dynamic Syst., Measurement Contr., vol. 107, pp. 1–7, Mar. 1985.

[24] —- “Impedence control: An approach to manipulation: Part II—Implementation,” J. Dynamic Syst., Measurement Contr., vol. 107, pp. 8–16, Mar. 1985.

[25] —- “Impedence control: An approach to manipulation: Part III—Applications,” J. Dynamic Syst., Measurement Contr., vol. 107, pp. 17–24, Mar. 1985.

[26] M. Raibert and J. Craig, “Hybrid position/force control of manipulators,” J. Dynamic Syst., Measurement Contr., vol. 103, no. 2, pp. 126–133, Feb. 1981.

[27] M. Mason, “Compliance and force control for computer controlled manipulators,” IEEE Trans. Syst., Man, Cybern., vol. SMC-11, pp. 418–432, June 1981.

[28] Michael A. Nielsen, "Neural Networks and Deep Learning", Last update: Tue Oct 2 11:05:11 2018 Determination Press, 2015

[29] McCulloch W.S. , Pitts W. Bulletin of Mathematical Biophysics (1943) 5: 115. https://doi.org/10.1007/BF02478259

[30] Backpropagation. Brilliant.org. Retrieved 17:51, May 23, 2019, from https://brilliant.org/wiki/backpropagation/

[31] Kingma Diederik, Ba Jimmy. (2014). Adam: A Method for Stochastic Optimization. International Conference on Learning Representations.

[32] Sefik Ilkin Serengil : Softsign as a Neural Networks Activation Function. Sefik Serengil, November 10, 2017.

[33] Prince Grover: 5 Regression Loss Functions All Machine Learners Should Know: Choosing the right loss function for fitting a model, June 5, 2018, from https://heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0

[34] Anish Singh Walia: Activation functions and it’s types-Which is better?, May 29, 2017, from https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f

[35] C. Duchi, John Hazan, Elad Singer, Yoram. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. Journal of Machine Learning Research. 12. 2121-2159.

[36] N. Dauphin, Yann Vries, Harm Chung, Junyoung Bengio, Y. (2015). RMSProp and equilibrated adaptive learning rates for non-convex optimization. arXiv. 35.

[37] Butterworth Filter Design. electronics-tutorials.ws. Retrieved 19:21, May 31, 2019, from https://www.electronics-tutorials.ws/filter/filter_8.html

[38] Nicolas Lauzier Robot Force Control: An Introduction, Feb 26, 2012 9:43:00 PM

[39] Alex Owen-Hill. Robotics Research 101: Getting Started with Force Control, on Feb 17, 2016 7:00:00 AM

[40] Villani, Luigi. (2013). Force Control in Robotics. 10.1007/978-1-4471-5102-9_169-1.

[41] Whitney DE (1977) Force feedback control of manipulator fine motions. ASME J Dyn Syst Meas Control 99:91–97

[42] Mathieu Bélanger-Barrette. How Do Industrial Robots Achieve Compliance, on Feb 18, 2014 4:03:00 PM

[43] Villani L., De Schutter J. (2008) Force Control. In: Siciliano B., Khatib O. (eds) Springer Handbook of Robotics. Springer, Berlin, Heidelberg

[44] Integral Action and PI Control. controlguru.com. Retrieved 17:10, June 08, 2019, from https://controlguru.com/integral-action-and-pi-control/

[45] Siciliano B., Villani L. (1999) Direct Force Control. In: Robot Force Control. The Springer International Series in Engineering and Computer Science (Robotics: Vision, Manipulation and Sensors), vol 540. Springer, Boston, MA

[46] Doya, Kenji. (2000). Reinforcement Learning in Continuous Time and Space. Neural computation. 12. 219-45. 10.1162/089976600300015961.

[47] Ivo Grondman, Lucian Busoniu, Gabriel Lopes, Robert Babuska. A survey of actor-critic reinforcement learning: standard and natural policy gradients. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, Institute of Electrical and Electronics Engineers, vol 42, no. 6, pp.1291-1307, 2012

[48] S. T., "An intro to Advantage Actor Critic methods: lets play Sonic the Hedgehog!," (2018, Jul 26).[Online]. Available at :https://medium.freecodecamp.org/an-intro-to-advantage-actor-critic-methods-lets-play-sonic-the-hedgehog-86d6240171d.

[49] R. S. Sutton and A. G. Barto,Introduction to Reinforcement Learning.Cambridge, MA, USA: MIT Press, 1st ed., 1998.

[50] Moustafa Alzantot Deep Reinforcement Learning Demysitifed (Episode 2)-—-Policy Iteration, Value Iteration and Q-learning, Jul 8, 2017

[51] Shaked Zychlinski, The Complete Reinforcement Learning Dictionary, The Reinforcement Learning Terminology, A to Z, Feb 32, 2019

[52] R. S. Sutton, "Learning to predict by the methods of temporal differences,"Machine Learning, vol. 3, pp. 9–44, Aug 1988.

[53] T. Degris, P. M. Pilarski, and R. S. Sutton, "Model-free reinforcement learning with continuous action in practice," in 2012 American Control Conference (ACC), pp. 2177–2182, June 2012

[54] H. Kimura, T. Yamashita, and S. Kobayashi, "Reinforcement learning of walking behavior for a four-legged robot," in Proceedings of the 40th IEEE Conference on Decision and Control (Cat. No.01CH37228), vol. 1,pp. 411–416 vol.1, Dec 2001.

[55] L. Baird and A. Moore, "Gradient descent for general reinforcement learning," in Proceedings of the 1998 Conference on Advances in Neural Information Processing Systems II, (Cambridge, MA, USA), pp. 968–974, MIT Press, 1999.

[56] D. Standage, D.-H. Wang, R. P. Heitz, and P. Simen, "Toward a unified view of the speed-accuracy trade-off,"Frontiers in Neuroscience, vol. 9,p. 139, 2015.

[57] I. Grondman, L. Busoniu, R. Babuska, and E. Schuitema, "Efficient Model Learning Methods for Actor Critic Control,"IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), vol. 42, no. 3,pp. 591–602, 2012.

[58] H. van Hasselt, "Reinforcement learning in continuous state and actions paces," 2012.

[59] Y. Wu, H. Wang, B. Zhang, and K. Du, "Using radial basis function networks for function approximation and classification," in ISRN Applied Mathematics, vol. 2012, pp. 1–34, 2012

[60] Chris McCormick, Radial Basis Function Network (RBFN) Tutorial, 15 Aug 2013. [Online]. Available at : https://mccormickml.com/2013/08/15/radial-basis-function-network-rbfn-tutorial/

[61] Bhasin, Shubhendu. (2019). REINFORCEMENT LEARNING AND OPTIMAL CONTROL METHODS FOR UNCERTAIN NONLINEAR SYSTEMS.

[62] Tarang Shah. About Train, Validation and Test Sets in Machine Learning. Dec 6, 2017. [Online]. Available at : https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7

[63] Shing-Yan LOO , S.H. TANG Mashohor Syamsiah. Active and Passive Compliance Mechanisms in Legged Robot Locomotion. Department of Mechanical and Manufacturing Engineering, Department of Computer and Communication Systems Engineering, Faculty of Engineering, Universiti Putra Malaysia. gs44638@student.upm.edu.my