Adaptive Force Trajectory Tracking for Robotic Interaction with Soft Objects

Master's thesis in Systems, Control and Mechatronics

NOYAN ÜZER

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

Master's thesis 2021

Adaptative Force Trajectory Tracking for Robotic Interaction with Soft Objects

Noyan Üzer

Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
Gothenburg, Sweden 2021

Adaptative Force Trajectory Tracking
for Robotic Interaction with Soft Objects
Noyan Üzer

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

Master's Thesis 2021
Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2021

Adaptative Force Trajectory Tracking for Robotic Interaction with Soft Objects

Noyan Üzer
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Controlling the interaction between a robotic manipulator and its environment is a necessity for the execution of a plethora of practical tasks. However, without the knowledge of the contact dynamics, sufficient control performance is hard to achieve. A pure motion control strategy can perhaps be favored in this case. If, however, the task is not carefully planned, this may end up with the breakage of the robot. A more preferable approach to take would be to use a mathematical model to describe the dynamics of the environment then based on this model design a control law which is capable of adaptively adjusting the position/velocity trajectory followed by the end-effector depending on the contact force readings.

In this thesis, the latter strategy was followed. In order to approximate the environment dynamics, two models were used: The linear Kelvin-Voight model and the nonlinear Hunt-Crossley model. The linear model while being widely used and easy to work with was shown not to be as consistent with physics of contact as its nonlinear counterpart. Based on these contact models two sets of force controllers were proposed for velocity/position controlled robotic manipulators: A set of Kelvin-Voigt based adaptive controllers and a set of Hunt-Crossley based neuro-adaptive controllers. It was shown that when the robot is under exact or asymptotically exact inner loop velocity control, the controllers provide asymptotic force trajectory tracking.

Finally, a simple artificial neural network was designed to approximate the dynamical properties of a compliant environment, in this case a sponge, for the purpose of creating a testing environment. A series of comparative experiments were performed and the step responses and tracking capabilities of the controllers were investigated.

Keywords: Adaptive force control, neuro-adaptive force control, robot force control, velocity based force control, neural network based function approximation, contact modeling

Acknowledgements

This has been a long and weird journey for me that was full of ups and downs. I would like thank my supervisor Yiannis Karayiannidis, and my family for bearing with my antics through out all this and I humbly apologize from them for the numerous times I tested their nerves. I also would like to thank Konstantinos Alfonsos for his work on neural networks as it greatly helped me with this project.

Noyan Üzer, Gothenburg, October 2021

Contents

List of Figures

List of Tables

1 Introduction

In today's world, robots find themselves a place in a wide array of both industrial and non-industrial applications, ranging from toys such as the AIBO™ to surgical systems like the Da Vinci™. A great deal of these applications and tasks however, require the robots to be in contact with the environments they work in, be it compliant and non-compliant environments, it is necessary for robots to know the dynamics of said environments. This is of upmost importance in order for the machines to avoid getting harmed and/or harming their surroundings.

Although it is possible to estimate these dynamics beforehand, for environments that may potentially change overtime or that are non-homogeneous in structure, this might not be the most optimal solution. This creates the necessity for a control algorithm that is capable of estimating these dynamics on the go as efficiently as possible such that the robots can adapt to the work environment.

This thesis investigates physics based modelling approaches and adaptive force trajectory tracking for one-dimensional (1-D) velocity/position controlled robotic manipulators in contact with compliant surfaces. Two different contact models are considered, namely the linear Kelvin-Voight model and the non-linear Hunt-Crossley model, and three different controllers are reported for each of the models. When the manipulator is under exact or asymptotically exact inner loop velocity tracking, the controllers provide asymptotic force trajectory tracking. Finally, the controllers are tested in a compliant environment properties of which is approximated by a neural network, the controllers are evaluated based on their tracking capacities and step responses and experimental results are presented. Since the controllers achieve asymptotic tracking through the estimation of a set of parameters, the convergence of control parameters are reported as well. In the remainder of this section, the literature related to the task at hand are briefly reviewed.

In [14], the existing contact models are reviewed and additional insight into the use of these models in robotic applications is provided. Papers [15] and [17] consider the problem of online estimation of unknown environment dynamics used for robotic contact tasks. The authors propose two new online parameter estimation methods for the nonlinear Hunt-Crossley model. The approaches are numerically evaluated based on their rate of convergence and sensitivity. The reader is referred to [16] for a more thorough examination of said identification methods. Paper [18] reports a recursive least-squares based method for estimating the environment force without the employment of any force sensors for tele-robotic surgery systems. In

[19], a method for online estimation of tissue properties for robot assisted surgery is presented. The method is paired with a real-time graphical overlay to provide the operator with information regarding the tissue.

In [20], the authors, using the standard model-reference adaptive control approach, investigate the role of adaptive controllers for force control with unknown system and environment parameters along with fundamental issues such as explicit force control, impedance control, and impedance control combined with a desired forced control. Concepts such as environment stiffness identification, stability of the complete system, and parameter convergence are analyzed as well. In [1], the problem of achieving exact dynamic force control with position/velocity controlled industrial robotic manipulators is addressed. An adaptive force controller for manipulators in contact with surface of unknown linear compliance is reported. The controller guarantees global asymptotic convergence of force trajectory tracking errors to zero when the robot is under exact or asymptotically exact inner loop velocity control. In [50], the problem of force/position control for a robotic manipulator in compliant contact with a surface 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.

The rest this paper is organized as follows. In Chapter 2, a crash course on neural networks is provided, and the data pre-processing techniques necessary for this thesis work are analyzed. In Chapter 3, the theory behind both force and adaptive control approaches are briefly covered. In Chapter 4, the assumptions made regarding the robot plant and its inner loop position/velocity controller are stated, basics of impact theory are presented along with the dynamical contact models used in this work. In Chapter 5, the adaptive force trajectory tracking controllers are presented and how they were driven are shown. In Chapter 6, the experiments performed with the controllers are explained and the outcomes are analyzed. Finally, in Chapter 7, conclusions regarding the experiments are discussed along with some suggestions for the future.

2 Neural Network Based Function Approximation

In this chapter, the theory behind machine learning and artificial neural networks is briefly discussed and how it apply to this project is explained, and some data pre-processing approaches are provided as well, since it is not healthy to train a network with raw data. Finally, training stages for neural networks are given.

2.1 Machine Learning & Artificial Neural Networks(ANN)

In today's world, artificial intelligence (AI) is a booming research field with a variety of practical applications. Intelligent software can be used for automating routine labor, understanding speech or images, making diagnoses in medicine, and supporting basic scientific research.

In its early days, AI managed to solve problems that were relatively simple for computers yet intellectually difficult for humans. The true challenge, however, was solving problems of the opposite nature – problems that humans solve intuitively without any extra effort such as recognizing speech or faces in images.

The solution is to allow computers learn from experience and describe the world in terms of a hierarchy of concepts where each concept consists of even simpler concepts. By gathering knowledge from experience, the need for human operators to formally specify all the knowledge that the computer needs can be avoided. The hierarchy of concepts allows the computer to learn complicated concepts by building them out of simpler ones. This is referred to as AI deep learning because if this so-called hierarchy of concepts were to be represented by a graph, the graph would be deep, with many layers.

Several AI projects attempted to tackle problems that require human intuition by hard coding the knowledge about the world in formal languages. A computer can reason about statements in these formal languages automatically using logical inference rules. Unfortunately, none of these projects has led to a major success due to the struggle of devising formal rules with enough complexity to accurately describe the world.

These difficulties showed that hard coding on its own was not only inefficient but unsatisfactory as well and that AI systems need the ability to acquire their own knowledge, by extracting patterns from raw data. This is referred to as machine learning. The introduction of machine learning allowed computers to tackle problems involving knowledge of the real world and make decisions that appear subjective [32].

There are three major machine learning approaches[33]:

- **Supervised learning** algorithms can apply what has been learned in the past to new data using labeled examples to predict future events. Starting from the analysis of a known training data-set, the learning algorithm produces an inferred function to make predictions about the output values.
- **Unsupervised learning** algorithms are used when the information used to train is neither classified nor labeled. Unsupervised learning studies how systems can infer a function to describe a hidden structure from unlabeled data.
- **Reinforcement learning** algorithms can interact with their environment by producing actions and deciding whether the action should be rewarded or punished.

2.1.1 The Perceptron

A perceptron is an algorithm for supervised learning of binary classifiers that enables neurons to learn and process elements in the training set. A neural network is an interconnected system of perceptrons, in other words perceptrons function as building blocks for neural networks.

There are two types of perceptron:

- Single-layer perceptron
- Multi-layer perceptron

Single-layer Perceptron

The single-layer perceptron(SLP) is a linear classification algorithm that collects input data in two groups by separating them with a straight line. Input is typically a vector x that stores the features of a data set(For example, for a data set that consists of vehicles, the feature vector could store certain properties of said vehicles such as the type, year of manufacture, number of doors etc.). The output of the perceptron is obtained by multiplying x by a weight vector w and then adding a bias term b:

$$z = w^\top x + b. \tag{2.1}$$

An SLP produces a single output based on several real-valued inputs by forming a linear combination using weights and biases. In some cases, the output is passed through a non-linear function called the activation function:

$$y = a(\sum_{i=1}^{n} w_i x_i + b) = a(w^\top x + b), \tag{2.2}$$

where n corresponds to the number of features of input i, and a is the non-linear activation function. The weight of an input feature indicates the importance of that feature, greater the weight the higher influence the feature has. Tweaking the bias on the other hand, moves the line (2.1) vertically, so that it would be a better representative of the data set.

Multi-layer Perceptron

A multi-layer perceptron (MLP) is a class of feedforward artificial neural network that is often used for supervised learning problems. MLPs learn a function $f(.) : R^m \rightarrow R^o$ by training on a set of input-output pairs, i.e., The algorithm learns the correlation between inputs and outputs, where m and o are dimensions of the input and output vectors respectively. Given a set of input features $X = x_1, x_2, \ldots, x_m$ and a target y, it can learn to approximate a non-linear function either for regression or classification.

MLPs simply are compositions of several single layer perceptrons. MLPs consist of an input layer, an output layer that makes a prediction for the given input and an arbitrary number of hidden layers placed in between the input and output which act as the main computational engine of the algorithm. MLPs with one hidden layer are capable of approximating any continuous function[35].

However, there are some disadvantages to them. These include[34]:

- MLPs are sensitive to feature scaling, i.e., the input features with greater scales tend to have more influence over the prediction of the network. Therefore, before starting the training process all input features must be on the same scale.
- MLPs require tuning a number of hyperparameters such as the number of hidden neurons, layers and iterations.
- MLPs with hidden layers have a non-convex loss function for which there exists several local minima. Therefore, different random weight initializations can affect the validation accuracy.

2.1.2 Loss Function

The loss function is simply a method of evaluating how well your network model fits to the dataset it was trained with. The accuracy of the network's predictions is related to the total loss of the network. While a greater loss value is an indicator of poor accuracy, a lower loss means that the output of network is quite close to the true output[38].

The loss functions can be collected under two main categories: Regression and classification loss. Classification loss, as the name suggests, is used when the task of the network is to distinguish different inputs from one another and place them under their respective classes, e.g., a network that is responsible for telling dogs from cats. On the other hand, regression loss is preferred when the goal is to approximate a function that can describe a given dataset as accurately as possible.

Half Mean Squared Error Loss

One of the objectives of this project is to model the contact dynamics of a compliant object using a neural network in order to create a testing environment. Given the nature of this task, a regression loss was chosen, specifically the half mean squared error (MSE) loss. MSE shows how close the networks predictions (the regression line) to a set of target outputs are by calculating the averaged squared differences between the predictions and true outputs as:

$$\mathbf{MSE} = \frac{1}{2N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 \tag{2.3}$$

where N represents the number of data samples.

MSE is sensitive towards outliers and given several examples with the same input feature values, the optimal prediction will be their mean target value [39].

2.1.3 Activation Function

Activation functions are critical to the design of a neural network. They define how the weighted sum of the input is transformed into an output from a neuron or neurons in a layer of the network. While the choice of activation function in the hidden layer affects the overall training performance, in the output layer it defines the type of the network.

Differentiability is a must have property of activation functions given that neural networks are typically trained by calculating the gradients of the loss function with respect to the network parameters then using those gradients to update the parameters[37]. These two steps are referred to as back propagation and optimization respectively. These topics will be covered more extensively in the upcoming sections.

There are two main classes of activation functions, namely linear and non-linear activation functions. A good deal of activation functions belongs to the latter category, and their purpose is to introduce nonlinearity to the network. A neural network without any nonlinear activation functions is nothing more than a linear regressor and is incapable of approximating complex functions regardless of the number of its hidden layers, since the composition of linear functions is still a linear function.

For this project, two different activation functions were tested: Hyperbolic tangent(TanH) and softsign activation functions.

TanH and Softsign Activation Functions

TanH(2.4) and Softsign(2.5) functions are quite similar to one another with one major difference: While TanH converges exponentially, Softsign converges polynomially[40].

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (2.4) \qquad g(x) = \frac{x}{1+|x|} \quad (2.5)$$

A problem with TanH is that it saturates the neurons meaning that large values snap to 1 and small values snap to -1. It becomes more challenging for the optimization algorithm to update the parameters to improve the model performance. At the same time, TanH function is not very sensitive to changes away from its mid-point which is the origin[41].

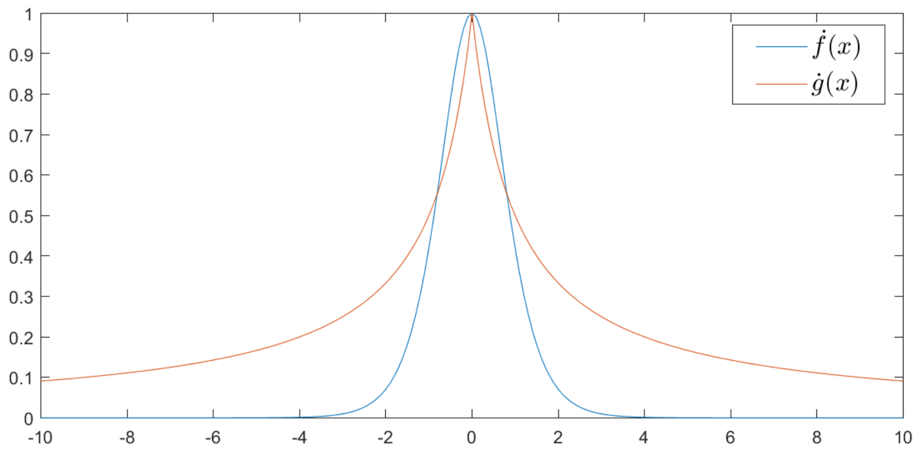


Figure 2.1: Comparison of the derivatives of TanH and Softsign functions

These cases apply to Softsign function to certain degree as well, but as can be seen in figure 2.1, even though it is not as sensitive to changes close to its mid-point, it is more forgiving towards inputs with greater magnitudes reducing the chance of saturation.

2.2 Pre-processing

The quality of the input data is of upmost importance when it comes to training neural networks, since the accuracy of the trained model depends on the input data. Unfortunately, clean and formatted data is hard to come by, and raw data is typically noisy, has missing values and perhaps in a format that cannot be used directly for training. Another issue arises when dealing with networks that take multiple input features: There could be a difference in scale between the features. In this case, the network puts more emphasis on the feature with the greater value during training, as it is unaware of the unit and type of the data it processes.

In order to overcome these issues, the data has to be processed before it is passed to the network as input. This technique is referred to as pre-processing and is a necessity when it comes to creating a neural network model.

The remainder of this section is dedicated to the presentation of the pre-processing methods used in this thesis for the preparation of the training data.

2.2.1 Filtering

Data filtering is the task of clearing the measured process data of noise (or errors) as much as possible. It is a vital task because measurement noise masks the important features in the data limiting its usefulness in practice.

In order to train the network to be used as a testing environment, force and position data was collected. However, due to the high frequency noise content of the data, it was not feasible to train the network without pushing the data through a low-pass filter first.

A low-pass filter, as the name suggests, only allows components of signals with frequencies lower than a certain threshold to pass. This threshold is referred to as cutoff frequency and it is determined by the user depending on the application. Order of a filter is another design parameter and it is the measure of a filter's complexity. The order affects the shape or width of the roll-off also known as the "transition band" (lower order filters have wider transition bands). In applications that use filters to shape the frequency spectrum of a signal such as in control systems, higher order filters are usually required as the transition band of a simple first order filter may be too wide to be eligible[21].

Butterworth Low-pass Filter

For the purpose of clearing the data of its high frequencies, a low-pass Butterworth filter was picked. The pass band of this filter is designed to have a frequency response which as flat as mathematically possible from 0Hz until the cutoff frequency with no ripples, meaning that the low frequency components of the input signal are preserved as much as possible. However, this property of the filter comes at a price: A wide transition band as the filter changes from the pass band to the stop band.

A Butterworth filter is normally used due to its adequate frequency response. In online applications, it is necessary to have a good trade-off between the transient response and the required attenuation characteristic. As the order of a filter increases so does its filtering performance at the expense of a longer settling time. A second order Butterworth filter manages to hit this sweet spot between the performance and the settling time[21].

2.2.2 Feature Scaling - Standardization

Feature scaling is a crucial step during the pre-processing phase and a requirement for the network to output healthy predictions. A machine learning model sees data as nothing but a bunch of numbers and simply attempts to form ties between the input features and the target output based on those numbers. If, however, there is a significant difference in scale between the input features, the features with greater magnitudes will impact the prediction of the network more remarkably. This will cause the machine learning to get insensitive towards the input features with lesser magnitudes regardless of how important they really are.

In order to overcome these issues, the input data has to be adjusted such that all its features share a common scale. The two most common ways of doing this are standardization and normalization. Normalization, also known as min/max scaling, scales and translates each and every feature to a given range. Standardization, on the other hand, transforms the features such that they have a mean of zero and a standard deviation of one. Another difference between the two is while standardization responds better when the data is normally distributed, normalization does not.

In general, the shape of the distribution is ignored in practice. The data is just transformed such that it is centered around the origin by removing the mean value of each feature, then scaled by dividing the features by their standard deviation as in equation (2.6)

$$z = \frac{x - u}{s} \tag{2.6}$$

where u is the mean and s is the standard deviation.

Machine learning estimators tend to behave nicer when the data looks more or less like it is normally distributed. That's why in this project standardization was picked as the feature scaling method.

2.3 Training ANNs

In supervised learning, the training of a neural network is composed of three main steps: Forward propagation, backward propagation and parameter optimization. During training, the network cycles between the three stages repeatedly while iterating over the training data in order to minimize the error between the network's prediction and the target/true output. Each of these training cycles is referred to as an epoch, and the amount of data instances processed at a time by the network is determined by a quantity called the batch size. Both the number of epochs and size of a training batch are design parameters also known as hyperparameters.

2.3.1 Forward Propagation(ForwardProp)

During forward propagation, the input data is fed to the network in order from the input layer to the output layer. Each hidden layer receives the input from its pre-

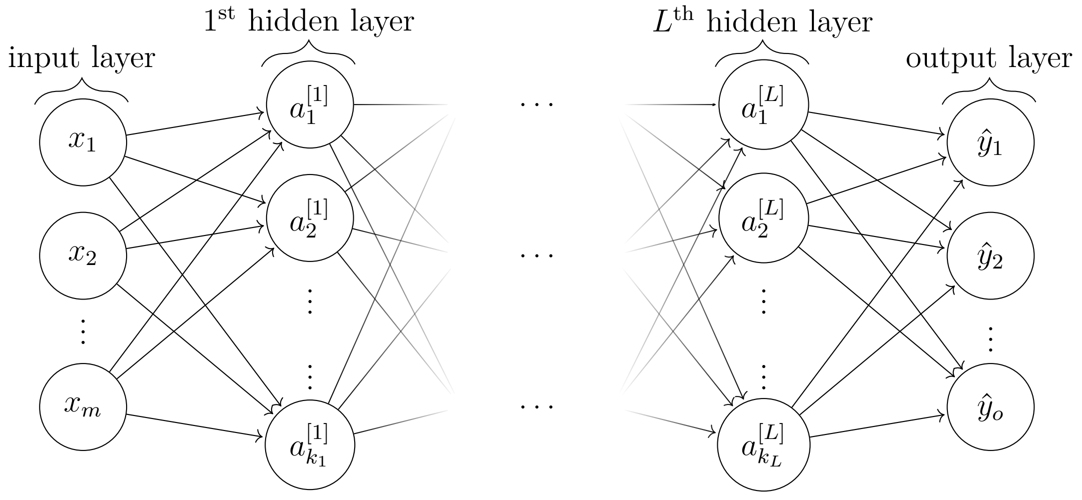


Figure 2.2: Network graph of a $(L + 1)$-layer perceptron with m inputs and o outputs. The L^{th} hidden layer contains k_L neurons.

decessor, processes it, and then passes it to the successive layer. In the meantime, intermediate values(activations) as well as the outputs are calculated and stored[36].

In figure 2.2, $a^{[L]}$ corresponds to the activation vector of all neurons of the L^{th} layer. Before the activation can be calculated, pre-activation $z^{[L]}$ has to be calculated according to equation (2.1) as follows:

$$z^{[L]} = W^{[L]^\top} a^{[L-1]} + b^{[L]}, \tag{2.7}$$

where $(W^{[L]})^\top$ and $b^{[L]}$ are the weight matrix and the bias vector of layer L respectively (Sizes of the weight matrix and the bias vector depend on the number of outputs of layer $L - 1$ and the number of outputs of layer L).

By plugging equation (2.7) into the activation function $f_L(.)$ of layer L

$$a^{[L]} = f_L(z^{[L]}), \tag{2.8}$$

activation vector for L^{th} layer $a^{[L]}$ is obtained. As stated earlier, operations (2.7) and (2.8) are repeated for every single layer, till the network output $\hat{y}$ is calculated. Then, the total loss J is obtained by plugging the ground truth y and the prediction $\hat{y}$ into the loss function $E(.)$ as:

$$J = E(y, \hat{y}). \tag{2.9}$$

2.3.2 Backward Propagation(BackProp)

The back propagation algorithm, by using the chain rule of calculus, calculates the partial derivatives of the loss function with respect to the network parameters starting from the output layer and moving towards the input layer. These partial derivatives reflect how the loss function changes with respect to each and every parameter. By using a gradient descent based optimization algorithm, e.g., stochastic gradient descent, batch gradient descent etc., the parameters can be adjusted in the direction for which the total loss is minimized.

Now, in order to clarify the math behind BackProp, an example[36] will be provided where the network is trained on a dataset $[x_i|y_i]$ with input features x_i and their target outputs y_i. For the sake of simplicity, it is assumed that the network in figure 2.2 has a single hidden layer(with arbitrary amount of neurons), and the bias terms are neglected.

By following the ForwardProp algorithm, the network output $\hat{y}$ and loss J are calculated as follows:

$$a^{[1]} = f_1(z^{[1]}) \text{ where } z^{[1]} = (W^{[1]})^\top x, \tag{2.10a}$$

$$\hat{y} = W^{[2]^\top} a^{[1]}, \tag{2.10b}$$

$$J = E(y, \hat{y}), \tag{2.10c}$$

where $W^{[1]}$ and $W^{[2]}$ are the weight matrices of the hidden layer and the output layer respectively.

Next, to find the gradient of the loss function with respect to $W^{[2]}$, $\partial J/\partial \hat{y}$ and $\partial \hat{y}/\partial W^{[2]}$ has to be computed. Then, the gradient can be found as:

$$\frac{\partial J}{\partial W^{[2]}} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial W^{[2]}} = \frac{\partial J}{\partial \hat{y}} a^{[1]^\top}. \tag{2.11}$$

To obtain the gradient of the loss function with respect to $W^{[1]}$, the back propagation needs to continue along the output layer to the hidden layer. The gradient with respect to the activations of the hidden layer $\partial J/\partial a^{[1]}$ is given by:

$$\frac{\partial J}{\partial a^{[1]}} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a^{[1]}} = W^{[2]^\top} \frac{\partial J}{\partial \hat{y}}. \tag{2.12}$$

The activation function f_1 applies element-wise so, for calculating the gradient of the loss function with respect to $z^{[1]}$, element wise multiplication operator $\odot$ has to be used:

$$\frac{\partial J}{\partial z^{[1]}} = \frac{\partial J}{\partial a^{[1]}} \frac{\partial a^{[1]}}{\partial z^{[1]}} = \frac{\partial J}{\partial a^{[1]}} \odot f_1'(z^{[1]}). \tag{2.13}$$

Finally, the gradient of the loss function with respect to $W^{[1]}$ can be calculated as follows:

$$\frac{\partial J}{\partial W^{[1]}} = \frac{\partial J}{\partial z^{[1]}} \frac{\partial z^{[1]}}{\partial W^{[1]}} = \frac{\partial J}{\partial z^{[1]}} x^\top \tag{2.14}$$

Now that the gradients are calculated, the next step is to optimize the network parameters.

2.3.3 Parameter Optimization

Earlier, it was stated that the loss function is a measure of the networks performance. The goal of training is to minimize the loss as much as possible. In order to do so, an optimization algorithm needs to update the model parameters, i.e., weights and biases, using the gradients calculated with respect to the loss function.

Gradient Descent

Gradient descent is the most basic optimization algorithm used to find the parameter values that minimize the loss. It updates the parameters in the opposite direction of their respective gradients to gradually reduce the loss as follows:

$$\hat{y} = \mathrm{ANN}_\theta(x) \quad \text{(Make a prediction)} \tag{2.15a}$$

$$\theta = \theta - \alpha \frac{\partial J}{\partial \theta} \quad \text{(Update the parameters)} \tag{2.15b}$$

Where $\hat{y}$ is the prediction of the network given input x, θ denotes the parameters, $\frac{\partial J}{\partial \theta}$ is the gradient of the loss J with respect to the parameters, and α is the learning rate.

To be able use gradient descent, values for hyperparameters learning rate and batch size has to be chosen. It is important to select appropriate values for these as they affect the optimization process. This, however, is not so straightforward.

If the learning rate is picked too small, then the parameters will take too long to converge to their optimal values, and if it is too large, the parameters may end up fluctuating around their optimal values and even diverge from them. The learning rate can be adjusted according to a pre-defined schedule [43], but the schedule has to be defined in advance thus fails to adapt to a dataset's characteristics [42]. Additionally, the same learning rate is used to update all parameters, and depending on the sparsity of the training data and frequencies of the features, it may not be logical to update all parameters to the same extent[44].

Batch size determines the frequency of updates. The parameters are updated more quickly for smaller batches, while larger batches increase the accuracy of the gradient of the loss function with respect to the parameters[45].

Adaptive Moment Estimation(Adam)

In this thesis, Adam was chosen as the optimizer. The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. It combines advantages of both AdaGrad[48] and RMSProp[49], i.e., it works well in online and non-stationary settings and with sparse gradients.

Some advantages of Adam are as follows[47]:

- Rescaling of the gradient does not affect the magnitude of the parameter updates
- Its step sizes are approximately bounded by the step size hyperparameter.
- It naturally performs a form of step size annealing. In other words, the algorithm adjusts its step size such that the global optimum can be found without visiting every possible solution. This also prevents Adam from getting stuck at local optima.

The Adam algorithm can be found below [47]:

Algorithm 1 Adam Algorithm

```
Require: $\alpha$ : Learning rate
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $J(\theta)$: Objective/Loss function with parameters $\theta$
Require: $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize first moment vector)
  $v_0 \leftarrow 0$ (Initialize second moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while $\theta_t$ not converged do
      $t \leftarrow t + 1$
      $g_t \leftarrow \nabla_\theta J_t(\theta_{t-1})$ (Get gradients w.r.t the loss function at timestep $t$)
      $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$ (Update biased first moment estimate)
      $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) {g_t}^2$ (Update biased second moment estimate)
      $\hat{m}_t \leftarrow m_t/(1 - \beta_1)$ (Compute bias-corrected first moment estimate)
      $\hat{v}_t \leftarrow v_t/(1 - \beta_2)$ (Compute bias-corrected second moment estimate)
      $\theta_t \leftarrow \theta_{t-1} - \alpha \hat{m}_t/(\sqrt{\hat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return $\theta_t$ (Updated parameters)
```

3
Control System Design

In this chapter, the reader is provided with the background on control methods that were used in this thesis. Firstly, a brief background on force control is provided and its role in robotic applications is discussed. Then, both direct and indirect adaptive control methods are explained and a short tutorial on how to design direct adaptive controllers is given.

3.1 Force Control

In order for a robotic manipulator to successfully execute practical tasks where, for example, the robot end-effector has to manipulate an object or perform some operation on a surface, controlling the interaction between the robot and the environment is a necessity. These tasks include polishing, deburring, machining or assembly.

The environment may constrain the motion of the manipulator during interaction, limiting the geometric paths the end-effector can follow. This is referred to as constrained motion. Using a pure motion controller in this case in order to control interaction is likely to fail unless the task is accurately planned. However, this would require an accurate model of both the robot and the environment and even though the manipulator can be modeled with satisfactory precision, an accurate description of the environment is hard to obtain.

During execution, inaccurately planned tasks may give rise to a contact force due to which the end-effector deviates from its desired trajectory and the controller reacts to reduce this deviation. This ultimately leads to a build-up of contact force until the parts that are in contact with the environment break. Scenarios such as this tend occur more frequently as the environment stiffness and position control accuracy increase. This drawback can be overcome if a compliant behavior is ensured during the interaction[11]. Here, the term compliance describes the relation between the motion and forces generated by a manipulator and the environment the robot works in during contact. This compliant behavior can be achieved either actively or passively. If the desired joint stiffness is enforced by a controller, this is called active compliance. On the other hand, passive compliance occurs as a result of the flexibility inherent in the mechanical linkage due to the limited stiffness of the robot links, drive system, gripper as well as the environment[12].

The contact force describes the state of interaction in the most complete fashion.

If the force measurements are available to the controller then, the performance can be improved with interaction control. The force readings can be obtained by a force/torque sensor attached to the manipulator, typically between the wrist and the end-effector.

Interaction control strategies can be grouped under two categories, namely, indirect force control and direct force control. The indirect methods achieve force control via motion control, while the direct methods offer the possibility of controlling the contact force to a desired value through the closure of a force feedback loop around the inner velocity control loop[11].

In the remainder of this section, both indirect and direct methods will be briefly discussed. The controllers designed for this project, although they are pure force controllers(since only the 1-D case was considered), can be modified to fit under both categories.

3.1.1 Indirect Force Control

Admittance and impedance control schemes belong under this category, where the deviation of the end-effector from the desired trajectory due to the environment is related to the contact force through a mechanical admittance/impedance with adjustable parameters. A mass-spring-damper system with adjustable parameters can be used to realize a robot manipulator under either admittance or impedance control. If the controller reacts by deviating the end-effector from its desired trajectory in response to interaction forces, this is referred to as admittance. On the other hand, if the controller generates forces to react to the changes in trajectory, this is called impedance. By these definitions, it can be seen that admittance control is the opposite, or dual of impedance control. Special cases of admittance and impedance control are compliance control and stiffness control, respectively. These two controllers consider only the static relationship between the deviation of the end-effector from its desired path and the contact force.

The impedance control computes the velocity input on the basis of position feedback as well as the force measurements. Under impedance control, asymptotical position trajectory tracking can be achieved, if there are no forces acting on the manipulator. In the presence of contact with the environment, a compliant dynamic behavior is imposed on the end-effector.

Due to the dynamics of the closed loop system being different in free space and during interaction, the selection of good impedance parameters is not easy. The control objectives for free space and interaction are different as well. In free space, the main focuses are motion tracking and disturbance rejection, contrarily, during interaction, the goal is to achieve a suitable compliant dynamic behavior for the end-effector. Another thing to keep in mind is that the dynamics of the closed loop system depends on the dynamics of the environment during contact.

During interaction, the contact force can be made small at the expense of a large position error in steady state, as long as the active stiffness(the measure of stiffness for the manipulator) is set low with respect to the stiffness of the environment, and vice versa. However, both the contact force and the position error also depend on the external disturbances; in particular, the lower the active stiffness, the higher the influence of disturbances. Moreover, a low active stiffness may result in a large position error also in the absence of interaction.

This drawback can be resolved by separating motion control from impedance control. To enhance disturbance rejection, the motion control action can be made stiff on purpose. This ensures tracking of a reference position trajectory resulting from the impedance control action rather than the tracking of the desired end-effector position. In other words, the desired position together with the measured contact force are inputs to the impedance equation which, via a suitable integration, generates the reference position trajectory for the motion control. This is referred to as admittance control.

In principle, measurement of the contact forces is not required by indirect control schemes. However, this results in an impedance or an admittance that is typically nonlinear and coupled. A linear and decoupled admittance/impedance can be achieved by using force measurements - that is, a force/torque sensor is installed on the manipulator[13].

3.1.2 Direct Force Control

With an indirect control approach, it is possible to limit the values of the contact force to a certain interval by suitably controlling the end-effector motion, if at least a rough estimate of the environment stiffness is available. However, certain interaction tasks require the precise value of the contact force which would be possible to obtain by tuning the active compliance control action and selecting a proper desired location for the end-effector given that an accurate estimation of the environment stiffness is at hand.

In order to bypass this requirement, a direct force control which operates on the error between the desired and the measured force values can be adopted. In the previous section, it was explained that even impedance and admittance control schemes require force measurements to obtain linear and decoupled contact dynamics during interaction even though these controllers do not aim to achieve force tracking. This is useful, as a compliant behavior can be realized specifically for the task directions that are constrained by the environment.

The direct force control scheme can be implemented by closing an outer force control loop that generates a desired velocity trajectory for the inner velocity control loop of the manipulator. However, with direct force control there is a trade off between force control and motion control. By sacrificing the control of the end-effector motion, contact force can be regulated. If a detailed description of the environment geometry is available, the motion control can be recovered along the directions that are not constrained by the environment while ensuring force control along the constrained task directions[11].

3.2 Adaptive Control

The research on adaptive control has its roots back to early 1950s. One of the primary motivators of the research was designing autopilots for high performance aircraft that is capable of operating over a wide range of speeds and altitudes[2].The dynamics of these vehicles are susceptible to severe changes and therefore the ordinary constant-gain feedback controller that could only work in one operating condition was not enough. Adaptive control was then proposed as a way of learning these changes in the dynamics and adjusting the controller parameters automatically[3].

Adaptive control methods can be utilized to achieve or to maintain a desired level of control performance by automatically adjusting the controllers in real time when dealing with dynamical systems with plant parameters that are unknown and/or can change over time[8]. For instance, in some control tasks the systems to be controlled have parameter uncertainty at the beginning of the operation, and unless this uncertainty is dealt with on-line in an adaptive fashion, it may result either in inaccurate or unstable control systems. In another case, some systems may have well known dynamics at the beginning, yet suffer from unpredictable parameter variations over time meaning that the initially appropriate controller may fail to meet the performance criteria if it is not continuously redesigned[4].

Due to the fact that parameter uncertainties and variations are almost unavoidable, adaptive control has a variety of industrial applications such as

- Autopilots for ships, and aircraft
- Industrial robots
- Power systems
- Chemical reactors etc.

3.2.1 What is Adaptive Control?

Throughout the history, even though there have been several attempts to define what an adaptive control system is, these efforts were not widely accepted. A meaningful definition, which would make it possible to tell whether a system is adaptive by looking at the controller hardware and software, is still not available[3]. Ioannou and Fidan[5] simply define adaptive control as the combination of an online parameter estimator with a control law in order to control classes of plants whose parameters are completely unknown and/or could change unpredictably.

Closed loop adaptive control approaches can be classified into two broad categories, namely, direct adaptive control and indirect adaptive control[6]. In direct control, the controller is parameterized in terms of the plant parameters and these parameters can be estimated directly hence, no explicit plant identification is necessary. Whereas, indirect control methods separate the parameter identification from the adjustment of the controller. The unknown plant parameters have to be estimated first then used to adjust the controller parameters[7]. However, these strategies cannot be utilized when dealing with nonlinear plants. In this case, a linear in the weights neural network can be used to describe the nonlinearity, then an intuition similar to that of the linear case can be adopted to estimate the unknown parameters. This is referred to as neuro-adaptive control. These concepts will be explained briefly in this text with more emphasis placed on the direct and neuro-adaptive approaches as the control design is based on these strategies.

Direct Adaptive Control

The design and tuning of a controller goes through the specification of the desired closed loop performance, and generally, the characteristics of a dynamic system can be used to describe this desired behaviour, e.g., for a tracking objective, the rise time ,and overshoot under a step input can be used to specify the input/output behaviour of a transfer function which in return can be used to describe the system itself[8].

In direct adaptive control, the controller is designed to guarantee that the closed loop behaviour matches the desired behaviour of the system plant which is described by a so-called reference model. However, there are also cases where direct adaptive controllers are designed without the help of a reference model but instead a reference trajectory. In order for tracking convergence to be possible, the controller should have perfect tracking capacity when the plant parameters are exactly known, i.e., the output of the reference model (or the reference trajectory) should be identical to the output of the plant. When these parameters are unknown or change over time, an online adjustment mechanism has to search for these values on the basis of the tracking error such that the perfect tracking can be achieved asymptotically[3][4].

One short coming of this approach is that the direct adaptive schemes are restricted to minimum-phase plants, i.e., plant models with zeros strictly located on the left hand side of the complex plane[2]. The direct adaptive control structure can be seen in figure 3.1.

Indirect Adaptive Control

In a non-adaptive scenario, in which the plant parameters are known, the controller parameters are driven based on those of the plant. When dealing with unknown plant parameters however, these values can be replaced by their estimates that are obtained with an online parameter estimator. This underlies the basic idea behind indirect adaptive control(see figure 3.2), given some input/output measurements a plant model can be estimated online, which then can be used to design a suitable controller[8]. As the previous statement suggests, the adaptation of the controller

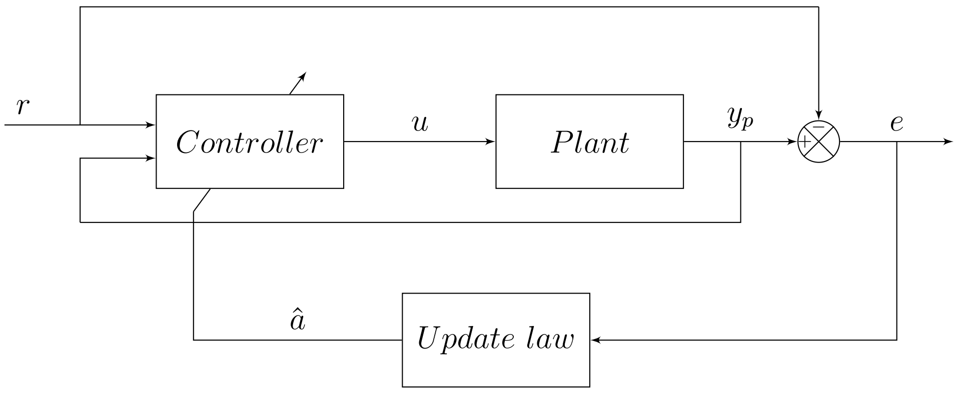


Figure 3.1: Block diagram of a direct adaptive control scheme

parameters is handled in two successive steps, an online plant identification followed by an online control parameter computation which is the reason why this scheme is called indirect. These two steps are repeated at each time instant until a set of parameters that fits the available input/output measurements from the plant are found[4].

This approach treats the estimated plant parameters as if they were the true parameters in order to calculate the controller parameters. This is called the certainty equivalence principle[3]. This allows combining different online estimators with different controllers in order to obtain a vast variety of adaptive schemes. Indirect adaptive controllers are applicable to both minimum and non-minimum phase systems yet, due to the parameter estimates being passed as true parameter values when calculating the controller, it is not guaranteed that these schemes will always work[2][8].

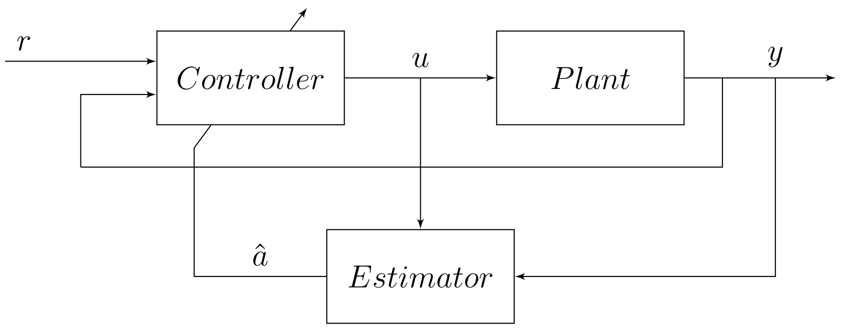


Figure 3.2: Block diagram of an indirect adaptive control scheme

Neuro-adaptive Control

A natural extension of the adaptive control results for linear time invariant (LTI) and linear time varying (LTV) plants is their use to control nonlinear systems. Nonlinear systems belong to a wider class of systems which also includes the LTI and LTV plants.

In most nonlinear systems source of the uncertainties are not only the parameters but the unknown nonlinear functions as well. There are two major strategies available for designing adaptive controllers for nonlinear systems[5]. The first method requires the nonlinear system to satisfy the following conditions[4]:

1. the nonlinear plant dynamics can be linearly parameterized
2. the full state is measurable
3. nonlinearities can be canceled stably (i.e., without unstable hidden modes or dynamics) by the control input if the parameters are known

The second method, which is known as neuro-adaptive control, approximates the unknown nonlinear functions by the multiplication of some unknown with known basis functions. These basis functions include those used in neural networks, and the approach followed is similar to that in the neural network area. The only difference is that the unknown parameters or weights, as they are referred to in the neural network literature, appear linearly or in a single layer[5].

3.2.2 How to Design Direct Adaptive Controllers

In adaptive control, the parameters of the plant are unknown, therefore the parameters of the chosen control law have to be continually estimated by an adaptive update law unlike in conventional control design, where the structure of the controller is picked first then the control parameters are calculated based on the known plant parameters.

In the remainder of this section steps involved in control design will be explained. Before that however, a useful lemma[4] has to be provided as it is the backbone of the adaptation law design.

Lemma 1. *Consider two signals e and Φ related by the following dynamic equation*

$$e(t) = H(s)[k\Phi^\top(t)v(t)] \tag{3.1}$$

where $e(t)$ is a scalar output signal, $H(s)$ is a strictly positive real(SPR) transfer function, k is an unknown constant with known sign, $\Phi(t)$ is a m by 1 vector function of time, and $v(t)$ is a measurable m by 1 vector. If the vector $\Phi(t)$ varies according to

$$\dot{\Phi}(t) = -sgn(k)\gamma e v(t) \tag{3.2}$$

with γ being a positive constant, then $e(t)$ and $\Phi(t)$ are globally bounded. Furthermore, if $v(t)$ is bounded, then

$$e(t) \to 0 \; as \; t \to \infty \tag{3.3}$$

Keep in mind that the equation (3.1), even though it contains both time and frequency domain notations, it simply means that $e(t)$ is the response of the linear system with the transfer function $H(s)$ to the input $[k\Phi^T(t)v(t)]$. In adaptive control, these hybrid notations are common and quite useful as they allow for skipping the step where the intermediate variables are defined.

Control Design Steps

Consider the following first order system with unknown but constant parameters a_p and b_p

$$\dot{y} = f(t; y, u) = -a_p + b_y u \tag{3.4}$$

where y is the output and u is the input. This system can be used to describe for instance, the braking of a car, or the flow of fluid from a tank.

1. Problem Specification

The parameters of the control law will be based on the unknown plant parameters. The adaptive update law will adjust these parameters based on the tracking error between the outputs of the system plant and the ideal system response which is described by a bounded reference trajectory, $r(t)$.

2. Choice of the Control Law

The controller has to be chosen such that if the plant parameters were exactly known, the closed loop dynamics of the system would yield zero tracking error.

Suppose that the following control law

$$u = g(t; r, y; \hat{a}_r, \hat{a}_y) \tag{3.5}$$

where $\hat{a}_r(t)$ and $\hat{a}_y(t)$ are variable feedback gains drives the tracking error to zero over time when plugged into (3.4).

Since the plant parameters are unknown, the control input will achieve these through an adaptation law that will continuously search for the right gains, based on the tracking error, until perfect tracking is achieved.

3. Choice of Adaptation Law

Before deriving the adaptation law the error dynamics function has to be defined with respect to the control parameter errors, $\tilde{a}_y$ and $\tilde{a}_r$ as such

$$\dot{e} = h_1(t; y; a_p, b_p; \tilde{a}_y, \tilde{a}_r) \tag{3.6}$$

where e is the tracking error.

Earlier, it was stated that hybrid conventions were common in adaptive control design. So, equation (3.6) can be manipulated in Laplace domain in order to obtain the following tracking error equation

$$e = N(s)h_2(t; y, r; \tilde{a}_y, \tilde{a}_r) \tag{3.7}$$

then using Lemma 1 and the relation (3.7) between the tracking and the parameter errors the adaptation laws can be written as

$$\dot{\hat{a}}_r = -sgn(b_p)\gamma_r er \tag{3.8a}$$

$$\dot{\hat{a}}_y = -sgn(b_p)\gamma_y ey \tag{3.8b}$$

where $\gamma_r, \gamma_y > 0$ are the adaptation gains.

4

Modelling of Contact Dynamics and Description of the Control System

In this chapter, assumptions regarding the robot plant and inner loop control are established, and the theory behind the contact dynamics modeling of compliant objects is briefly covered. Finally, the basics of impact theory are provided, and different contact modeling approaches used in this work are discussed. The purpose of this chapter is to form the foundation necessary for control design.

4.1 System Description: Force Control with 1-D Position/Velocity Control

4.1.1 Plant Assumptions

1) The plant has two states which are position $x(t)$ and velocity $\dot{x}(t)$, both of which are continuous. The only input to the plant is the desired velocity trajectory $\dot{x}_d(t)$, and the plant outputs are the actual position $x(t)$ and velocity $\dot{x}(t)$. The velocity tracking error then can be written as follows:

$$\Delta\dot{x}(t) = \dot{x}(t) - \dot{x}_d(t) \tag{4.1}$$

2) The robot is in soft contact with the environment at all times.
3) The outer loop force controller has access to the robot position and end point force measurements.

4.1.2 Inner Loop Velocity Control Assumptions

The dynamical model of the manipulator is not available to the user. Therefore, the end-effector force cannot be controlled directly by commanding the joint torques. By closing a force control loop around the inner velocity/position control loop, however, the reference trajectory of the inner loop controller can be modified to control the end-effector force based on the sensed force error. This is referred to as implicit force control in literature[1]. Since the manipulator's model is unavailable, the robot plant can be picked to be a free mass as:

$$u = m\ddot{x} \tag{4.2}$$

where m corresponds to the mass of the manipulator. Under the following velocity controller:

$$u = m\ddot{x}_d(t) - k_p \Delta\dot{x} \tag{4.3}$$

the closed-loop dynamics become

$$m\Delta\ddot{x} + k_p \Delta\dot{x} = 0. \tag{4.4}$$

For all $\mathcal{C}^1$ desired velocity trajectories $\dot{x}_d(t)$, whether they be bounded or unbounded, the closed loop system results in a bounded velocity error $\Delta\dot{x}(t)$ that satisfies:

$$\lim_{t\to\infty} \Delta\dot{x}(t) = 0 \tag{4.5}$$

4.2 Contact Model Dynamics

In order to design adaptive force controllers, it is necessary to have a description of the environment properties, as the dynamics model of the environment directly affects how the controller is structured.

4.2.1 Basic Impact Theory

Impact is a complex physical phenomenon, which occurs when two or more bodies undergo collision with one another [23]. When objects impact the duration of the contact tends to be very brief, and the contact occurs with a high force output. The bodies that took part may end up malformed due to the impact[24]. Contact on the hand, although it is frequently used interchangeably with impact, inherently implies a continuous process which takes place over a finite time[14].

There are two separate approaches when it comes to dealing with contact and impact analysis, namely continuous analysis, or force-based methods and discrete, or impulse-momentum methods. The dynamic analysis of a mechanical system is done in two main parts, before and after the impact occurs, and these two intervals are complimented by secondary phases such as slipping, sticking and reverse motion. To model the process of energy transfer and dissipation, various coefficients are employed, mainly the coefficient of restitution and the impulse ratio[25] [26]. The first approach, continuous analysis, is more naturally suited for contact modeling as it is based on the fact that the interaction forces act in a continuous manner during the impact which describes the true dynamical behavior of the bodies more effectively. The applications of the second approach, however, are confined primarily to impact between rigid bodies. It assumes that the interaction between the objects occurs in a short time and that the configuration of impacting bodies does not change significantly. Extension of this method to flexible bodies is quite complicated unlike the continuous approach [27] .

Impact of two bodies (see Fig. 4.1) is characterized by large reaction forces and changes in velocities of the two bodies. As a consequence, the bodies are subject to elastic and/or plastic deformation, with dissipation of energy in various forms[28].

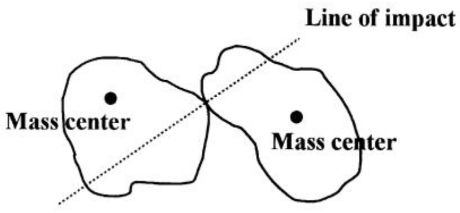


Figure 4.1: Impact between two bodies[14]

The straight line that is normal to the tangential plane at the contact point is called the line of impact[23]. Depending on where the mass centers of the bodies are located with respect to the line of impact and whether their velocity vectors are aligned with it, impact can be classified into four types[14].

- Central: Mass centers are located on the line of impact
- Eccentric: At least one mass center is off the line of impact
- Direct: The initial velocities of the bodies are aligned with the line of impact
- Oblique: At least one initial velocity vector is not aligned with the line of impact

There are also tangential impacts where the initial relative velocity of the bodies is perpendicular to the line of impact.

The properties of contacting bodies such as the material, geometry and velocity all influence the dynamics of impact. In general, there are two phases of impact: compression and restitution. The compression phase begins as soon as the bodies contact and lasts until the maximum deformation is reached. The restitution begins at the point of max deformation and lasts until the bodies are separated. If the impact happened fast enough it is not guaranteed for the deformed bodies to recover back to their initial conditions because of the permanent(plastic) deformation and the resulting energy loss[14].

The objective of impact modeling is to determine the after-impact conditions of the system, given its initial (pre-impact) configuration. As there are many parameters that influence an impact, one possible solution is to use experimentally measured coefficients such as the coefficient of restitution, defined along normal direction, and coefficients along tangential directions [25] [26].

The energy loss due to the motion along the line of impact can be expressed by the coefficient of restitution which is usually denoted by e. The coefficient has to satisfy the condition $0 \leq e \leq 1$ where $e = 0$ corresponds to perfectly plastic and $e = 1$ to perfectly elastic impact. The coefficient of restitution depends on many elements, such as the geometry of the bodies in contact, the approach velocity, the material properties, the duration of contact and, possibly, friction[28].

For the sake of simplicity, friction was not considered in this work and therefore a detailed explanation of the remaining concepts will not be provided here.

4.2.2 Discrete Contact Dynamics Models

This contact model is mainly used for describing the impact involving rigid or at least very hard and compact bodies. The effects of deformation at the contact point are taken into account through coefficients. This formulation assumes that the impact is instantaneous and impact forces are impulsive; kinetic variables have discontinuous changes while no displacements occur during the impact. The impact problem is solved using the impulse-momentum principle (discrete methods) and the relations between the variables before and after impact[25][26].

When it comes to impacts involving multiple bodies and multiple contact points discrete methods are at a disadvantage. Thankfully, within the context of this project this was not the case. Another issue with discrete models is their use of Coulomb's law to model friction during impact. Wang and Kumar[29] have noted that inconsistencies arise when Coulomb's law of friction is used with rigid bodies.

4.2.3 Continuous Contact Dynamics Models

The continuous model, also referred to as compliant contact model, overcomes the problems associated with the discrete models. The basis of the continuous formulation for contact dynamics is to explicitly account for the deformation of the bodies during impact or contact which is done by defining the normal contact force f_c as a function of local indentation x and its rate $\dot{x}$ as follows[24][25]:

$$f_c(x, \dot{x}) = f_{\dot{x}}(\dot{x}) + f_x(x) \tag{4.6}$$

Next, two of the existing contact models will be briefly explained: The linear Kelvin-Voigt model and the nonlinear Hunt-Crossley model.

Kelvin – Voigt model(Spring – dashpot model)

This model consists of a linear damper and a spring connected in parallel with each other. While the spring represents the elastic behavior of the body, the damper is responsible for energy dissipation[28].

The contact force is defined as:

$$f_c = k(x - x_e) + b\dot{x} \quad \text{where} \quad k,\ b > 0 \quad x_e \geq 0 \tag{4.7}$$

where the parameters k and b represent the spring and damping constants respectively, and x_e is the undeformed environment position where the contact force is equal to zero.
This model has three weaknesses[31]:

- It is not realistic enough due to the contact force being discontinuous at the beginning of the impact. This is caused by the linear damping term. Under normal circumstances, one would expect both elastic and damping forces to be initially zero.
- As the bodies are separating, i.e., the indentation tends to zero, their relative velocity tends to be negative resulting in a negative contact force.
- The equivalent coefficient of restitution defined for this model does not depend on impact velocity.

Even though the model is not consistent with the concept of restitution, it still provides a reasonable approach for capturing the energy dissipation associated with the normal forces without explicitly considering plastic deformation issues.

Hunt – Crossley model(Nonlinear damping model)

Hunt and Crossley[30] introduced this model as an alternative to that of Kelvin-Voigt. It includes a nonlinear damping term in order to address the problems with the linear spring-dashpot model. The contact force is modeled as:

$$f_c = k(x - x_e)^n + b\dot{x}(x - x_e)^n \tag{4.8}$$

where parameter n depends on the material and geometry of the contact surface, and is usually valued between 1 and 2 [16].

As with the spring-dashpot model, the damping parameter b can be related to the coefficient of restitution, since both are related to the energy dissipated by the impact process. Here, unlike the Kelvin-Voigt model, the damping depends on the indentation making the model more physically sound as a plastic region is more likely to develop for larger indentations. Another advantage is that the contact force has no discontinuities at initial contact and separation, but it begins and finishes with the correct value of zero.

5

Controller Design

In this chapter, the control strategies used in this thesis are proposed and their derivations are shown. A set of controllers are designed for each contact model. Separate design approaches are followed depending on the contact model.

5.1 Adaptive Control Design for the Kelvin-Voigt Contact Model

For the Kelvin-Voigt(KV) contact model three control strategies were proposed:

1. Pure adaptive control
2. Adaptive + Proportional(P) control
3. Adaptive + Proportional-Integral(PI) control

All controllers follow the same strategy in terms of design, and they all provide asymptotic force trajectory tracking under exact or asymptotically exact inner loop velocity tracking.

5.1.1 Adaptive Control and Adaptive + P Control

Choosing the control law

Under the assumptions stated in Chapter 4, a force controller for the KV contact model (4.7) was selected such that the force trajectory tracking errors would asymptotically converge to zero. For this purpose the following control law was proposed:

$$\dot{x}_d = \hat{\gamma}_0 + \hat{\gamma}_1 x + \hat{\gamma}_2 f_d \tag{5.1}$$

If the parameters of the model k, b, and x_e were known, the controller would provide asymptotic force trajectory tracking for the following control parameter values

$$\gamma_0^* = \frac{k}{b}x_e \qquad \gamma_1^* = -\frac{k}{b} \qquad \gamma_2^* = \frac{1}{b} \tag{5.2}$$

Plugging the controller (5.1) into the equation (4.1) and assuming that the parameter values are as in (5.2), the closed loop equation can be written as:

$$\begin{aligned}
\Delta\dot{x} &= \dot{x} - \hat{\gamma}_0 - \hat{\gamma}_1 x - \hat{\gamma}_2 f_d \\
&= \dot{x} - \frac{k}{b}x_e + \frac{k}{b}x - \frac{1}{b}f_d \\
&= \frac{1}{b}[k(x - x_e) + b\dot{x} - f_d] \\
&= \frac{1}{b}(f_c - f_d) \qquad (5.3)
\end{aligned}$$

Under the exact or asymptotically exact inner loop velocity tracking assumption (4.5), (5.3) indicates the controller (5.1) can achieve at least asymptotically exact force trajectory tracking.

Choosing the adaptation laws

In order to derive the adaptation laws for the controller parameters, the force tracking error ($\Delta f = f_c - f_d$) had to be written in the form given by (3.1).

$$\tilde{\gamma}_0 = \hat{\gamma}_0 - \gamma_0^* \qquad \tilde{\gamma}_1 = \hat{\gamma}_1 - \gamma_1^* \qquad \tilde{\gamma}_2 = \hat{\gamma}_2 - \gamma_2^* \qquad (5.4)$$

The parameter errors (5.4) were defined as the difference between the controller parameter provided by the update law and the ideal parameters.

The force tracking error can be driven by manipulating the equation (4.1) as follows:

$$\begin{aligned}
b\Delta\dot{x} &= b\dot{x} - b\dot{x}_d \\
&= b\dot{x} - b(\hat{\gamma}_0 + \hat{\gamma}_1 x + \hat{\gamma}_2 f_d) \\
&= b\dot{x} - b(\gamma_0^* + \gamma_1^* x + \gamma_2^* f_d) - b(\tilde{\gamma}_0 + \tilde{\gamma}_1 x + \tilde{\gamma}_2 f_d) \\
&= b\dot{x} - kx_e + kx - f_d - b(\tilde{\gamma}_0 + \tilde{\gamma}_1 x + \tilde{\gamma}_2 f_d) \\
b\Delta\dot{x} &= \Delta f - b(\tilde{\gamma}_0 + \tilde{\gamma}_1 x + \tilde{\gamma}_2 f_d) \qquad (5.5)
\end{aligned}$$

Keeping in mind that the inner loop controller provides asymptotically exact velocity tracking, equation (5.5) can be written as

$$\begin{aligned}
\Delta f &= b(\tilde{\gamma}_0 + \tilde{\gamma}_1 x + \tilde{\gamma}_2 f_d) \\
&= b\begin{bmatrix}\tilde{\gamma}_0 & \tilde{\gamma}_1 & \tilde{\gamma}_2\end{bmatrix}\begin{bmatrix}1 \\ x \\ f_d\end{bmatrix} \qquad (5.6)
\end{aligned}$$

The relation between the force error and the parameter error given in (5.6) is the same as (3.1). Hence, according to Lemma 1 the update laws are

$$\dot{\hat{\gamma}}_0 = -sgn(b)\alpha_0\Delta f \qquad (5.7a)$$

$$\dot{\hat{\gamma}}_1 = -sgn(b)\alpha_1\Delta f x \qquad (5.7b)$$

$$\dot{\hat{\gamma}}_2 = -sgn(b)\alpha_2\Delta f f_d \qquad (5.7c)$$

where α_0, α_1, and α_2 are strictly positive adaptive gains.

Alternatively, the update laws in (5.7) can be written as in (5.8) to counter parameter drift. In adaptive control literature, this is referred to as fixed σ-modification [5].

$$\dot{\hat{\gamma}}_0 = -sgn(b)\alpha_0\Delta f - \sigma_0\hat{\gamma}_0 \tag{5.8a}$$
$$\dot{\hat{\gamma}}_1 = -sgn(b)\alpha_1\Delta f x - \sigma_1\hat{\gamma}_1 \tag{5.8b}$$
$$\dot{\hat{\gamma}}_2 = -sgn(b)\alpha_2\Delta f f_d - \sigma_2\hat{\gamma}_2 \tag{5.8c}$$

where $\sigma_{0,1,2}$ are positive design parameters.

Controller (5.1) can also be enriched with a proportional force error term as in equation (5.9).

$$\dot{x}_d = \hat{\gamma}_0 + \hat{\gamma}_1 x + \hat{\gamma}_2 f_d - K_p\Delta f \tag{5.9}$$

The derivation of the adaptation laws for this controller was not shown here, as it was discovered that augmenting the pure adaptive controller with a proportional controller did not change the update laws given in (5.7) and (5.8).

5.1.2 Adaptive + PI Control

The controller given in (5.1) was augmented with a PI controller to obtain:

$$\dot{x}_d = \hat{\gamma}_0 + \hat{\gamma}_1 x + \hat{\gamma}_2 f_d - k_p\Delta f - k_i\int \Delta f\, dt \tag{5.10}$$

Even though the derivation procedure for the adaptation laws was exactly the same, the newly introduced integral term changed how the parameter errors were related to the force error and therefore, they had to be updated as follows:

$$\begin{aligned}
b\Delta\dot{x} &= b\dot{x} - b\dot{x}_d \\
&= b\dot{x} - b(\hat{\gamma}_0 + \hat{\gamma}_1 x + \hat{\gamma}_2 f_d) + bk_p\Delta f + bk_i\int \Delta f\, dt \\
&= b\dot{x} - b(\gamma_0^* + \gamma_1^* x + \gamma_2^* f_d) - b(\tilde{\gamma}_0 + \tilde{\gamma}_1 x + \tilde{\gamma}_2 f_d) + bk_p\Delta f + bk_i\int \Delta f\, dt \\
&= b\dot{x} - kx_e + kx - f_d - b(\tilde{\gamma}_0 + \tilde{\gamma}_1 x + \tilde{\gamma}_2 f_d) + bk_p\Delta f + bk_i\int \Delta f\, dt \\
&= (1 + k_p)\Delta f + bk_i\int \Delta f\, dt - b(\tilde{\gamma}_0 + \tilde{\gamma}_1 x + \tilde{\gamma}_2 f_d) \qquad (5.11)
\end{aligned}$$

Considering an auxiliary state $z(t) = \int \Delta f(t)\, dt$, equation (5.11) becomes:

$$b\Delta\dot{x} = (1 + bk_p)\dot{z} + bk_i z - b(\tilde{\gamma}_0 + \tilde{\gamma}_1 x + \tilde{\gamma}_2 f_d) \tag{5.12}$$

due to the inner loop velocity control assumption (4.5):

$$(1 + bk_p)\dot{z} + bk_i z = b(\tilde{\gamma}_0 + \tilde{\gamma}_1 x + \tilde{\gamma}_2 f_d) \tag{5.13}$$

Equation (5.13) can be transformed in Laplace domain and rewritten as:

$$z = \frac{b}{(bk_p + 1)s + bk_i}(\tilde{\gamma}_0 + \tilde{\gamma}_1 x + \tilde{\gamma}_2 f_d) \tag{5.14}$$

Then the adaptation laws can be chosen as follows:

$$\dot{\hat{\gamma}}_0 = -sgn(b)\alpha_0 z \tag{5.15a}$$
$$\dot{\hat{\gamma}}_1 = -sgn(b)\alpha_1 z x \tag{5.15b}$$
$$\dot{\hat{\gamma}}_2 = -sgn(b)\alpha_2 z f_d \tag{5.15c}$$

Refer to (5.16) for the update laws with σ-modification applied.

$$\dot{\hat{\gamma}}_0 = -sgn(b)\alpha_0 z - \sigma_0 \hat{\gamma}_0 \tag{5.16a}$$
$$\dot{\hat{\gamma}}_1 = -sgn(b)\alpha_1 z x - \sigma_1 \hat{\gamma}_1 \tag{5.16b}$$
$$\dot{\hat{\gamma}}_2 = -sgn(b)\alpha_2 z f_d - \sigma_2 \hat{\gamma}_2 \tag{5.16c}$$

Normally, Lemma 1 is enough proof to show that the controller is asymptotically stable under exact or asymptotically exact inner loop velocity tracking. However, for the sake showing another approach, controller's stability was also proven using Lyapunov-like analysis [1] as below:

Starting with the rearrangement of the terms of equation (5.12) as follows:

$$\dot{z} = -\frac{b}{(1 + bk_p)}k_i z + \frac{b}{(1 + bk_p)}(\tilde{\gamma}_0 + \tilde{\gamma}_1 x + \tilde{\gamma}_2 f_d) + \frac{b}{(1 + bk_p)}\Delta\dot{x} \tag{5.17}$$

yields the dynamical equation of the closed-loop system. In order to continue, an additional assumption has to be made regarding the inner loop velocity tracking which is $\Delta\dot{x}(t) \in \mathcal{L}_2$, i.e.:

$$\int_0^\infty \|\Delta\dot{x}(t)\|^2 \, dt < \infty \tag{5.18}$$

As most velocity controllers that guarantee asymptotically exact velocity tracking guarantee that the velocity error is $\mathcal{L}_2$, no generality was lost by this assumption[1].

Next, define:

$$\psi(t) = b\Delta\dot{x}(t) \tag{5.19}$$

and keep in mind that:

$$\Delta\dot{x}(t) = 0 \quad \forall t \Rightarrow \psi(t) = 0 \quad \forall t \tag{5.20}$$

$$\Delta\dot{x}(t) \in \mathcal{L}_2, \ \lim_{t\to\infty} \Delta\dot{x}(t) = 0 \Rightarrow \psi(t) \in \mathcal{L}_2, \ \lim_{t\to\infty} \psi(t) = 0 \tag{5.21}$$

Since $\psi(t) \in \mathcal{L}_2$, there exists a number $0 \leq M < \infty$ such that:

$$\int_0^t \psi(\tau)^2 \, d\tau \leq 4k_i bM \quad \forall t \in [0, \infty) \tag{5.22}$$

The following Lyapunov function candidate was selected:

$$V(t, z, \tilde{\gamma}) = \frac{1 + bk_p}{2} z^2 + \frac{b}{2\alpha_0}\tilde{\gamma}_0^2 + \frac{b}{2\alpha_1}\tilde{\gamma}_1^2 + \frac{b}{2\alpha_2}\tilde{\gamma}_2^2 + M - \frac{1}{4k_i b}\int_0^t \psi(\tau)^2 \, d\tau \tag{5.23}$$

Equations (5.22) and (5.23) imply that $V \geq 0$ for all time t, and that V is radially unbounded in z and $\tilde{\gamma}$. Equation (5.23) is not a conventional Lyapunov candidate as a conventional Lyapunov function is required to be equal to zero when z and $\tilde{\gamma}$ are set to zero[10], yet $V(t, 0, 0) = M$ where M may not be equal to zero.

Differentiating (5.23) along the trajectories of the closed loop system (5.17) to obtain:

$$\begin{aligned}\dot{V}(t,z,\tilde{\gamma}) =&(1+bk_p)z\dot{z} + \frac{b}{\alpha_0}\tilde{\gamma}_0\dot{\tilde{\gamma}}_0 + \frac{b}{\alpha_1}\tilde{\gamma}_1\dot{\tilde{\gamma}}_1 + \frac{b}{\alpha_2}\tilde{\gamma}_2\dot{\tilde{\gamma}}_2 - \frac{1}{4k_ib}\psi^2(t)\\ =&(1+bk_p)z\left[-\frac{b}{(1+bk_p)}k_iz + \frac{b}{(1+bk_p)}(\tilde{\gamma}_0+\tilde{\gamma}_1x+\tilde{\gamma}_2f_d) + \frac{b}{(1+bk_p)}\Delta\dot{x}\right]\\ &+ \frac{b}{\alpha_0}\tilde{\gamma}_0\dot{\tilde{\gamma}}_0 + \frac{b}{\alpha_1}\tilde{\gamma}_1\dot{\tilde{\gamma}}_1 + \frac{b}{\alpha_2}\tilde{\gamma}_2\dot{\tilde{\gamma}}_2 - \frac{1}{4k_ib}\psi^2(t)\\ =&\left[b(\tilde{\gamma}_0+\tilde{\gamma}_1x+\tilde{\gamma}_2f_d)z + \frac{b}{\alpha_0}\tilde{\gamma}_0\dot{\tilde{\gamma}}_0 + \frac{b}{\alpha_1}\tilde{\gamma}_1\dot{\tilde{\gamma}}_1 + \frac{b}{\alpha_2}\tilde{\gamma}_2\dot{\tilde{\gamma}}_2\right]\\ &+\left[-bk_iz^2 + \psi(t)z - \frac{1}{4k_ib}\psi^2(t)\right] \qquad (5.24)\end{aligned}$$

Note that $\dot{\tilde{\gamma}} = \dot{\hat{\gamma}}$, since $\tilde{\gamma} = \hat{\gamma} - \gamma^*$ and γ^* is constant as it represents the optimal parameter value. Now, plugging equations (5.15a) to (5.15c) into (5.24) to obtain:

$$\dot{V}(t,z,\tilde{\gamma}) = -\left[\sqrt{k_ib}z - \frac{1}{2\sqrt{k_ib}}\psi(t)\right]^2 \qquad (5.25)$$

Thus, $\dot{V}(t,z,\tilde{\gamma}) \le 0\ \forall t$. Now according to Lyapunov's stability theorem[10], since $V \ge 0$ and $\dot{V} \le 0$ for all time t, $z(t) = \int \Delta f(t)\,dt$ and $\tilde{\gamma}$(and consequently $\hat{\gamma}$) are bounded. Also,

$$\int_0^\infty \left[\sqrt{k_ib}z - \frac{1}{2\sqrt{k_ib}}\psi(t)\right]^2 dt \le V(0,z(0),\tilde{\gamma}(0)) < \infty \qquad (5.26)$$

meaning that:

$$\left[\sqrt{k_ib}z - \frac{1}{2\sqrt{k_ib}}\psi(t)\right] \in \mathcal{L}_2 \qquad (5.27)$$

If either (5.20) or (5.21) holds, then:

$$\frac{1}{2\sqrt{k_ib}}\psi(t) \in \mathcal{L}_2 \qquad (5.28)$$

According to Minkowski's Theorem[22], for two functions f and g that are in $\mathcal{L}_p$, $f(.) + g(.) \in \mathcal{L}_p$ for $p \in [1,\infty]$. Based on this, it can be seen that:

$$\sqrt{k_ib}z = \left[\sqrt{k_ib}z - \frac{1}{2\sqrt{k_ib}}\psi(t)\right] + \frac{1}{2\sqrt{k_ib}}\psi(t) \in \mathcal{L}_2 \qquad (5.29)$$

Knowing that f_d, $z(t)$, and $\tilde{\gamma}$ are all bounded it can be observed that $\dot{z}(t)$(5.17) is bounded as well which proves the uniform continuity of $z(t)$[4]. Finally, by Barbalat's Lemma[4]:

$$\lim_{t\to\infty} z(t) = 0 \tag{5.30}$$

5.2 Adaptive Control Design for the Hunt-Crossley Contact Model

The approach for selecting the controller structure is exactly the same as before. The control law has to provide asymptotically exact force tracking under asymptotically exact inner loop control velocity tracking.

For the HC contact model(4.8) the following control law was chosen:

$$\dot{x}_d = \hat{\gamma}_0 + \hat{\gamma}_1 \frac{1}{(x - \hat{x}_e)^{\hat{n}}} f_d \tag{5.31}$$

where

$$\hat{\gamma}_0 = -\frac{\hat{k}}{\hat{b}} \qquad \hat{\gamma}_1 = \frac{1}{\hat{b}} \tag{5.32}$$

As can be seen, the control law in equation (5.31) is non-linear. Unless this non-linearity is linearly parameterized, regular adaptive control techniques cannot be applied. In this case, a neuro-adaptive control approach was adopted and three controllers were proposed:

1. Pure neuro-adaptive control
2. Neuro-adaptive + P control
3. Neuro-adaptive + PI control

5.2.1 Neuro-adaptive Control and Neuro-adaptive + P Control

Choosing the control law

The control law (5.33) provides asymptotically exact force trajectory tracking under the inner loop velocity control assumptions made in Chapter 4. The controller makes use of a linear in the weights neural network to approximate the nonlinearity in (5.31).

$$\dot{x}_d = \hat{w}^\top(t) J(x) f_d + \hat{\theta} \tag{5.33}$$

The first term on the right hand side corresponds to the single layer neural network that describes the non-linearity and the second term is the bias term, i.e.:

$$\hat{w}^\top(t)J(x) = \frac{1}{\hat{b}(x-\hat{x}_c)^{\hat{n}}} \qquad \hat{\theta} = -\frac{\hat{k}}{\hat{b}} \tag{5.34}$$

The parameter $\hat{w}$ corresponds to the estimated weight matrix of the neural network and $J(x)$ to the matrix of regressor terms.

Just like in the earlier scenarios, the force error has to be expressed in terms of the parameter errors such that the update laws can be driven.

For $\dot{x} = w^{*\top}J(x)f_c + \theta^*$ and (5.33), the velocity error $\Delta\dot{x}$ can be written as:

$$\Delta\dot{x} = w^{*\top}J(x)f_c - \hat{w}^\top(t)J(x)f_d - \tilde{\theta} \tag{5.35}$$

Equation (5.35) can be manipulated as below to obtain $\Delta f(t)$:

$$\begin{aligned}\Delta\dot{x} &= w^{*\top}J(x)f_c - w^{*\top}J(x)f_d + w^{*\top}J(x)f_d - \hat{w}^\top(t)J(x)f_d - \tilde{\theta}\\ &= w^{*\top}J(x)\Delta f - \tilde{w}J(x)f_d - \tilde{\theta}\end{aligned} \tag{5.36}$$

The inner loop velocity controller provides exact or at least asymptotically exact velocity tracking hence:

$$\begin{aligned}w^{*\top}J(x)\Delta f &= \tilde{w}J(x)f_d + \tilde{\theta}\\ \Delta f &= (w^{*\top}J(x))^{-1}(\tilde{w}J(x)f_d + \tilde{\theta})\\ &= (w^{*\top}J(x))^{-1}\begin{bmatrix}\tilde{w} & \tilde{\theta}\end{bmatrix}\begin{bmatrix}J(x)f_d\\ 1\end{bmatrix}\end{aligned} \tag{5.37}$$

Finally, the update laws are:

$$\dot{\hat{w}} = -\alpha_0 \Delta f J(x) f_d \tag{5.38a}$$

$$\dot{\hat{\theta}} = -\alpha_1 \Delta f \tag{5.38b}$$

Refer to (5.39) for update laws with σ-modification.

$$\dot{\hat{w}} = -\alpha_0 \Delta f J(x) f_d - \sigma_w \hat{w} \tag{5.39a}$$

$$\dot{\hat{\theta}} = -\alpha_1 \Delta f - \sigma_\theta \hat{\theta} \tag{5.39b}$$

The P variant of controller (5.33) can be seen below:

$$\dot{x}_d = \hat{w}^\top(t)J(x)f_d + \hat{\theta} - k_p\Delta f \tag{5.40}$$

It was observed that the addition of the proportional feedback controller did not alter the derivation procedure. Controller (5.40) shares the same update laws as controller (5.33).

5.2.2 Neuro-adaptive + PI Control

The PI variant of controller (5.33) can be seen below:

$$\dot{x}_d = \hat{w}^\top(t)J(x)f_d + \hat{\theta} - k_p\Delta f - k_i \int \Delta f\, dt \tag{5.41}$$

The strategy for deriving the adaptation laws for this controller and the adaptive + PI controller are basically the same.

For $\dot{x} = w^{*\top} J(x) f_c + \theta^*$ and (5.41), the velocity error $\Delta\dot{x}$ can be written as:

$$\Delta\dot{x} = w^{*\top} J(x)\Delta f - \tilde{w}J(x)f_d - \tilde{\theta} + k_p\Delta f + k_i \int \Delta f\, dt \tag{5.42}$$

Substituting $z(t) = \int \Delta f(t)\, dt$, equation (5.42) becomes:

$$\Delta\dot{x} = (w^{*\top} J(x) + k_p)\dot{z} + k_i z - \tilde{\theta} - \tilde{w}J(x)f_d \tag{5.43}$$

Since the inner loop velocity controller provides exact or at least asymptotically exact velocity tracking, equation (5.43) can be rewritten as:

$$(w^{*\top} J(x) + k_p)\dot{z} + k_i z = \tilde{\theta} + \tilde{w}J(x)f_d \tag{5.44}$$

Applying Laplace transformation to equation (5.44) to obtain:

$$z = \frac{1}{s(w^{*\top} J(x) + k_p) + k_i} \begin{bmatrix} \tilde{w} & \tilde{\theta} \end{bmatrix} \begin{bmatrix} J(x)f_d \\ 1 \end{bmatrix} \tag{5.45}$$

Finally, the update laws are:

$$\dot{\hat{w}} = -\alpha_0 z J(x) f_d \tag{5.46a}$$

$$\dot{\hat{\theta}} = -\alpha_1 z \tag{5.46b}$$

Refer to (5.47) for update laws with σ-modification.

$$\dot{\hat{w}} = -\alpha_0 z J(x) f_d - \sigma_w \hat{w} \tag{5.47a}$$

$$\dot{\hat{\theta}} = -\alpha_1 z - \sigma_\theta \hat{\theta} \tag{5.47b}$$

6
Experiments and Results

In this chapter, steps for modelling a neural network based testing environment are covered, and the results regarding the accuracy of this approach are reported. Next, outcomes of the controller tests on the said environment are provided, and the controllers are compared on the basis of their tracking capacities. Additionally, the effects of different trajectories on performance are investigated.

6.1 Approximation of Contact Model Dynamics

In order to analyze the behavior of the controllers proposed in Chapter 6, a testing environment that would replicate the dynamical properties of a compliant object was required. In this case, a sponge was selected for this purpose, and a neural network was designed using the supervised learning approach for estimating the dynamics of the sponge. Now, there are three major things to keep in mind:

1. The robotic manipulator used in this project is velocity controlled.
2. The only accessible states of the robot are its end-effector position and velocity.
3. The controllers are responsible for adjusting the velocity trajectory of the end-effector based on the error between the reference force trajectory and contact force readings.

Therefore, the network model, given some inputs, needs to output the contact force readings of the force/torque sensor of the robot when the end-effector interacts with the environment.

6.1.1 Training Setup

Step 1: Input-Output Selection

It goes without saying that data is needed to train any neural network. However, before getting started with data collection the inputs and output(s) of the network have to be defined. For this case, the output was chosen to be the contact force. Selection of the inputs, on the other hand, was not as straightforward. In Chapter 5, the inputs to the environment were stated as the robot's position and velocity. The issue is the network can not distinguish between scalar and vector quantities, and if these inputs alone were used, it would not be able to tell at which direction the force vector is pointing. The direction information is necessary to understand whether the end-effector is moving towards the sponge or away from it. To resolve this, apart from the end-effector's current position and velocity, its previous position was selected as an input. This way the network would be able to recognize the force vector direction by looking at the change in position.

As of now, the inputs and the output of the network are as follows:

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

Step 2: Data Collection

Collecting the data was quite straightforward. The objective was to record force and position data as the end-effector probed the sponge without losing contact with the surface of the sponge. For this purpose, the following compliance control scheme was used:

$$\upsilon = \dot{p}_r + \alpha F - K_p(p - p_r) \tag{6.1}$$

where $\dot{p}_r$ is the desired velocity, F is the measured contact force, p is the actual position of the end-effector, and p_r is the desired position trajectory.

The controller (6.1) is the combination of a simple P-only controller with gain K_p, a feedforward term and a compliance term with gain α which maps the contact force to the end-effector position. The controller was not tuned as it did not need to function perfectly. In fact, its only purpose was to track a trajectory along the normal axis to the surface of the sponge so that data could be collected. The compliance term was added to make sure that the robot does not apply too much force to the sponge.

To ensure the data was rich enough for training, a complex sinusoidal trajectory that consisted of several sinusoids with unique frequencies and amplitudes was used. In order to make sure that the robot never lost contact with the surface, the tracking was started from a position where the end-effector's tip was slightly pressed into the sponge. Additionally, the amplitude of the trajectory was chosen such that the end-effector would never back away from its starting point leaving the surface. At each time step, position of the end-effector and the force readings from the force/torque sensor were stored.

Initially, velocity data was collected as well. However, after seeing how noisy it was compared to the position data, it was discarded. Instead, velocity information was driven by taking the derivative of the position as follows:

$$\upsilon_t = \frac{dp}{dt} = \frac{p_t - p_{t-1}}{\Delta t} \tag{6.2}$$

where $\Delta t = 0.008$ is the step time which is equivalent to the refresh rate of the ROS node 125Hz.

Step 3: Data Preparation

Before moving on to training, the data had to be processed. It was observed that both position and force data contained a significant amount of high frequency measurement noise. To clear the data of its high frequency content, a second order Butterworth filter with a cutoff frequency of 0.015Hz was used. How the filtered position and force data compare to their unfiltered counterparts can be seen in figure 6.1.

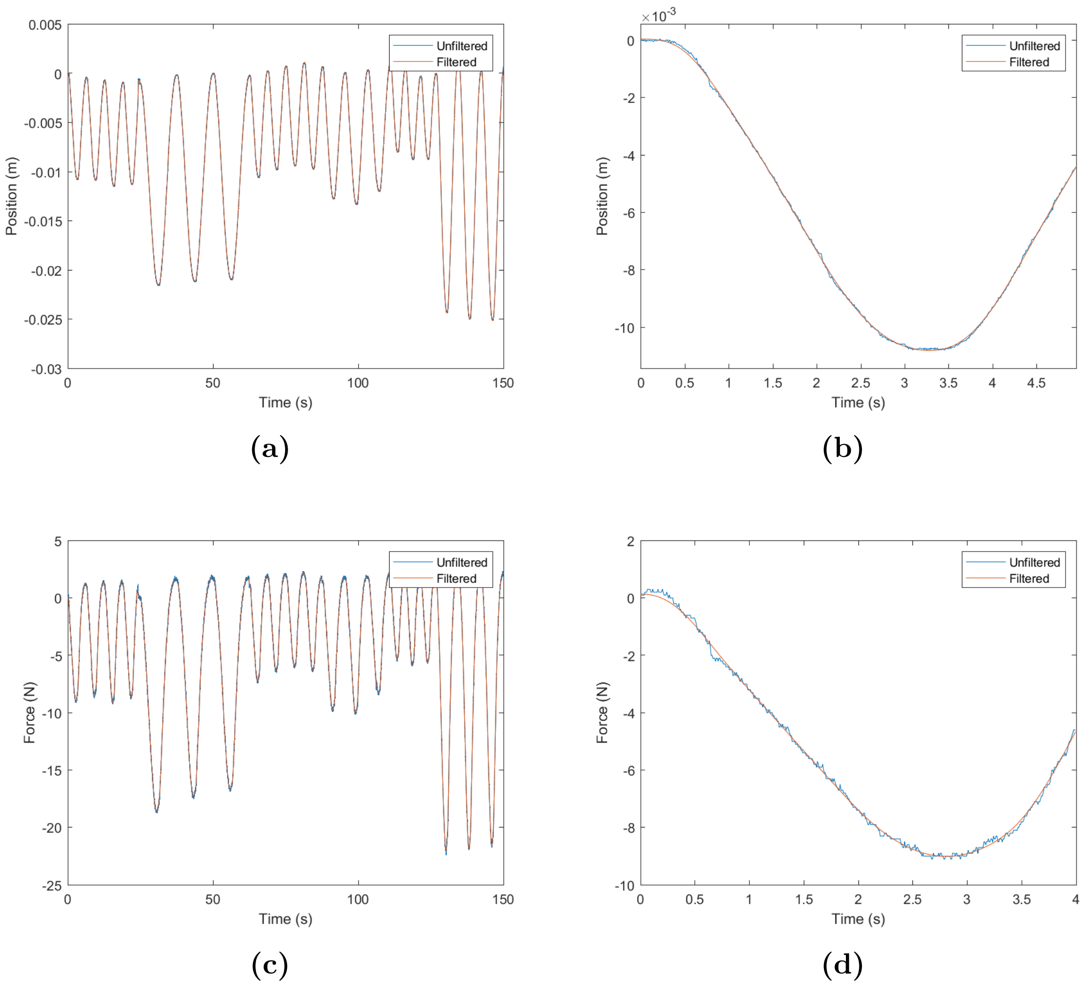


Figure 6.1: Filtered vs Unfiltered data. (a)Position. (b)Position-Zoomed in. (c)Force. (d)Force-Zoomed in

Since the velocity had to be driven from the position, the cutoff frequency was chosen after experimenting with the behavior of velocity for different cutoff frequencies. At first, a cutoff frequency of 0.1 Hz was chosen then it was gradually lowered until the velocity data no longer contained any outliers, large spikes, etc. The difference between the velocity obtained from the filtered position and the unfiltered position can be seen in figure 6.2.

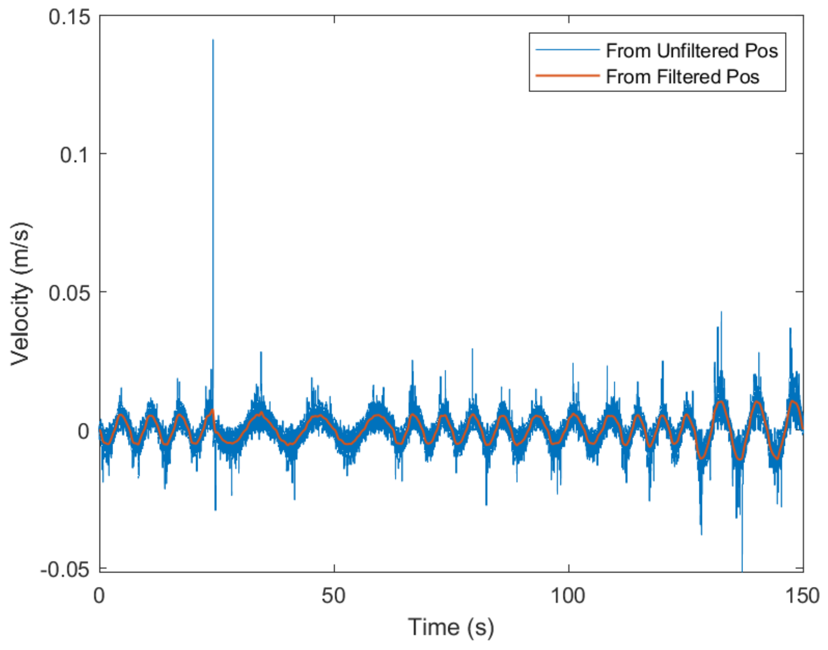
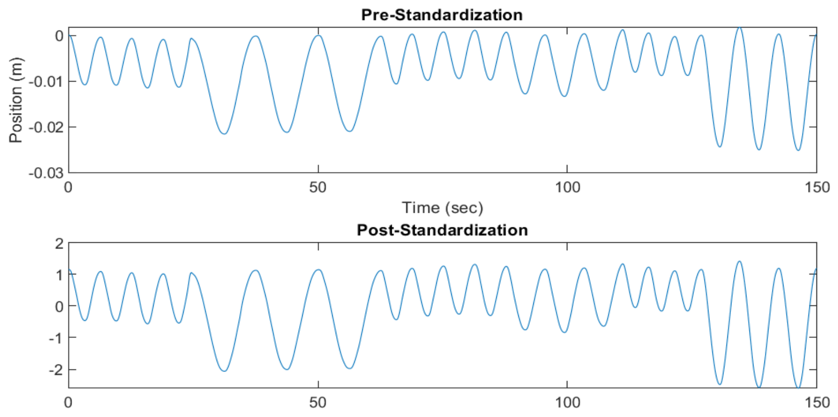


Figure 6.2: Unfiltered vs Filtered velocity

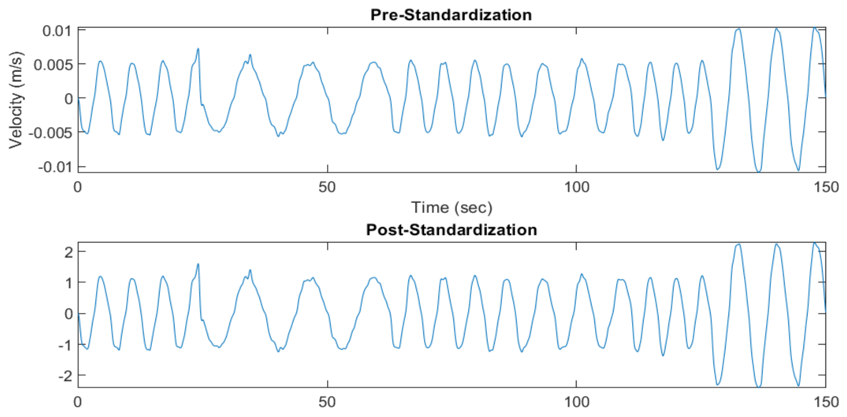
After the data was filtered, it was separated into three different sets, namely the training, validation, and test sets. The training set is the portion of the data reserved for training the network and it is also the largest set among the three. The validation set is used for evaluating the network's performance at regular intervals during training. The network does not directly learn from this set, i.e., updates weights and biases, but validation metrics can be used to tune the model hyperparameters. The test set, unlike the other two sets, is not available to the network during training and is used for checking whether the trained network is capable of generalization. In other words, this set shows how well the network performs when dealing with unseen data.

For this project, %60 of the entire data set was assigned to the training set, while the validation and test sets got %20 each. By using MATLAB's **dividerand** function, the data instances were distributed among the three sets at random. Randomization of the data is important, as it can reduce the chance of overfitting the network.

Finally, the data had to be standardized. Although it is a preprocessing step, standardization can be considered a part of the training process as well; it is only applied to the training set, and the mean and standard deviation information are stored so that the validation and test sets can be standardized based on these metrics – in other words, the network learns these values in a way. For this particular application, the target data was not standardized. In figure 6.3, it can be seen how the standardized input features look like compared to their non-standardized counterparts.



(a)



(b)

Figure 6.3: Standardized vs Non-standardized input features. (a)Position. (b)Velocity.

6.1.2 Training Process

The network was designed and trained using MATLAB's deep learning toolbox. The decisions behind the network's architecture were based on another thesis work[46] that investigated a similar problem, i.e., the approximation of contact dynamics of compliant objects. In their case, the network predicted the velocity of the robot's end-effector for given position and force inputs.

The network used in this project (see figure 6.4) consists of 2 hidden layers with 32 neurons each. As the activation function hyperbolic tangent TanH was chosen after experimenting with both TanH and Softsign functions and observing no significant difference in training results. Another reason why TanH was favored over Softsign was due to it being readily available in the deep learning toolbox. The output layer did not have an activation function which is the reason why the target outputs were not standardized in the first place.

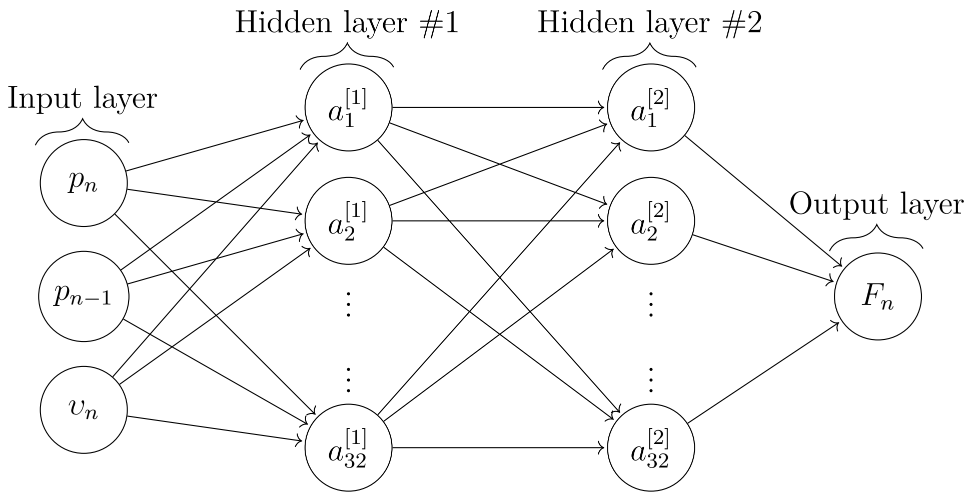


Figure 6.4: Architecture of the network

Mean squared error(MSE) loss was picked as the loss function and it was paired with ADAM optimizer for updating the network parameters. The hyperparameters of ADAM were selected as follows:

Parameter	**Value**
Learning rate	0.001
β_1	0.9
β_2	0.999

Table 6.1: ADAM optimizer parameters

The network was trained for 300 epochs with a mini-batch size of 128. A comparison between the predictions of the network and the target output for different data sets can be seen in figure 6.5. Refer to table 6.2 for the root mean squared error(RMSE) values between the predictions and targets, and to table 6.3 for the loss values for the training and validation sets.

Set	**RMSE**
Training Set	0.5594
Validation Set	0.5566
Test Set	0.5574
Entire Set	0.5584

Table 6.2: RMSE values between the predictions and target outputs

Set	**Initial Loss**	**Final Loss**
Training Set	32.0931	0.1010
Validation Set	26.9871	0.1549

Table 6.3: Initial and final loss values of the training and validation sets

Considering the target force output varies in the interval $[-22, 2]$, it is safe to say that the network's performance is satisfactory, and since the training and validation losses are fairly small and close to each other, the network does not suffer from high bias nor high variance.

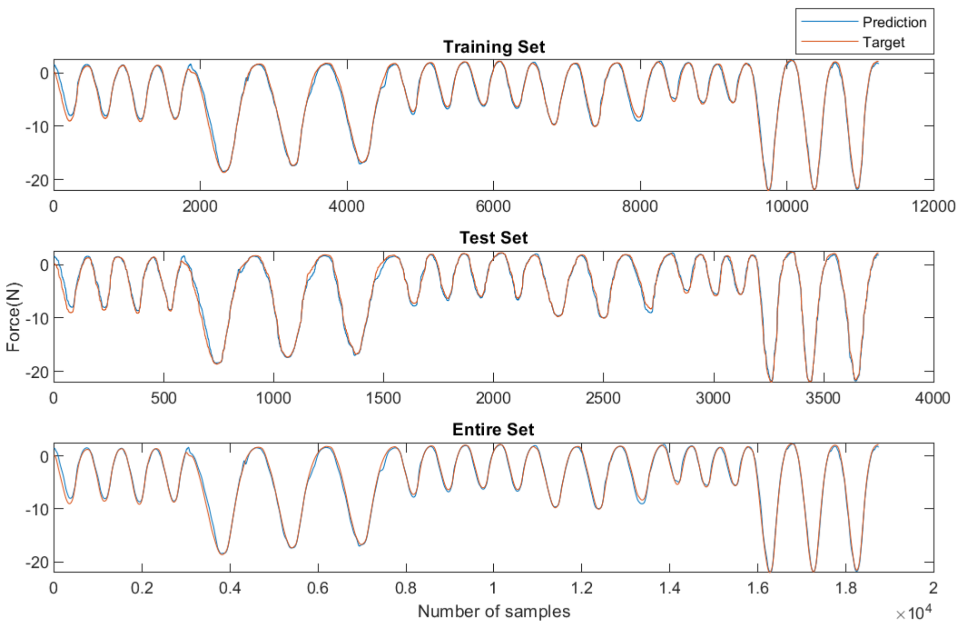


Figure 6.5: Predictions of the network vs Target outputs

6.2 Control Experiments

Tracking capacity and step response are two important metrics to assess a controller's overall performance. Tracking capacity describes how well a system is able to follow a given reference trajectory under a specific control law. Step response, on the other hand, is the measure of a system's reaction to a sudden change in input. It consists of two parts, namely the transient response and the steady state response. The steady state response of a system corresponds to the time interval in which the changes in the system's states are infinitesimally small, and the transient response is a system's initial reaction to an input until it reaches steady state.

In this section, the controllers proposed in Chapter 6 were evaluated based on these two metrics. Additionally, the effects of input magnitude and frequency on performance were investigated. Finally, the convergence of estimated parameters were examined.

6.2.1 Tracking Capacity and Step Response

To get the best out of each controller, the tracking capacity and step response had to be examined simultaneously. The tracking performance of the controllers were evaluated based on the root mean squares of their tracking errors. Meanwhile, properties such as the amount of overshoot, settling time (ST) and steady-state error (SSE) were considered when evaluating the quality of the response to a sudden change in input.

The tracking capacity was tested on a sinusoidal trajectory of the form:

$$f_d(t) = 0.5\sin(2\pi f t + \frac{\pi}{2}) + 0.5 \quad f = 0.1Hz \tag{6.3}$$

In order to examine the transient and steady state responses, a unit step signal was used. The unit step is defined as:

$$s(t) = \begin{cases} 0 & t \leq 1 \\ 1 & t > 1 \end{cases} \tag{6.4}$$

Before the controllers were tuned, a set of upper limits were defined for the previously described performance indices. Tracking RMSE, overshoot and steady-state error limits were set with respect to the input magnitude. An upper limit for the settling time was not set as it is closely related to the steady-state error. Instead, it was simply calculated based on the point where the force tracking error becomes less than or equal to %1 of the input magnitude. Refer to table 6.4 for the aforementioned limits.

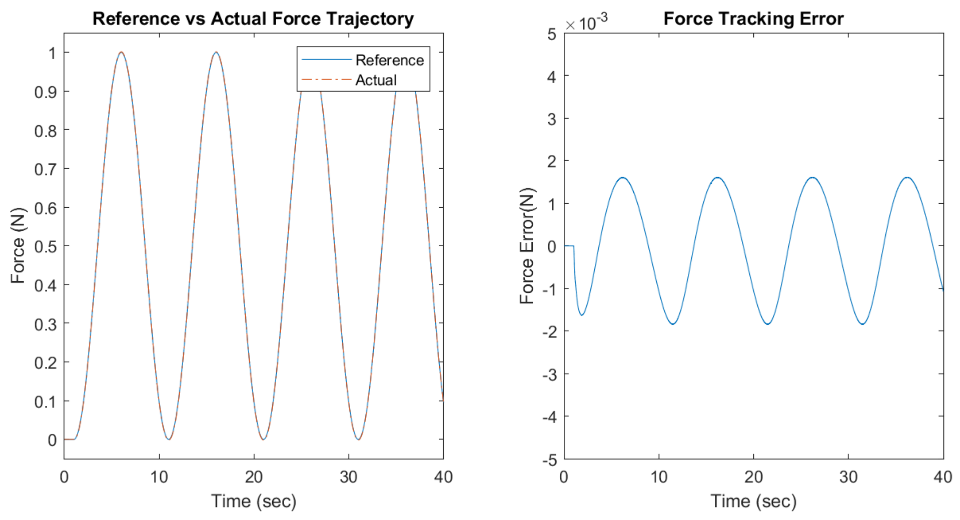
Index	**Limit**
Tracking RMSE	$\leq$ %1
Overshoot	$\leq$ %40
Steady-state Error	$\leq$ %1

Table 6.4: Upper limits for performance indices

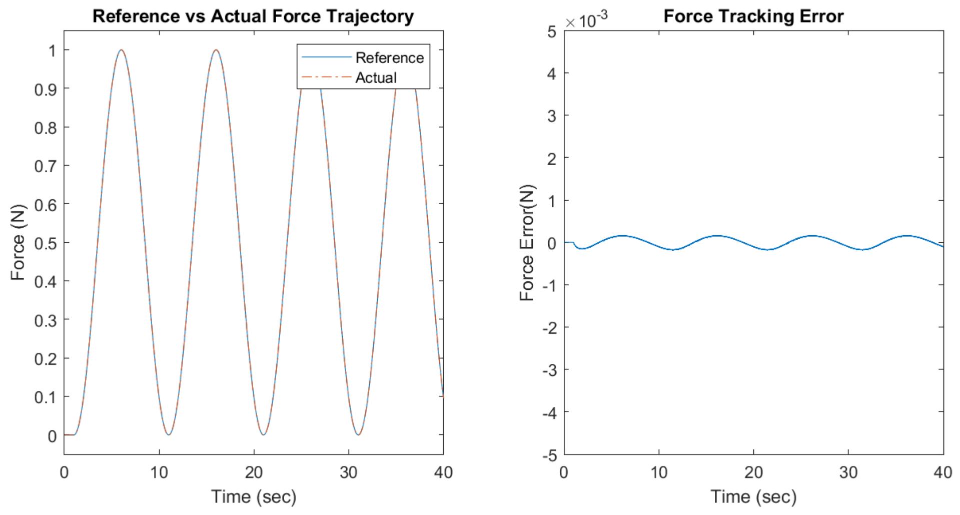
The tracking test results are presented in figures 6.6 and 6.7. Step responses of the controllers can be seen in figures 6.8 and 6.9. In table 6.5, the performance indices are reported.

Controller	**RMSE**	**Overshoot**	**SSE**	**ST**
Adaptive	0.0012	%39.53	%0.034	0.74s
Adaptive + P	0.00011	%39.12	%0.015	0.06s
Adaptive + PI	0.00016	%32.33	%0.02	0.08s
Neuro-Adaptive	0.0030	%36.47	%0.04	0.89s
Neuro-Adaptive + P	0.00025	%15.55	%0.01	0.09s
Neuro-Adaptive + PI	0.00014	%16.26	%0.03	0.2s

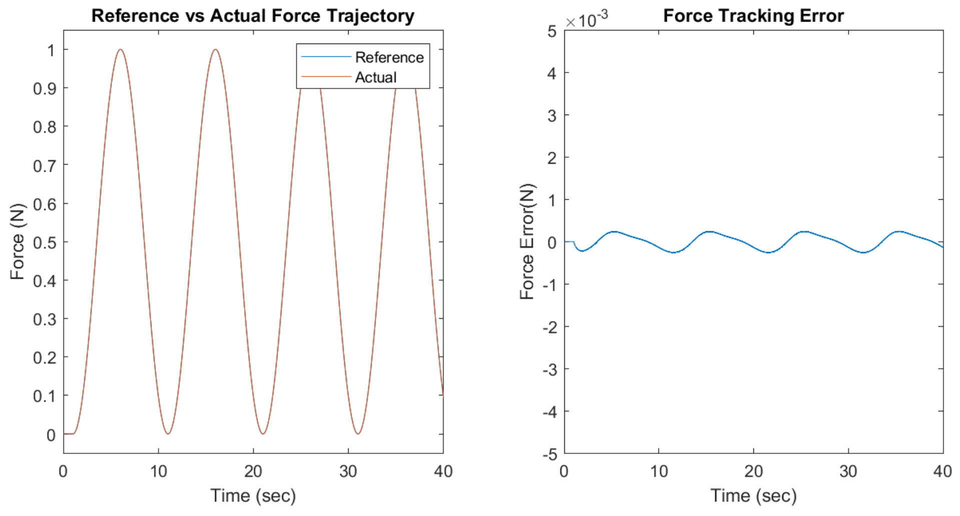
Table 6.5: RMSE, Overshoot, SSE and ST values for controllers



(a)

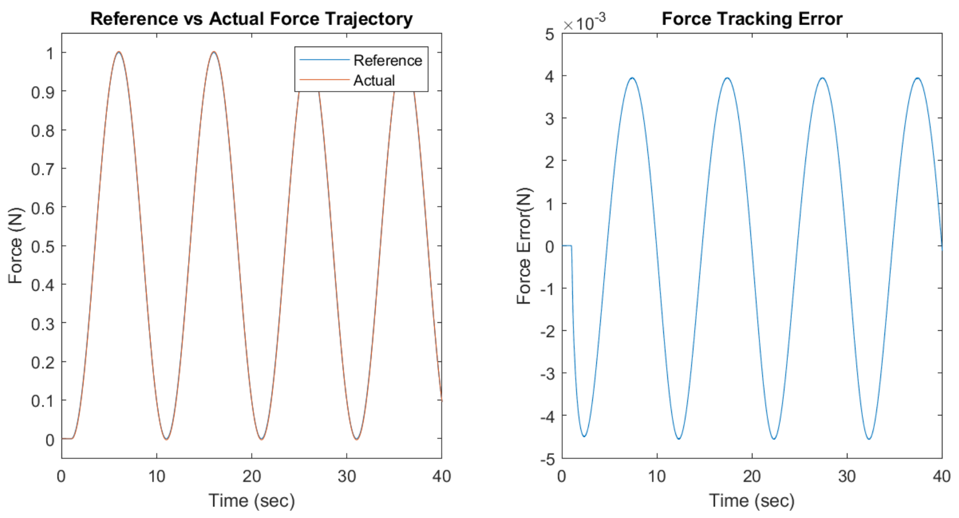


(b)

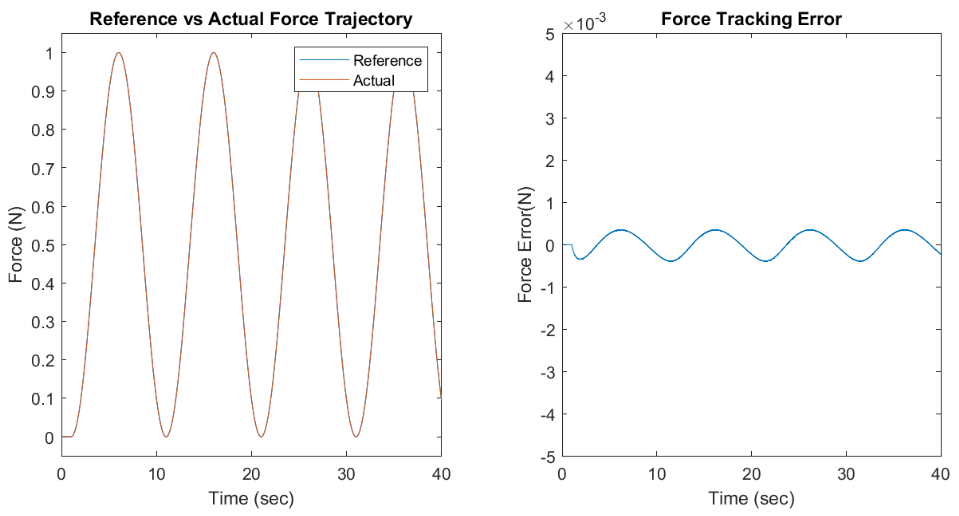


(c)

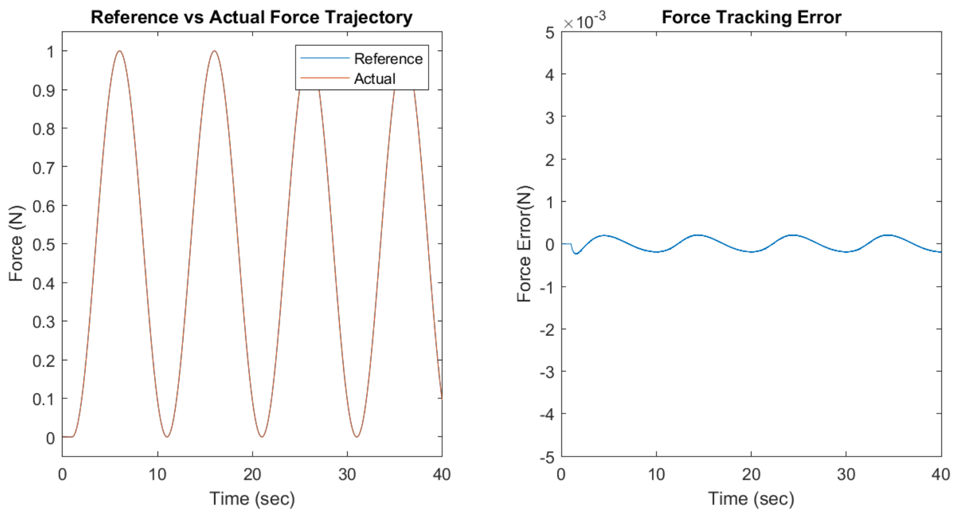
Figure 6.6: Force tracking graphs for Kelvin-Voight based adaptive controllers: (a) Adaptive control, (b) Adaptive control + P, (c) Adaptive control + PI



(a)



(b)

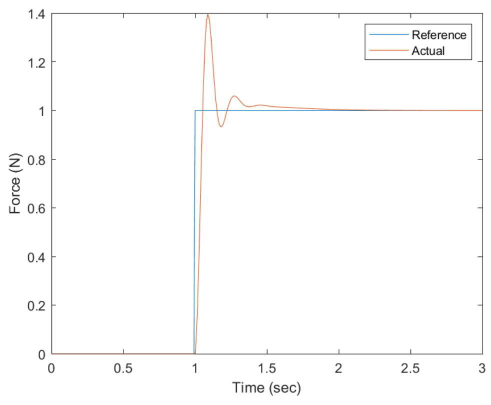


(c)

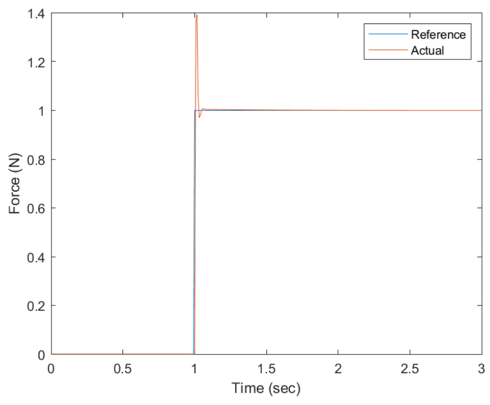
Figure 6.7: Force tracking graphs for Hunt-Crossley based neuro-adaptive controllers: (a) Neuro-adaptive control, (b) Neuro-adaptive control + P, (c) Neuro-adaptive control + PI

As can be seen, aside from the stand-alone adaptive and neuro-adaptive controllers, all controllers display a similar tracking performance, thus it is hard to make a comparison. On the other hand, their step responses draw a different picture. It goes without saying that SSE and ST values are important to step response analysis, but in this case, they are fairly close to each other and are not very helpful. However, it can be observed from the overshoot data that neuro-adaptive +P and +PI controllers outperform the rest.

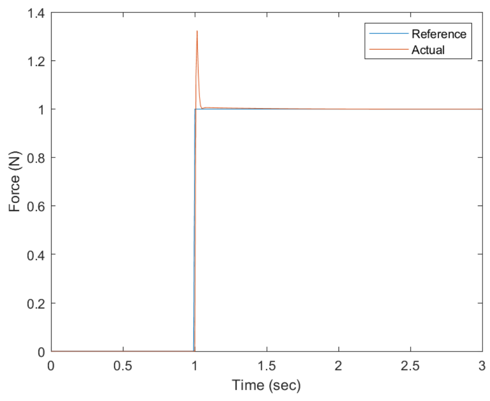
These results also show that, at least for this particular compliant environment, the linear Kelvin-Voigt contact model is a good representative of the environment regardless of its simplicity, and the controllers based on this model perform on par with the controllers based on the nonlinear Hunt-Crossley model. However, keep in mind that the sponge environment is not perfectly elastic, and it is not guaranteed that the Kelvin-Voigt based controllers will function the same when acting on a more elastic environment. Further testing with different objects is needed.



(a)

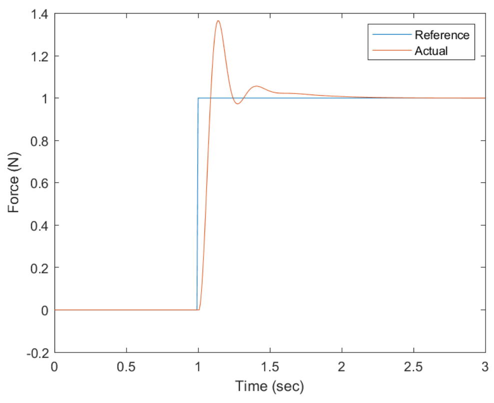


(b)

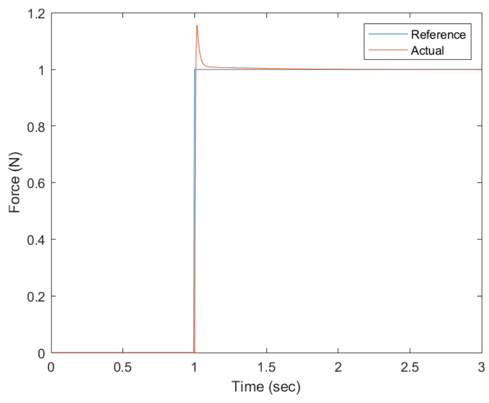


(c)

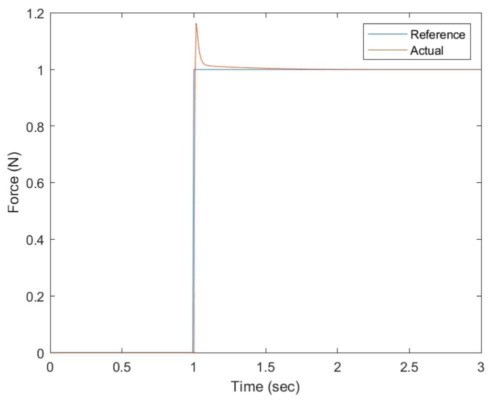
Figure 6.8: Step response graphs for controllers: (a) Adaptive control, (b) Adaptive control + P, (c) Adaptive control + PI



(a)



(b)



(c)

Figure 6.9: Step response graphs for controllers: (a) Neuro-adaptive control, (b) Neuro-adaptive control + P, (c) Neuro-adaptive control + PI

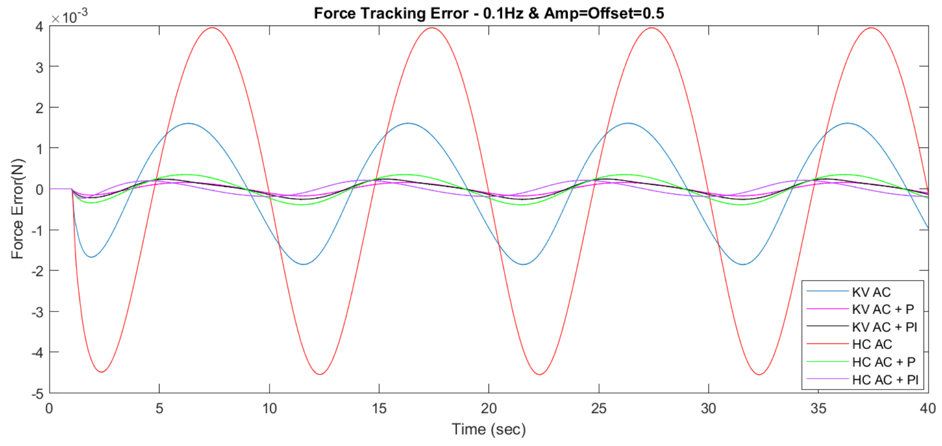
6.2.2 Effects of Input Magnitude and Frequency on Tracking Performance

Tuning is a task specific procedure, and in most cases, controllers have to be tuned from scratch for different objectives. As this can be quite time consuming, it is not very efficient. If the objective of a robot is, say, travel across a terrain with changing dynamical properties, then the control algorithm it runs either has to be tuned over and over again or designed in a way that it can adapt to changes.

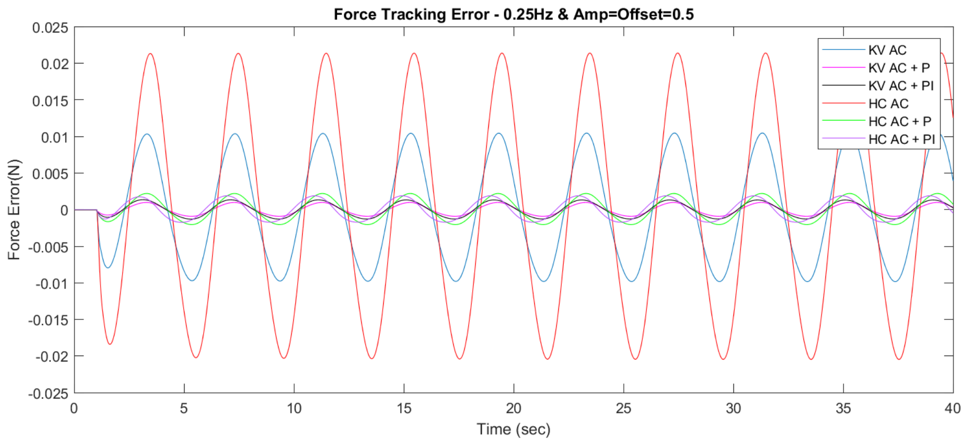
Previously, the tracking capabilities of the proposed controllers were examined using a sinusoidal (6.3) with constant frequency, amplitude and offset. Here, the goal is to investigate the controllers' ability to adapt to different trajectories without any tuning. Again, for this purpose, a sinusoidal signal (6.5) was used.

$$f_d(t) = A\sin(2\pi ft + \frac{\pi}{2}) + C \tag{6.5}$$

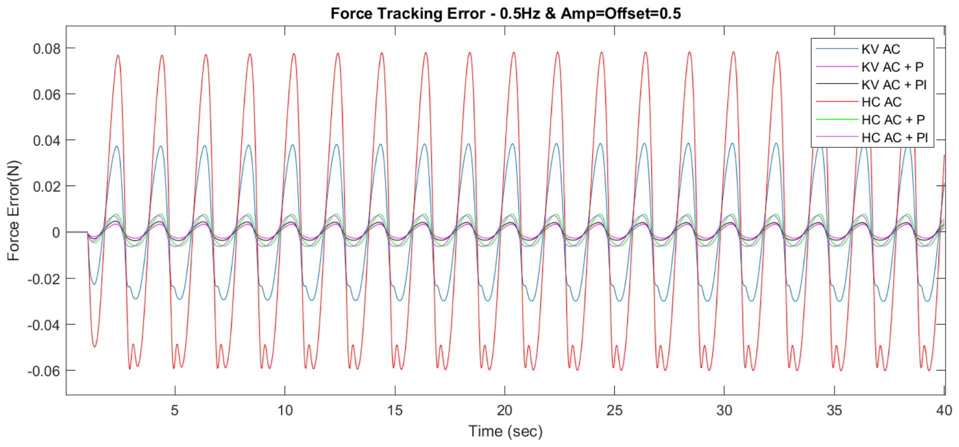
where A is the amplitude, C is the offset, and f is the frequency. Three different frequencies ranging from 0.1Hz to 0.5Hz were tested together with two different amplitude-offset pairs to assess the effects of input magnitude and frequency. The results are reported in figures 6.10-6.11, and tables 6.6-6.7.



(a)

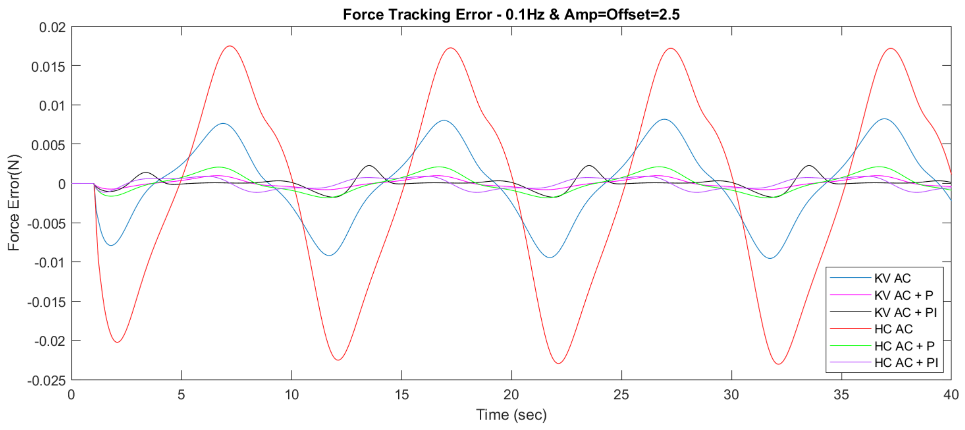


(b)

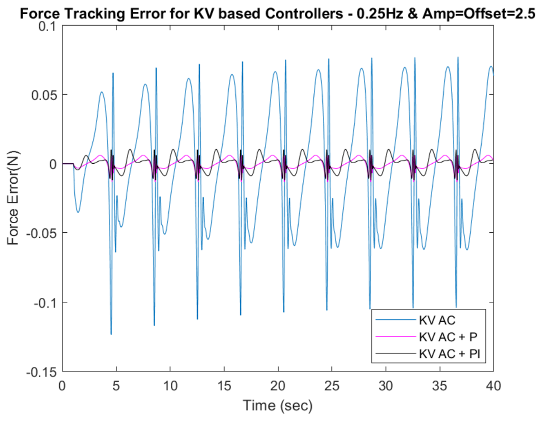


(c)

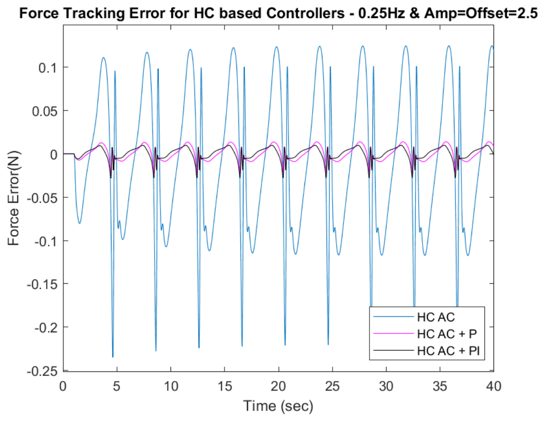
Figure 6.10: Effects of different frequencies on performance for Amplitude=Offset=0.5: (a) 0.1Hz, (b) 0.25Hz, (c) 0.5Hz



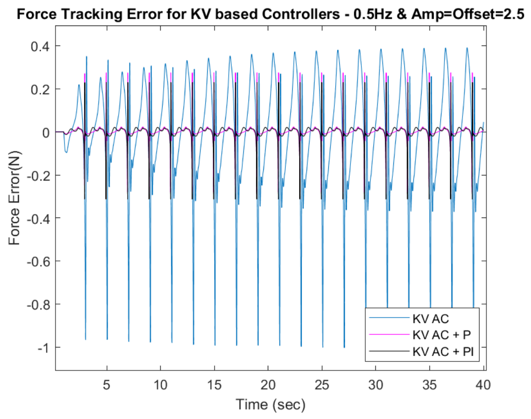
(a)



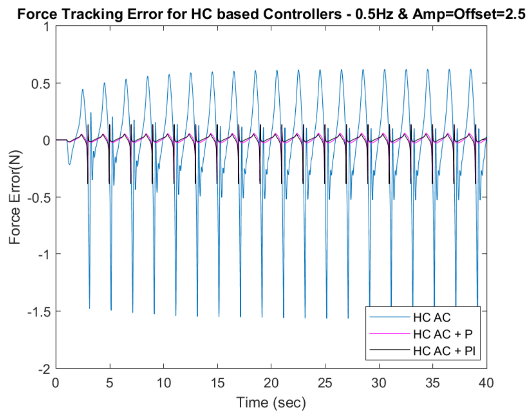
(b)



(c)



(d)



(e)

Figure 6.11: Effects of different frequencies on performance for Amplitude=Offset=2.5: (a) 0.1Hz, (b) 0.25Hz, (c) 0.5Hz

Controller	**f=0.1Hz**	**f=0.25Hz**	**f=0.5Hz**
Adaptive	0.0012	0.0070	0.0243
Adaptive + P	0.00011	0.00065	0.0021
Adaptive + PI	0.00016	0.00091	0.0028
Neuro-Adaptive	0.0030	0.0143	0.0496
Neuro-Adaptive + P	0.00025	0.0015	0.0048
Neuro-Adaptive + PI	0.00014	0.0013	0.0048

Table 6.6: Tracking RMSE values at different frequencies for Amp=Offset=0.5

Controller	**f=0.1Hz**	**f=0.25Hz**	**f=0.5Hz**
Adaptive	0.0051	0.0445	0.255
Adaptive + P	0.00054	0.0038	0.0372
Adaptive + PI	0.00080	0.0052	0.0403
Neuro-Adaptive	0.0116	0.0858	0.4316
Neuro-Adaptive + P	0.0012	0.0082	0.0501
Neuro-Adaptive + PI	0.00069	0.0076	0.0509

Table 6.7: Tracking RMSE values at different frequencies for Amp=Offset=2.5

These results indicate that all the controllers suffer from performance degradation to some degree when the input frequency and magnitude change. This is reasonable considering that the controllers were not tuned for every different input.

6.2.3 Parameter Convergence Analysis

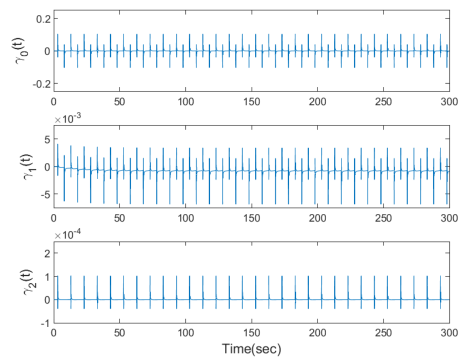
All the presented control laws achieve asymptotic force trajectory tracking through the adaptive estimation of a set of control parameters. The purpose of this part is to investigate the effects of different inputs on parameter convergence. Two main things to be examined are:

1. How a change in the frequency of a sinusoidal input affects the convergence rate and steady state parameter values
2. How different types of inputs influence parameter convergence

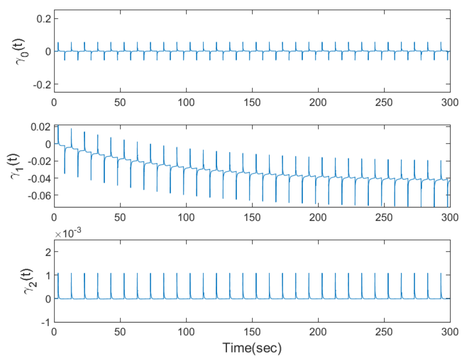
One thing to note is that, only the linear Kelvin-Voigt model based adaptive controllers were analyzed here. In order to perform the tests, the neural network approximated environment was replaced with the Kelvin-Voigt model itself.

$$f_c = k(x - x_c) + b\dot{x} \quad \text{where} \quad k = 100,\ b = 45,\ x_c = 0 \tag{6.6}$$

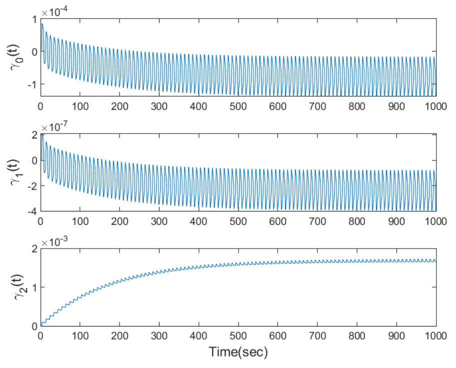
If this had not been done and the neural network was used instead, the estimated control parameters would not have physical correspondences due to the mismatch between the environment model and the model the controllers are based on. This is also the reason why the neuro-adaptive controllers were not examined, as the linear in the weights neural network simply approximates the non-linearity in the Hunt-Crossley model rather than being an exact representative of it.



(a)



(b)



(c)

Figure 6.12: Parameter convergence plots for controllers: (a) Adaptive control, (b) Adaptive control + P, (c) Adaptive control + PI

Effects of Input Frequency on Convergence

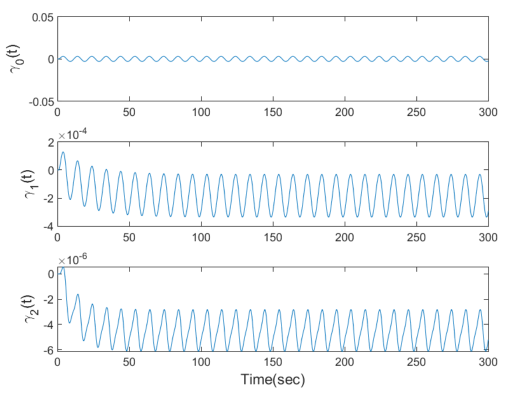
In order to analyze the effects of frequency change, the following trajectory was used:

$$f_d(t) = 0.5sin(2\pi ft + \frac{\pi}{2}) + 0.5 \tag{6.7}$$

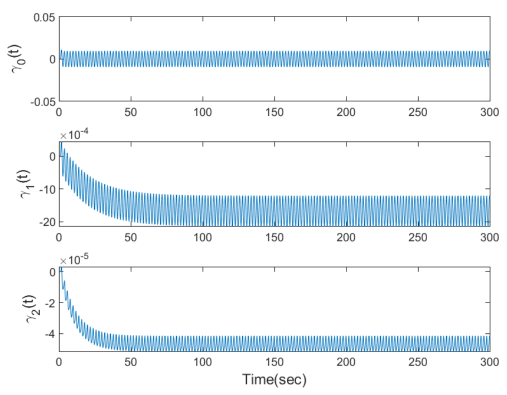
first with $f = 0.1$Hz, then with $f = 0.5$Hz. The parameter convergence plots are reported in figure (6.13). Although a significant difference in convergence rate cannot be observed, it can be seen that a higher frequency resulted in larger steady state values.

Effects of Input Type on Convergence

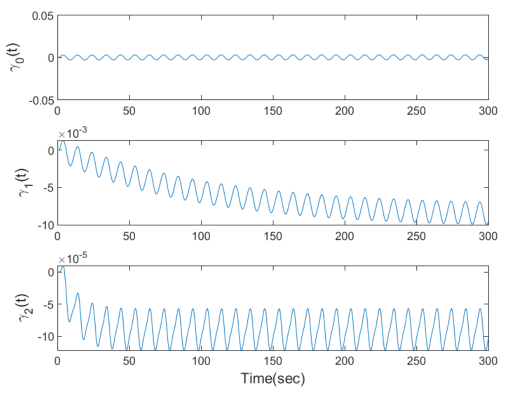
A pulse train trajectory was used to find how the parameters would be affected. Each pulse had an amplitude of 1 and a width of 10 seconds. The results are reported in figure (6.12). Using a pulse train trajectory made no observable change neither in convergence rate nor steady state values. It can also be observed that none of the parameters converge to their set values. This could be due to the reference signal not being persistently exciting enough to cause a difference.



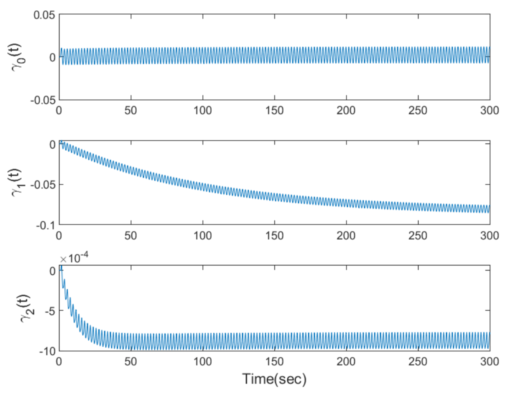
(a)



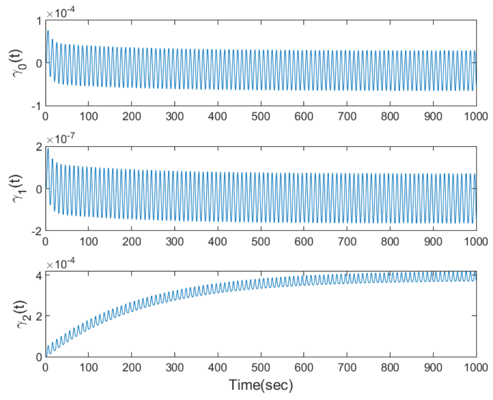
(b)



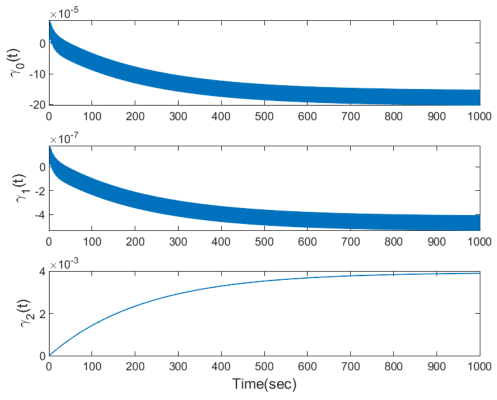
(c)



(d)



(e)



(f)

Figure 6.13: Parameter convergence plots for controllers: (a) Adaptive control at f=0.1Hz, (b) Adaptive control at f=0.1Hz, (c) Adaptive control + P at f=0.1Hz, (d) Adaptive control + P at f=0.5Hz, (e) Adaptive control + PI at f=0.1Hz, (f) Adaptive control + PI at f=0.5Hz

7
Conclusion and Future Work

In this paper two sets of force controllers were presented:

1. A set of linear Kelvin-Voigt dynamical model based adaptive controllers
2. A set of nonlinear Hunt-Crossley dynamical model based neuro adaptive controllers

In order to examine the performance of controllers, a neural network approximated testing environment was used. The controllers were evaluated based on their tracking capabilities and step responses. Sinusoidal trajectories with different magnitudes and frequencies were used to analyze how the controllers handle a change in the reference input without any further tuning. Finally, the effects of different inputs on control parameter convergence were investigated.

An interesting outcome was the fact that the pure neuro-adaptive controller was outperformed by every other controller which remained unexplained. Considering that the nonlinear Hunt-Crossley model is more physically sound, it was expected that the controller at least performed better than the pure Kelvin-Voigt model based adaptive controller.

Keep in mind that the controllers were designed for a force control problem in 1-D. In the future, the controllers could perhaps be redesigned for force control problems in 3-D as in their current state the controllers are limited in their applications. Also, in this thesis only a single type of testing environment was used (a sponge with semi-elastic properties) to examine the performance of controllers. Data could be collected from objects with different elasticities to create new neural network approximated environments such that the performance of controllers could be observed under different conditions.

Bibliography

[1] Roy, J., Whitcomb, L. (2002). Adaptive force control of position/velocity controlled robots: theory and experiment. IEEE Trans. Robotics Autom., 18, 121-137.

[2] Ioannou, P. A., Jing S. (1995). Robust adaptive control. Prentice-Hall, Inc., USA.

[3] Åström, Karl J., Wittenmark, B. . (1994). Adaptive Control (2nd. ed.). Addison-Wesley Longman Publishing Co., Inc., USA.

[4] Slotine, J.-J. E., Li, W. (1991). Applied nonlinear control. Englewood Cliffs, N.J: Prentice Hall.

[5] Ioannou, P. A., Fidan, B. (2006). Adaptive control tutorial. Philadelphia, Pa : Society for Industrial and Applied Mathematics, http://www.loc.gov/catdir/toc/fy0710/2006051213.html

[6] Kaufman H., Bar-Kana I., Sobel K., Bayard D. S. , and Neat G. W. (2012). Direct Adaptive Control Algorithms: Theory and Applications (1st. ed.). Springer Publishing Company, Incorporated.

[7] Kumpati S. Narendra, Valavani L. S. (1978). Direct and Indirect Adaptive Control, IFAC Proceedings Volumes, Volume 11, Issue 1, Pages 1981-1987, ISSN 1474-6670

[8] Landau, I., Lozano, R. and M'SAAD, Mohammed and Karimi, Alireza (2011). Adaptive control. Algorithms, analysis and applications. 2nd ed. 10.1007/978-0-85729-664-1.

[9] Landau, I. (1972). Model Reference Adaptive Systems—A Survey (MRAS)—What is Possible and Why? Journal of Dynamic Systems Measurement and Control-transactions of The Asme, 94, 119-132.

[10] Khalil, Hassan K. (1996) Nonlinear Systems. Upper Saddle River, NJ: Prentice Hall.

[11] Siciliano, B., Villani, L. (2000). Robot Force Control. 10.1007/978-1-4615-4431-9_1.

[12] Wang, W., Loh, R.N.K., Gu, E.Y. (1998), "Passive compliance versus active compliance in robot-based automated assembly systems", Industrial Robot, Vol. 25 No. 1, pp. 48-57. https://doi.org/10.1108/01439919810196964

[13] Villani L., De Schutter J. (2008) Force Control. In: Siciliano B., Khatib O. (eds) Springer Handbook of Robotics. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-30301-5_8

[14] Gilardi, G., & Sharf, I. (2002). "Literature survey of contact dynamics modelling. Mechanism and Machine Theory", 37, 1213-1239.

[15] Haddadi, A., Hashtrudi-Zaad, K. (2008). A New Method for Online Parameter Estimation of Hunt-Crossley Environment Dynamic Models, IEEE/RSJ International Conference on Intelligent Robots and Systems, 2008, pp. 981-986, doi: 10.1109/IROS.2008.4650575.

[16] Haddadi, A., Hashtrudi-Zaad, K. (2012). Real-Time Identification of Hunt–Crossley Dynamic Models of Contact Environments. IEEE Transactions on Robotics - TRob. 28. 555-566. 10.1109/TRO.2012.2183054.

[17] Diolaiti N., Melchiorri C., Stramigioli S. (2005). Contact impedance estimation for robotic systems, in IEEE Transactions on Robotics, vol. 21, no. 5, pp. 925-935, doi: 10.1109/TRO.2005.852261.

[18] Son H. I., Bhattacharjee T., Lee D. Y. (2010). Estimation of environmental force for the haptic interface of robotic surgery, in Int J Med Robot.6(2):221-30. doi: 10.1002/rcs.311. PMID: 20506442.

[19] Yamamoto T., Vagvolgyi B., Balaji K., Whitcomb L. L., Okamura A. M. (2009) "Tissue property estimation and graphical display for teleoperated robot-assisted surgery," 2009 IEEE International Conference on Robotics and Automation, pp. 4239-4245, doi: 10.1109/ROBOT.2009.5152674.

[20] Singh S. K., Popa D. O. (1995) "An analysis of some fundamental problems in adaptive control of force and impedance behavior: theory and experiments," in IEEE Transactions on Robotics and Automation, vol. 11, no. 6, pp. 912-921, doi: 10.1109/70.478439.

[21] Butterworth Filter Design. electronics-tutorials.ws. Retrieved from https://www.electronics-tutorials.ws/filter/filter_8.html

[22] Vidyasagar M. (1993). Nonlinear Systems Analysis, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall.

[23] Riley, W.F., Sturges, L.D., (1996). Engineering Mechanics Dynamics, John Wiley and Sons, New York.

[24] Barkan, P. (1974). Impact design, in: Mechanical Design and Systems Handbook, Section 31, McGraw-Hill, New York

[25] Brach, R.M. (1991). Mechanical Impact Dynamics: Rigid Body Collisions, John Wiley and Sons, New York.

[26] Brach, R.M.(1998). Formulation of rigid body impact problems using generalized coefficients, International Journal of Engineering Science 36 (1) 61–71.

[27] Kim, S.W. (1999). Contact Dynamics and Force Control of Flexible Multi-Body Systems, Ph.D. Thesis, Department of Mechanical Engineering, McGill University, Montreal.

[28] Goldsmith, W. (1960). Impact: The Theory and Physical Behavior of Colliding Solids, Edward Arnold Publishers Ltd, London.

[29] Wang, Y.T., Kumar, V. (1994). Simulation of mechanical systems with multiple frictional contacts, Journal of Mechanical Design 116 571–580.

[30] Hunt, K.H., Crossley, F.R.E. (1975). Coefficient of restitution interpreted as damping in vibroimpact, Journal of Applied Mechanics 42, Series E 440–445.

[31] Marhefka, D.W., Orin, D.E. (1999). A compliant contact model with nonlinear damping for simulation of robotic systems, IEEE Transactions on Systems, man, and Cybernetics—Part A: Systems and Humans 29 (6) 566–572.

[32] Goodfellow I., Bengio Y., Courville A. (2016). Deep Learning, MIT Press.Retrieved from http://www.deeplearningbook.org

[33] What is the Definition of Machine Learning? | Expert.ai. (2021). Retrieved from https://www.expert.ai/blog/machine-learning-definition/

[34] 1.17. Neural network models (supervised) — scikit-learn 0.24.2 documentation. (2021). Retrieved from https://scikit-learn.org/stable/modules/neural_networks_supervised.html

[35] A Beginner's Guide to Multilayer Perceptrons (MLP). (2021). Retrieved from https://wiki.pathmind.com/multilayer-perceptron

[36] 4.7. Forward Propagation, Backward Propagation, and Computational Graphs — Dive into Deep Learning 0.17.0 documentation. (2021). Retrieved 29 August 2021, from https://d2l.ai/chapter_multilayer-perceptrons/backprop.html

[37] Brownlee, J. (2021). How to Choose an Activation Function for Deep Learning. Retrieved 29 August 2021, from https://machinelearningmastery.com/choose-

an-activation-function-for-deep-learning/

[38] Introduction to Loss Functions. (2021). Retrieved 29 August 2021, from https://algorithmia.com/blog/introduction-to-loss-functions

[39] Mean Squared Error (2021). Retrieved from https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/mean-squared-error

[40] Serengil, S. (2021). Softsign as a Neural Networks Activation Function. Retrieved from https://sefiks.com/2017/11/10/softsign-as-a-neural-networks-activation-function/

[41] Brownlee, J. (2021). A Gentle Introduction to the Rectified Linear Unit (ReLU). Retrieved from https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

[42] Darken, C., Chang, J., Moody, J.(1992). Learning rate schedules for faster stochastic gradient search. Neural Networks for Signal Processing II Proceedings of the 1992 IEEE Workshop, (September):1–11.

[43] Robbins, H., Monro, S. (1951). A Stochastic Approximation Method. The Annals of Mathematical Statistics, 22(3):400–407.

[44] Ruder, S. (2017). An overview of gradient descent optimization algorithms.

[45] Katanforoosh, K. et al. (2019). Parameter optimization in neural networks, deeplearning.ai.

[46] Alfonsos, K. (2020). Data-driven modeling for robotic manipulation of soft and deformable objects, https://hdl.handle.net/20.500.12380/301291.

[47] Kingma P. D., Ba J. (2017). Adam: A Method for Stochastic Optimization, arXiv:1412.6980 .

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

[49] Dauphin, Y., Vries H. Chung J., Bengio Y. (2015). RMSProp and equilibrated adaptive learning rates for non-convex optimization. arXiv. 35.

[50] Karayiannidis Y., Rovithakis G., Doulgeri Z. (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.