Supervised Neural Network Learning for Improved Passivity in Robot Interaction Control Applications

by

Antonio Griego

B.S., Computer Science, University of New Mexico, 2018

THESIS

Due to serious health issues Antonio graduated without submitting this thesis. This late draft is posted for archival purposes - Matthew Fricke.

Master of Science
Computer Science

The University of New Mexico

Albuquerque, New Mexico

September, 2025

Dedication

To my parents, Dino and Ruth, for their overwhelming love and support.

Acknowledgments

I would like to thank Professor Melanie Moses for giving me the opportunity as an undergraduate that eventually opened every door on my path towards graduate school.

I would also like to thank Research Assistant Professor Matthew Fricke, whose mentorship and guidance gave me significant real world and research experience as a computer scientist before earning my degree.

And finally, I would like to thank Dr. Jonathon Slightam, who taught me mechanical engineering and provided the inspiration and guidance for this thesis.

Supervised Neural Network Learning for Improved Passivity in Robot Interaction Control Applications

by

Antonio Griego

B.S., Computer Science, University of New Mexico, 2018

M.S., Computer Science, University of New Mexico, 2025

Abstract

If a robot controller is able to adapt to the challenges of an unstructured environment, it will have a major advantage over previous systems that require precise descriptions of the environment in order to function. One control system could be used for an array of different use cases without having the need to design and test a controller for each specific scenario. Designing such a controller is a difficult challenge. The fine tuning of robot impedance control parameters can be a laborious and time-consuming process. It is very desirable to develop methods for streamlining the tuning of impedance control parameters through simulation that insures stability, desired tracking, and interaction characteristics. Leveraging neural network machine learning techniques utilizing data collected from simulation, the goal is to develop a methodology and technique to find an optimal solution for a given impedance control setup. In addition, by focusing on solving a specific impedance control problem it may lead to insights in how these techniques can be broadened to other applications.

Contents

List of Figures

List of Tables

Glossary

adaptive impedance/force control

: the impedance/force control law is adapted based on measured feedback

control law
: a mathematical formula used by the controller to determine the outout "u" that is sent to the plant

damping
: a decrease in the amplitude of an oscillation as the result of energy being drained from the system to overcome frictional or other resistive forces

force
: a push or pull upon an object resulting from the object's interaction with another object

force control
: output torque is controlled to match the desired force applied by the end-effector on an external object

impedance control
: A distinction between impedance control and the more conventional approaches to manipulator control is that the controller attempts to implement a dynamic relation between manipulator variables such as end-point position and force rather than just control these variables alone. This change in perspective results in a simplification of several control problems. [1]

instantaneous mechanical work the energy transferred to an object via the application of force along a displacement; the product of force and displacement $dW = (F)(dX)$

iterative learning control scheme A method of tracking control for systems that work in a repetitive mode.

Jacobian matrix which provides the relation between joint velocities and end effector velocities of a robot manipulator

kinematics the branch of mechanics concerned with the motion of objects without reference to the forces which cause the motion; the features or properties of motion in an object

inverse kinematics is the mathematical process of calculating the variable joint parameters needed to place the end of a kinematic chain (such as a robot manipulator) in a given position and orientation

manipulation mechanical interaction with the object(s) being moved or interacted with

motion control a controller which calculates and controls mechanical trajectories, treating the robot and end effector as an isolated system in the simplest form

stiffness the extent to which an object resists deformation in response to an applied force

Chapter 1

Introduction

The primary motivation for this thesis is to develop a technique to guarantee stability for robot manipulation in variable environments. This will be achieved using impedance control and machine learning by dynamically altering impedance control parameters in response to sensed environmental conditions.

Robot interactions in well structured and known environments is well studied [2][3]. For example, factory robots have been successfully deployed in numerous manufacturing roles. All environmental variables are controlled in order to produce an efficient assembly line. A robotic arm assembling a component or product is programmed to know precisely where it can and cannot move and does not have to adapt to any surprising or unexpected circumstances.

However, it is desirable to develop strategies and robust algorithms where robots can be deployed to perform manipulation tasks in the field. This will open the door to a host of new and innovative applications. Mobile robots can be deployed in emergency situations to search for survivors or other resources. Bomb squad robots may be designed to be more autonomous and versatile requiring little or perhaps no human manipulation to perform vital tasks. Such applications, as well as many more,

become possible when a robotic system is able to adapt and adjust to an unknown environment.

In this chapter, the main problem is introduced as well as a discussion of the core concepts. This chapter will also present an outline detailing how the rest of the thesis will be organized.

1.1 Technical Challenges with Robot Interaction

Stable mechanical manipulation is a core problem that has been the center of a vast body of research and development [4]. Of particular interest to this work is the ability of a robot manipulator to maintain stability while performing a manipulation task in uncertain environments.

Early manipulators were directly controlled by human operators. For example, Goertz developed manipulators to facilitate interactions with dangerous radioactive material for lab work [4]. These systems maintained stability due to the direct control of the human operators coupled with sensors in the manipulators that provided force reflection.

Eventually, control systems were developed for robotic arms that made use of computers in order to enable automatic control without human intervention. Many different types of control schemes exist in the literature including position control, force control, impedance control, hybrid position-force control, and others.

Of primary concern with control schemes such as position control and force control is that the environment being interacted with needs to be precisely defined and known in advance of any manipulation task. It is possible to meet this requirement with some industrial applications in predictable and controlled environments.

Passivity analysis can be used to determine if a system interacting with an environment is stable. Passivity analysis focuses on interaction through the perspective of a robot manipulator or end effector [5]. If a system can be guaranteed to be passive, then it may interact with a wide range of possible environments with little to no stability problems.

On the other hand, Non-passive systems may produce force feedback instabilities when coupled with some environments. Due to hardware design or any number of other possible issues, it may be difficult to ensure passivity in a robot. It is a challenge then to design algorithms and control systems that can dynamically adjust an impedance controller to robustly adapt to changes in environment and interaction. By using a neural network to train on a vast variety of potential impedance control parameters and environmental feedback, it may be possible to generalize a robust and adaptive response to fine tuning an impedance controller on the fly.

1.1.1 Impedance Control

Impedance control is an approach for controlling the dynamic interaction between a manipulator and its environment, and it overcomes many of the limitations of force control or position control alone [1]. In particular, previous successful robot control schemes have been based on position control which treats the robot as an isolated system. This method is undesirable for capturing the dynamic interaction between a robot and an uncertain environment.

The fine tuning of robot impedance control parameters remains a difficult problem and a laborious and time-consuming process. In particular, applications where a robot needs to interact with an uncertain, dynamic environment are difficult to solve because impedance control requires an accurate model of manipulator and environment dynamics. Iterative methods exist for learning or selecting impedance control

parameters [6][7]. These methods are effective for situations where the environment to be interacted with is well defined and consistent, but these methods may not effectively handle situations where the environment to be interacted with may be dynamically changing over time.

This work presents a method for tuning robot impedance control parameters quickly by using a supervised machine learning method to develop a trained neural network model.

This model is trained on a vast array of various impedance control and environment parameters in order to generate a robust response. This neural network accepts as input the current impedance control parameters and measured environment dynamics and predicts updated impedance control parameters to ensure stability of the system. This type of system falls under the category of function fitting or regression problems. In particular, this system is trying to solve a multi-variable input problem with a single output. By using machine learning to generalize the impedance controller response, this system may not be limited in scope to any one particular task or environment interaction, and may result in a robust impedance controller that can be used in mobile robots that need to perform tasks in uncertain environments.

A bespoke controller optimized for a precise and well defined problem may outperform this controller in its task of expertise. However, unlike the proposed system it would be unable to adapt to changes that may be seen when deploying autonomous robots in varying environments.

1.1.2 Neural Network

The Neural Networks used in this work were developed using the Matlab Deep Learning Toolbox [8]. Three neural network architectures were used and their performances compared with each other:

1. FITNET – function fitting neural network

   This neural network is a standard regression/function fitting neural network architecture. The size of the hidden layers and the training functions may be chosen from a suite of options.

2. DLNETWORK – custom deep learning network

   This neural network is a deep learning network architecture available in Matlab. Like Fitnet, many parameters may be adjusted

3. GRNN – generalized regression neural network

   This neural network is a function fitting neural network with one hidden layer. It uses radial basis functions in order to approximate the function fitting solution.

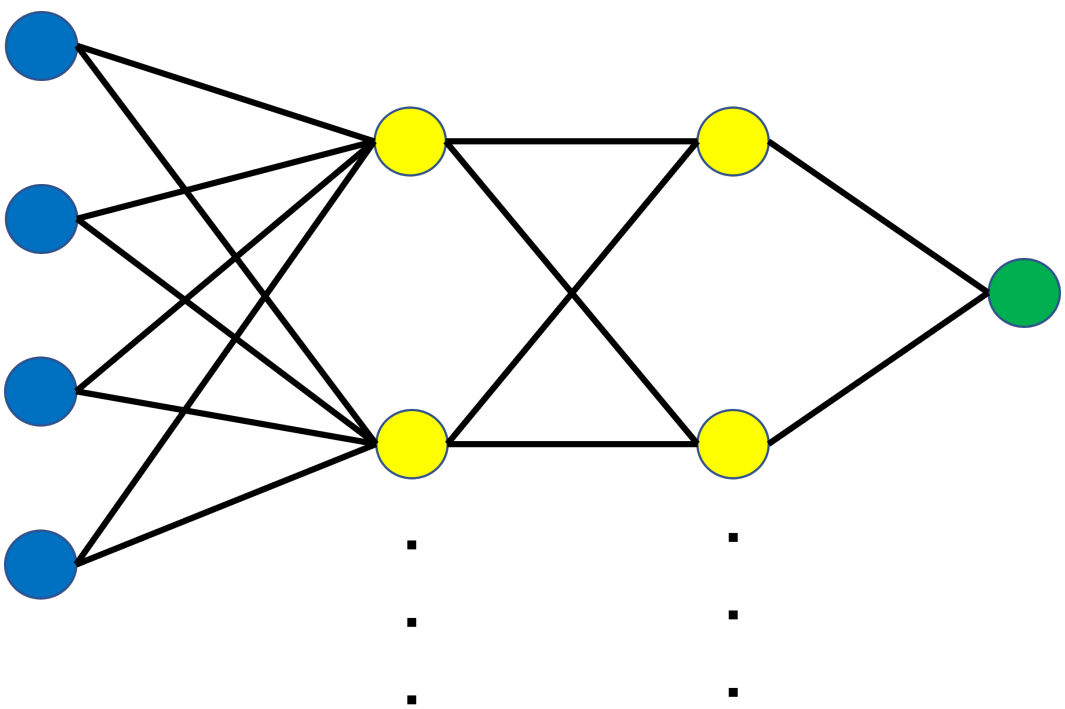

Figure 1.1: NN Example with 2 hidden layers

Other work has also been done in implementing neural networks with impedance control [9][10][11][12]. These works have focused on using machine learning to deal with specific tasks and limited or well defined environments, and they have established that the use of machine learning with impedance control is a viable and effective tool.

It is the goal of this thesis to build upon these results by applying machine learning tools in a broader and more ambitious scope with a simple exemplar problem. Once a proven foundation is designed, this work can be extended to include more complex systems.

1.1.3 Stability

A well established measure of stability in dynamic systems is Nyquist stability analysis. It will be a primary tool used in this work at many stages of development from generating data sets to verifying results.

The Nyquist plot of a sinusoidal transfer function $G(j\omega)$ is a plot of the magnitude of $G(j\omega)$ versus the phase angle of $G(j\omega)$ in polar coordinates as ω is varied from zero to infinity [13]. The Nyquist plot is often called the polar plot. The polar plot is the locus of vectors $|G(j\omega)|\angle G(j\omega)$ as ω is varied from zero to infinity.

The advantage of using a Nyquist plot is that it depicts the frequency response characteristics of a system over the entire frequency range in a single plot [13].

It is desirable to ensure stability to help guarantee that any motion planning or interaction activity the robot performs will execute successfully and to protect hardware from damage.

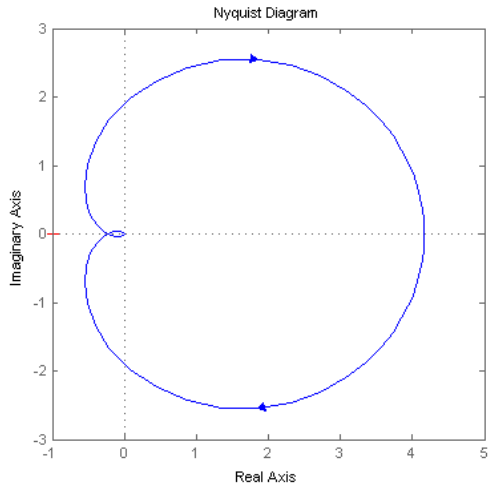

Figure 1.2: A Nyquist Plot.

1.2 Technical Aims and Methodology

The current literature has detailed useful and powerful techniques for robot control such as impedance control. Such tools have been used to great effect to produce control schemes for robots in known and predictable environments. Machine learning provides additional tools that may prove useful to use in conjunction with these techniques in order to expand the ability of robots to operate in unknown environments in a robust way.

Manipulation in sanitized environments is well studied; this work intends move forward to study methods that could be able to deploy autonomous robots in the field, perhaps in uncertain or unknown environments and to interact with and manipulate unknown objects.

To begin the discussion of methodology and its orchestration, background research and information will be discussed in Chapter 2. This will include a discussion

of control methods and an analysis of the decision to use impedance control in this work. Several forms of control will be compared including force, position, impedance, and hybrid control. There will also be discussion on previous work in learning methods to include iterative techniques and other neural network and machine learning techniques.

A simple exemplar robot model is designed and implemented in Matlab. This model will be a 1 DOF robot. This model will be the basis for simulation and machine learning training. This model will be presented in Chapter 3.

In addition to this model will be an impedance controller that is designed to work with the robot model and will have several parameters presented in Chapter 4. These parameters can be used as input and output for a function fitting machine learning algorithm. A neural network designed to work with these parameters can, given a set of parameters as input, output a desired damping value in order to make the robot system stable.

Chapter 5 will present the coupled stability analysis method. This is the method used for determining the stability of the coupled system of the robot model to the environment. A proposed approach to guaranteeing stability will be presented.

In order to generate a neural network that can perform this task, it must be designed well and trained on a suitable data set. Chapter 6 will review the neural network frameworks used in the machine learning experiments with Matlab and the neural network toolbox. A suitable robot model is first designed in Matlab. The physical parameters of this simulated robot, along with the parameters defining an impedance controller, will serve as the input to a neural network. The expected output will be a damping value for the impedance controller. Metaphorically, this damping value will serve like the brakes of a motor vehicle, being applied or released as necessary to achieve the desired movement. This type of machine learning problem

is a function fitting or a regression problem.

Experiments will be done with a few types of neural networks so that they can be tested against each other as well as with varying hidden layer sizes and other parameters in order to experimentally determine the best performing network for this type of problem. Chapter 7 will present the results of the work with some discussion about the networks ability to generalize and provide stable impedance control parameters with untrained input.

Chapter 8 will discuss these results in further detail and Chapter 9 will present the conclusions drawn from this work and whether the initial findings for this work is promising. Also, possible future work and extensions will be discussed.

Chapter 2

Background

Robot systems and control is a well established and long running area of research and development. Initial development began with human operated robot arms with force reflection and eventually shifted into computer controlled robots [4].

Many of the early successful applications for industrial robotics were restricted to the case where the robot could be treated as an isolated system with no environment interaction. Examples of such applications included spray-painting and welding. However, there was also a need for a more robust control scheme that could handle the case where a robot needs to interact with the environment in order to handle such tasks as drilling, bending, grinding, etc. Hogan [1] developed impedance control to solve these types of problems. However, the development of control schemes for dealing with undefined or unpredictable environments is an open area of research and development.

Machine learning techniques such as neural networks are being used increasingly in robot control applications. The promised advantage of using machine learning in the context of robot controls is that it will allow better generalization than previous iterative techniques [12]. Previous work integrating neural networks into impedance

control schemes achieved success in lowering uncertainty and providing stability in dealing with more specific types of environments and tasks such as deburring [12] and trajectory planning [11][10][9] when coupled to static environments.

Given an impedance controller with adjustable parameters, robot actuator dynamics are known, and that the environment being interacted with can be measured, this work proposes and tests a general and robust strategy of updating impedance control parameters to ensure stability.

2.1 Control Methods

2.1.1 Position Control

2.1.2 Force Control

2.1.3 Hybrid Position/Force Control

2.1.4 Impedance Control

2.1.5 Force Feedback

2.1.6 Interaction with Environment

2.2 Non-Machine Learning Techniques

Previous work has been established using non-machine learning methods for learning impedance control parameters. Kim [14] proposed a recursive least-square filter-

based episodic natural actor-critic algorithm in order to find optimal impedance control parameters. Arimoto [15] presents a physical interpretation of practice-based learning that steadily learns a desired task by monotonously increasing the grade of impedance matching pertaining to the dynamics of the robot task with controller dynamics. Cheah [16] introduces a method for learning impedance control for robot manipulators. Unlike other approaches, Cheah's method is implemented without the need to switch the learning controller from non contact to and from contact tasks. Wang [7] presents an iterative learning control law for the impedance control of robotic manipulators. This method does not require a reference trajectory for training and instead the performance is determined by a target impedance.

This foundational work demonstrates the effectiveness of impedance control and various novel techniques for learning impedance control parameters. However, most of these approaches are somewhat brittle in the sense that they focus on teaching one specific task or interaction to the controller and does not generalize to new environments without completely retraining the controller for each new scenario. It is the hope that this work can build off the ideas of these methods and develop a method that can generalize a method to adapt impedance control parameters for stability no matter what environment an actuator may interact with.

2.3 Other Machine Learning Approaches

In addition to traditional techniques described previously, there has also been a lot of work in using machine learning techniques to learn impedance control parameters. Tsuji [10] developed a method to use an array of neural networks to regulate the impedance parameters of a manipulator's end-effector while identifying environmental characteristics through on-line learning. Jung [11] proposes a method using neural networks to compensate for uncertainties in the robot model. A novel error

signal is proposed for the neural network training. Katic [12] presents an application of connectionist structures for fast and robust online learning of internal robot dynamic relations used as part of impedance control strategies in the case of robot contact tasks. Cohen [9] presents an evaluation of the associative search network (ASN) learning scheme which is a stochastic scheme that uses a single scalar value as a measure of the system performance.

Previous work using machine learning methods has provided solid evidence that neural networks can be applied effectively to solve impedance control parameter learning problems. This work either focuses on using neural networks to train an actuator on learning specific tasks or focuses on using online learning methods. In contrast, this work presents an offline learning method that hopes to generalize its adjustment of impedance control parameters to any environment or task. The goal is not necessarily to teach an impedance controller any specific task, but to give it a robust response to any environment to ensure passivity of the system.

Chapter 3

System Modeling

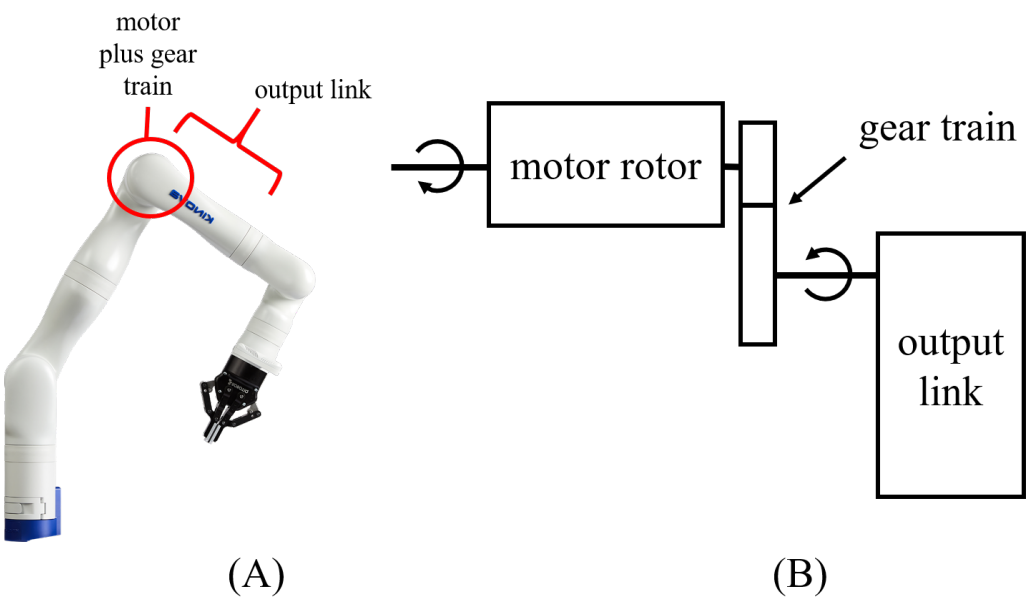

Figure 3.1: Model diagram for a 1-DOF Robot Arm Actuator

This section presents a robot actuator model designed in MATLAB/Simulink in order to test the proposed stability scheme. The actuator model is a 1-DOF robot

drive. The actuator is split into two parts that must be designed and simulated. The system will be represented with a mathematical model [13].

3.1 Drive Motor and Gear Train Dynamic Model

The first stage in the design of this model is to develop free body diagrams and model diagrams for all of the components. Figure 3.2 shows the free body diagram of part 1 of the actuator.

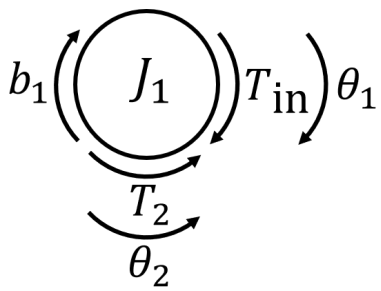

Figure 3.2: Free Body Diagram for Drive Motor and Gear Train

Where J_1 is the moment of inertia, which is defined as $\frac{\text{change in torque}}{\text{change in angular acceleration}} = \frac{Nm}{\text{rad/sec}^2}$. b_1 is the damping constant for the shaft connecting J_1 to the actuator motor. T_{in} is the input torque supplied by the actuator motor. T_2 is the load torque from actuator part 2. θ_1 is the angular position (in radians) of the shaft connecting actuator part 1 to the motor. θ_2 is the angular position (in radians) of the shaft connecting the gear train to actuator part 2.

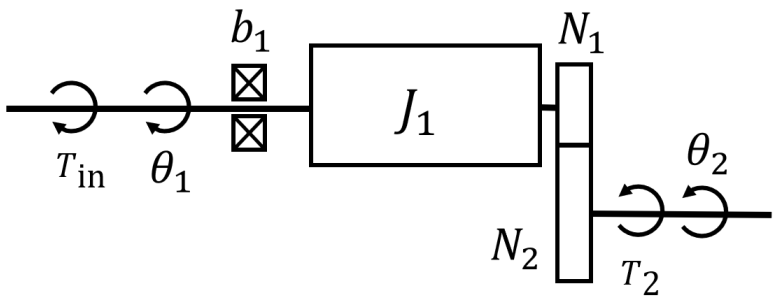

Figure 3.3: Model Diagram for Drive Motor and Gear Train

Next, the model diagram is developed (see Figure 3.3), preserving all of the relationships defined in the free body diagram with the addition of two new components. N_1 is the number of teeth in the gear attached to J_1. N_2 is the number of teeth in the gear attached to the shaft connected to actuator part 2.

It is assumed that the gear train is perfect with no slippage or other complicating factors. With a schematic of actuator part 1 complete, we can proceed to use Newton's Second Law [13] to derive the equation of motion that will represent the mathematical model of this part of the system.

$$J_1\ddot{\theta}_1 = T_{\text{in}} - \left(\frac{N_1}{N_2}\right) T_2 - b_1\dot{\theta}_1 \tag{3.1}$$

In Equation 3.1, the torque on the right hand side of the equation is defined to be the input torque from the motor minus the load torque of the shaft scaled by the gear train ratio minus the effects of damping in the system. In order to facilitate implementation of our model in Simulink, it is helpful to make some substitutions in

order to solve Equation 3.1 in terms of θ_2.

$$\theta_1 = \left(\frac{N_2}{N_1}\right)\theta_2$$

therefore:

$$\begin{aligned} J_1\ddot{\theta}_1 &= \left(\frac{N_2}{N_1}\right) J_1\ddot{\theta}_2 \\ b_1\dot{\theta}_1 &= \left(\frac{N_2}{N_1}\right) b_1\dot{\theta}_2 \end{aligned} \tag{3.2}$$

and thus, equation 3.1 becomes:

$$\left(\frac{N_2}{N_1}\right) J_1\ddot{\theta}_2 = T_{\text{in}} - \left(\frac{N_1}{N_2}\right) T_2 - \left(\frac{N_2}{N_1}\right) b_1\dot{\theta}_2$$

finally, we simplify and solve in terms of $\ddot{\theta}_2$

$$\ddot{\theta}_2 = \left(\frac{1}{J_1}\right)\left(\frac{N_1}{N_2}\right) T_{\text{in}} - \left(\frac{1}{J_1}\right)\left(\frac{N_1}{N_2}\right)^2 T_2 - \left(\frac{1}{J_1}\right) b_1\dot{\theta}_2$$

Equation 3.2 is the equation of motion for actuator part 1. Using these same methods, we can now derive the equation of motion (EOM) for actuator part 2.

3.2 Output Link Model

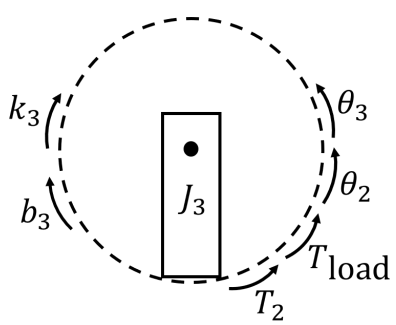

Figure 3.4: Free Body Diagram for Output Link Model

Where J_3 is the moment of inertia. b_3 is the damping constant for the shaft between the gear train and J_3. k_3 is the spring/stiffness constant for the shaft between the gear train and J_3. T_2 is the load torque from actuator part 2. T_{load} is the load torque from the environment. θ_2 is the angular position (in radians) of the shaft connecting the gear train to actuator part 2. θ_3 is the angular position (in radians) of the shaft connecting to J_3.

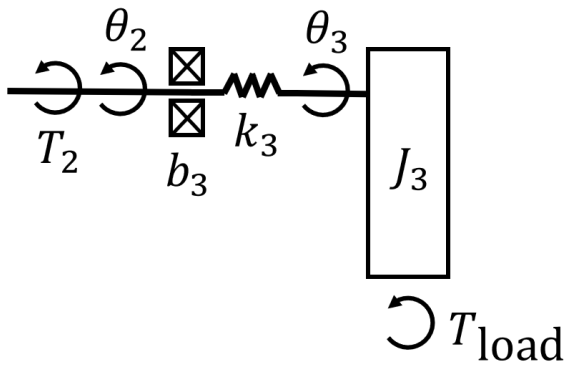

Figure 3.5: Model Diagram for Output Link Model

As we can see from the model diagram in Figure 3.5, the angles θ_2 and θ_3 represent angular positions at opposite ends of the same shaft. Due to the physical properties of damping and stiffness represented in this model, it is not assumed that these angles are identical and must be taken into account. We can thus derive the equations of motion for actuator part 2 as follows:

$$
\begin{aligned}
&J_3\ddot{\theta}_3 = T_{\text{load}} - b_3\left(\dot{\theta}_3 - \dot{\theta}_2\right) - k_3\left(\theta_3 - \theta_2\right) \\
&\text{solve in terms of acceleration:} \\
&\ddot{\theta}_3 = \left(\frac{1}{J_3}\right)T_{\text{load}} - \left(\frac{b_3}{J_3}\right)\left(\dot{\theta}_3 - \dot{\theta}_2\right) - \left(\frac{k_3}{J_3}\right)\left(\theta_3 - \theta_2\right)
\end{aligned}
\tag{3.3}
$$

3.3 System Model

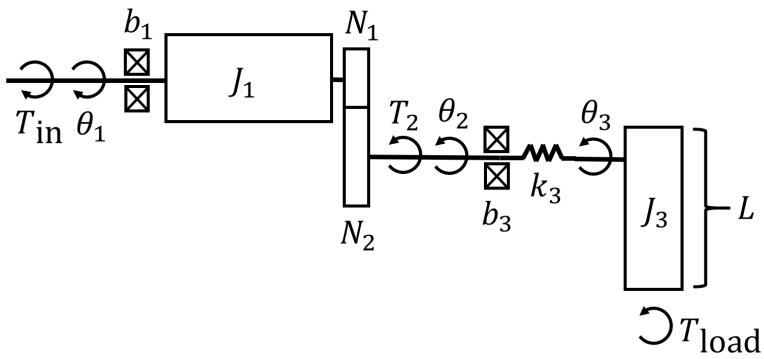

Figure 3.6: Model Diagram for a 1-DOF Robot Arm Actuator

With the equations of motion established for all components of the actuator, we can now combine everything for a complete model representation (see Figure 3.6).

Recall that:

$$\begin{aligned}\ddot{\theta}_2 &= \left(\frac{1}{J_1}\right)\left(\frac{N_1}{N_2}\right)T_{\text{in}} - \left(\frac{1}{J_1}\right)\left(\frac{N_1}{N_2}\right)^2 T_2 - \left(\frac{1}{J_1}\right)b_1\dot{\theta}_2 \\ \ddot{\theta}_3 &= \left(\frac{1}{J_3}\right)T_{\text{load}} - \left(\frac{b_3}{J_3}\right)\left(\dot{\theta}_3 - \dot{\theta}_2\right) - \left(\frac{k_3}{J_3}\right)(\theta_3 - \theta_2)\end{aligned} \tag{3.4}$$

Observing the whole system, we can perform one final simplification, as $T_2 = b_3(\dot{\theta}_2 - \dot{\theta}_3) + k_3(\theta_2 - \theta_3)$. Thus, after substituting and simplifying we have:

$$\ddot{\theta}_2 = \left(\frac{1}{J_1}\right)\left(\frac{N_1}{N_2}\right)T_{\text{in}} - \left(\frac{1}{J_1}\right)\left(\frac{N_1}{N_2}\right)^2\left[b_3\left(\dot{\theta}_2 - \dot{\theta}_3\right) + k_3\left(\theta_2 - \theta_3\right)\right] - \left(\frac{1}{J_1}\right)b_1\dot{\theta}_2 \tag{3.5}$$

$$\ddot{\theta}_3 = \left(\frac{1}{J_3}\right)T_{\text{load}} - \left(\frac{b_3}{J_3}\right)\left(\dot{\theta}_3 - \dot{\theta}_2\right) - \left(\frac{k_3}{J_3}\right)(\theta_3 - \theta_2) \tag{3.6}$$

Chapter 4

Impedance Controller Design

Previously, the equations of motion describing the actuator model were derived. These equations can be used to set up a model in Simulink representing a simulated robot actuator. This model has one input and one output. u is the input and represents a torque to be applied to the actuator (T_{in}). θ_3 is the output and represents the angular position of the inertial element J_3

An impedance controller can be created in Simulink to make use of this input and output and control the simulated robot actuator. In order to formalize and explain the design of the impedance controller used in this work, we first look at a spring-mass-damper system:

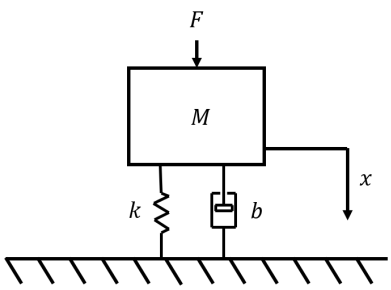

Figure 4.1: Spring Mass Damper System

The impedance controller will be based on the design of the spring mass damper system and it will dynamically control the impedance of the robot actuator through the adjustment of the impedance control parameters, where those parameters are analogous to the physical attributes of the mass spring damper system.

Where F_d is the desired force output. Force applied from the environment to the actuator is subtracted from the desired force. This represents an error term for the force. X_d is the desired position. The measured actuator position is subtracted from the desired position. This represents an error term for the position. J_e is the inertial/mass parameter (analogous to M in Figure 4.1). This scaling parameter is a gain on the acceleration of the impedance controller's spring mass damper system. B_e is the damping parameter. This scaling parameter is a gain on the velocity of the impedance controller's spring mass damper system. K_e is the spring/stiffness parameter. This scaling parameter is a gain on the position error term of the impedance controller's spring mass damper system. K_p the proportional gain constant. The "P" from "PID" control. This final parameter is a proportional gain applied to the output of the whole impedance control system.

Finally, now that all of the elements are in place we simply need to choose appropriate values for the constants defined in both the actuator model and the impedance control model, and we can then observe the dynamic behaviour of the controlled system. For example, with the following settings:

F_{env}	F_d	X_d	J_e	B_e	K_e	K_p
0	0	1	0.05	7	24	50

Table 4.1: Impedance Controller Parameters

L	N_1	N_2	J_1	b_1	J_3	b_3	k_3
1	1	10	0.005	0.5	0.05	5.0	500

Table 4.2: Robot Actuator Parameters

The observed dynamic behaviour is recorded in Figure 4.2, where the controller

over time commands the position of the actuator to converge to the desired position $X_d = 1$.

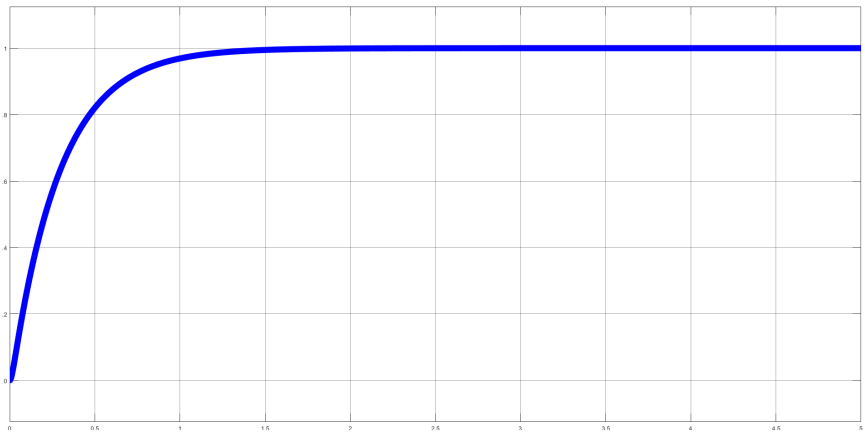

Figure 4.2: Step Response of Impedance Controlled Actuator

4.1 Simulink Model

1. discuss implementation of impedance control model in simulink
2. what kinds of visuals might be useful?
   (a) diagram of either or both impedance controller model and actuator model?
3. perhaps discuss what kind of data this model produces and how it is used
4. an overview of the Matlab tools used with appropriate citations

4.2 Controller Validation

- parameter selection for impedance control model:

  Fd = desired force

Be = damping term

Ke = spring/stiffness term

Kp = proportional gain on the system

- initial parameter selection for impedance controller?
- testing and validating response functions from actuator model

Chapter 5

Coupled Stability Analysis

This chapter presents the coupled stability analysis used to determine the stability of an impedance controlled system when interacting with various simulated environments.

5.1 Passivity Analysis

1. With the impedance controller in place, we can perform passivity analysis and curve fitting using Matlab's Control System Toolbox.

2. In order to verify if our machine learning approach is effective, we need a way to evaluate the stability of any particular set of impedance control parameters for a given environment.

3. In order to do this, we need to calculate the transfer functions of both the system and the environment. The transfer function of a linear, time-invariant differential-equation system is defined as the ratio of the Laplace transform of the output (response function) to the Laplace transform of the input (driving

function) under the assumption that all initial conditions are zero [13].

4. For a given set of impedance controller and robot actuator parameters, we can run a Simulink simulation and obtain the trajectory data for those values (see Figure 4.2 for an example). Using this trajectory data, we can curve fit the data using a nonlinear least squares method to a defined second order model. The curve-fitted parameters J_{sys}, B_{sys}, and K_{sys} define the differential-equation system of the controller and actuator system. We can define our transfer function for our controller as an impedance:

5. $$Z_{\text{sys}}(s) = \frac{Y(s)}{X(s)} = \frac{1}{(J_{\text{sys}}\ s^2) + (B_{\text{sys}}\ s) + K_{\text{sys}}} \tag{5.1}$$

6. For this work, we will characterize the environment as a mass-spring system with J_{env} and K_{env} defining the differential-equation system. We can define our transfer function for the environment as an admittance:

7. $$Y_{\text{env}}(s) = \frac{X(s)}{Y(s)} = \frac{(J_{\text{env}}\ s^2) + K_{\text{env}}}{s} \tag{5.2}$$

8. And finally, we can calculate the transfer function of the coupled dynamic system as the product of the system impedance and the environmental admittance:

9. $$G(s) = Z_{\text{sys}}\ Y_{\text{env}} = \frac{(J_{\text{env}}\ s) + (K_{\text{env}}\ \frac{1}{s})}{(J_{\text{sys}}\ s^2) + (B_{\text{sys}}\ s) + K_{\text{sys}}} \tag{5.3}$$

10. A Nyquist plot can be generated using this transfer function and analyzed to determine the stability of the system.

5.2 Curve Fitting

1. Matlab control system toolbox provides the analysis tools to generate and operate on transfer functions
2. however, we don't know the parameters of the transfer function off hand for any given set of impedance control parameters and coupled environment
3. we must estimate the required transfer functions using a curve fitting technique
   (a) first, generate a sweep of values for Ke and Be (two of our impedance control parameters)
   (b) the impedance parameter Je is kept constant at 0.05
   (c) second, we run the simulink model of the impedance controlled robot and obtain the model response to the assigned settings
   (d) FIGURE: add an example or two of the step model response to demonstrate the curve we are curve fitting to
   (e) in matlab, we use code for a 2nd order model in order to curve fit a differential equation to the data plot from simulink
      i. (TODO: need to review this code ore closely and parse it so I can give a proper explanation in this paper)
4. FIGURE: add a figure demonstrating the curve fitted plot to the response from the model
5. discuss the fact that the curve fit is NOT perfect, it is an approximation
   (a) How does the approximation affect overall performance? Is this something I can measure?

Chapter 6

Neural Network

This chapter presents the different neural network models, a detailed description of the data set and how it is set up, and how the models are trained. Different models are created to find the best performing architecture that is ideal for this problem space.

This chapter is structured in the following way: (1) a discussion of the problem setup and the step by step process of how and why the data set was designed, (2) the three neural network architectures used is then detailed, and (3) a comparison of the benefits and shortfalls of each architecture is finally discussed.

6.1 Problem Setup

The goal is to develop a neural network that, given system parameters as input, can predict a stable damping value for the coupled dynamic system. Using Matlab, a simple model was developed to represent a robot arm as described in Chapter XX. In chapter XX, a state space model is defined which in turn is used to develop a transfer function representing the robot arm model and control system and the environment

with which it interacts. The parameters of the system or the environment can be adjusted, i.e., inertia and stiffness, and the stability of the system can be observed.

The stability of the system can be analyzed by looking at the closed loop coupled system which can be described by

<equation 15>

For coupled system analysis, this can be by looking at the open loop component of the coupled system.

<equation 16>

Using the Nyquist analysis method and varying control system parameters with the tools available in MATLAB, we can determine whether or not a specific coupled system conditions are stable and adjust parameters to make the system stable, e.g., adjust the damping value of the impedance controller for an already passive system.

6.1.1 Generating A Dataset

One of the primary concerns and most important steps in the development of an optimal machine learning system is to procure a high quality data set to use for training. A good data set has a sufficient number of samples that cover a broad range of inputs that the system is likely to encounter. In addition, it is important that these inputs are in a numerical format that lends itself to being used in machine learning methods.

Table 6.1 lists and defines the parameters used in the model for data set creation. These parameters are used to generate a data set of eleven thousand samples.

J_e	The emulated moment of inertia in the impedance controller.
B_e	The emulated damping in the impedance controller.
K_e	The emulated stiffness in the impedance controller.
M_{env}	The emulated mass of the environment.
B_{env}	The emulated damping of the environment. This value is always set equal to zero.
K_{env}	The emulated stiffness of the environment.

Table 6.1: The parameters that are adjusted to produce the data set.

Using the transfer function representing our system and environment we can iterate through a large set of value ranges for the parameters, observe the stability of the resulting system, and record values that generate stable systems. In order to get an optimized set of damping values to train with there are several steps of processing that was performed:

The minimum stable damping value (B_e) for each combination of eleven thousand parameter samples is calculated via the following method:

$B_e = 1$
if coupled system with B_e = unstable **then**
    $B_e = B_e * 1.05$
else
    if coupled system with B_e = stable **then**
        stable system has been found
    end if
end if

After calculating the minimum stable damping values for the entire data set, some additional processing is done in order to create a more optimal data set. When training a neural network on a set of data, the resulting neural network may over or underestimate its predictions within an acceptable range. Specific to our use case, we never want these predictions to be underestimated when dealing with the

minimum stable damping values as any values below this will result in an unstable system. Moderate to large overestimates may still produce a stable system. While we want the neural network to predict the smallest stable value possible, there are many scenarios where a slightly larger value will still be stable thus providing an incentive to train a neural network that consistently overestimates rather one that can possible underestimate.

In order to improve the accuracies of neural networks trained on this data the minimum values were averaged using a nearest neighbors approach, with the rule that any individual value may not drop below its minimum stable damping value. This results in a smoother regression curve that is easier to fit to while training a neural network.

Another processing method was to take the log of the input data and output data when training the neural networks. The difference in stable damping values can vary exponentially, and using a log scaling results in a data set that is easier to work with. More specifically, it simplifies the regression problem the neural network is trying to solve by eliminating extreme variations in the training data from data point to data point.

As described in Table 6.1, our system and environment is defined by six variables. These values themselves could be used as inputs directly into a neural network. However, through experimentation, it was found that converting these values into the natural frequency of the impedance controller and the natural frequency of the environment yielded a data set which trained better and had higher accuracies.

The natural frequency is defined as:

$$W_{SYS} = \sqrt{\frac{K_e}{J_e}} \qquad\qquad W_{ENV} = \sqrt{\frac{K_{ENV}}{M_{ENV}}} \tag{6.1}$$

Simplifying our multi-parameter system into these natural frequencies makes the

mapping of our regression space straightforward. In essence, a four input one output system is converted into a two input one output system where the inputs are the natural frequencies of the system and the environment and the output is the stable damping value..

This application of machine learning is well defined as a regression problem, as we are attempting to map a specific set of input parameters to a stable damping output value. This regression problem can be presented as a mapping into a three dimensional space, where the x and y axis may be defined as the natural frequency of the system and the environment and the z axis is defined to be the stable damping value which is the solution.

6.1.2 Data Sweep of Parameters

The parameters used by the impedance controller system and the environment represent the parameters of spring damping systems. The parameter ranges selected thus reflect the common conventions of such systems as well as a selection for systems that behave in a particular manner when given different values for damping in the impedance controller system.

The following is an example of the range of values selected in Matlab code:

```
Je = 0.125 * (0.5 : 0.5 : 5);
    % The value 0.125 represents the baseline value for the emulated
    % moment of inertia of the impedance controller.
Ke = 25 * (0.5 : 0.5 : 5);
    % The value 25 represents the baseline value for the emulated
    % stiffness of the impedance controller.
M_env = 0.125 * (0.5 : 0.5 : 5);
```

```
    % The value 0.125 represents the baseline value for the emulated
    % mass of the environment.
B_env = 0;
    % The environment is defined as a spring mass system with
    % no damping.
K_env = 25 * (0.5 : 0.5 : 5);
    % The value 25 represents the baseline value for the emulated
    % stiffness of the environment.
```

6.1.3 Regression Map

6.1.4 Function Approximation

6.1.5 Choosing A Neural Network

Having defined the system and generated a data set as previously described, the next step in the problem setup is to choose a neural network architecture to use in order to generate a neural network and train it on the data set. Matlab provides a robust set of tools for generating and training neural networks.

Three neural network types were chosen in order to compare and contrast performance among different architectures in the pursuit of an optimal result:

1. generalized regression neural network (GRNN)

2. function fitting neural network (fitnet)

3. deep learning network (DL network)

6.2 Generalized Regression Neural Network

A generalized regression neural network is a radial basis function network. A radial basis function is a function whose value depends on the distance between the input and a fixed point. Specifically in terms of this work, the fixed points may be represented as the training set output (the ideal damping values to produce stability). Sums of radial basis functions are used to approximate given functions. The approximation process can be interpreted as a kind of neural network called a radial basis function network.

Radial basis function networks are a type of artificial neural network that uses radial basis functions as activation functions. The network output is a linear combination of radial basis functions of the inputs and neuron parameters. One of the primary uses for these networks is for function approximation, which as previously discussed, is the core operation we are trying to achieve.

A GRNN network is produced in Matlab by defining the inputs and outputs as a subset of the data set and the desired training results. Also, a value for the spread of the radial basis functions must be included. Generally speaking, the larger the value for the spread the more smooth of a function approximation results. Alternatively, smaller values for the spread produces a tighter fit but may risk over-fitting the training data if taken too far.

The pros to this approach include the fact that this type of network is very fast to design and implement and generally produces a good result for the type of function approximation problem we are trying to solve. By design, this neural network contains a single hidden layer composed of a number of neurons equal to the number of inputs used to train. As such, we select a subset of the entire data set to use to represent this hidden layer. Unfortunately, this presents a con to the approach in the sense that we simply cannot use an entire data set as input because

it would use too much memory to run and train in Matlab.

6.3 Function Fitting Neural Network

The function fitting neural network (or fitnet) is a standard regression/function fitting neural network architecture available in Matlab. While it is designed to perform the same function as the GRNN previously discussed, there are some important key differences. The biggest difference is in the structure of the network. While the GRNN is designed as a single layer neural network, fitnet allows the user to completely decide the size and structure of the network. This includes the number of hidden layers and the number of neurons in each hidden layer. Thus, because we can define the structure of the network more precisely we are also able to train the network on the entire data set instead of a subset of the values. Many different variations in number of neurons per layer and number of layers were tested to see which configuration produced the best training results.

The benefits of this type of network over both GRNN and the Deep Learning Network is that the speed of training is generally faster with this system. There is more control over parameters of the neural network than the GRNN but also less control than with the Deep Learning Network tool.

6.4 Deep Learning Network

The deep learning network (DL Network) is the third neural network design tool that was tested in this work. Just as fitnet provided more variables and settings to adjust than GRNN, the DL Network in turn provides even more flexibility and control than fitnet. The deep learning network tool allows each individual layer in the network to

be customized and adjusted. While this feature is available to fitnet, the ability to do so with the DL network is significantly more streamlined and easier to use.

Each hidden layer was designed to be fully connected and the ReLu activation function is used between each layer. The same number of layers and neurons per layer were used as fitnet in order to provide a direct comparison of performance between the two neural network structures.

While extremely flexible, this comes at the cost of run time. This network on average takes more time to train than the other networks discussed. However, that being said, this particular architecture ended up performing with the best accuracy.

6.5 Model

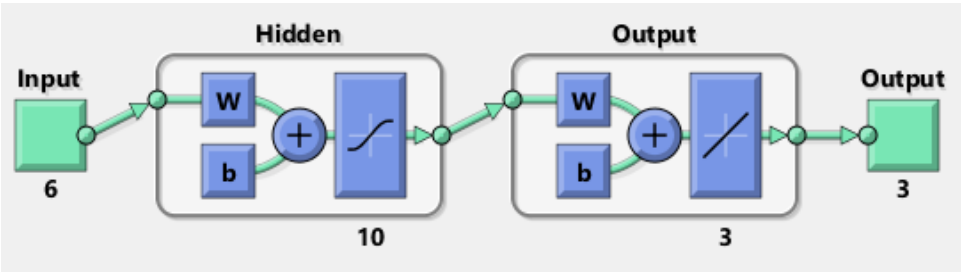

Figure 6.1: Model diagram for function fitting neural network. This network has three layers: one input layer, one hidden layer, and one output layer. The input and output layers are of size six and three respectively due to the required input and output parameters. The hidden layer was chosen to have ten neurons, but can be set to any arbitrary size and number of hidden layers.

Number of Input Neurons	Number of Hidden Layers	Neurons Per Layer	Number of Output Neurons
6	1	2	3
6	1	3	3
6	1	6	3
6	1	9	3
6	1	18	3
6	3	2	3
6	3	3	3
6	3	6	3
6	3	9	3
6	3	18	3
6	6	2	3
6	6	3	3
6	6	6	3
6	6	9	3
6	6	18	3
6	9	2	3
6	9	3	3
6	9	6	3
6	9	9	3
6	9	18	3
6	12	2	3
6	12	3	3
6	12	6	3
6	12	9	3
6	12	18	3

Table 6.2: Different Sizes of Neural Networks Implemented

1. Why choose a neural network approach as opposed to another approach such as nonlinear regression?

   (a) TODO: need to come up with the pros and cons of the NN technique vs a more straightforward way to simply adjust impedance control on the fly

2. Using the Matlab Deep Learning Toolbox, the Neural Network design used in this application was the function fitting neural network (fitnet). This network

was chosen because function fitting is essentially what we want to achieve with impedance control parameter tuning. We would like to create a generalized function in the sense that given any set of impedance control parameters and environment parameters, we will generate a mapping to a new and stable set of impedance control parameters to use in a control loop. The advantage of this design is that it is lightweight and easy to integrate such a system into a simulated or hardware control loop without significant overhead.

3. features of fitnet:

   (a) has an input layer (current impedance control parameters and sensed environment parameters)

   (b) has an output layer (new suggested impedance control parameters)

   (c) has at least one or more hidden layers; each layer may have any number of neurons

4. In order to test and verify which type of neural network architecture would be appropriate for this application, several iterations of neural networks were created with different numbers and sizes of hidden layers. Each of these fitnet neural networks was trained on the full data set and performance was compared in order to determine which architecture produced the most reliable results.

5. One major component of this neural network structure is the training function. In our neural network we use the Bayesian regularization back propagation training function. This function updates the weight and bias values of the neural network according to Levenberg-Marquardt optimization. It minimizes a combination of squared errors and weights, and then determines the correct combination to produce a network that generalizes well. Additionally, the transfer function used in the neural network is the hyperbolic tangent Sigmoid transfer function.

6.6 Dataset Design and Collection

In order for a machine learning application to be successful, it requires a large and high quality data set. The neural network used in this work will have the target inputs and outputs specified in Table 6.3

Inputs	Outputs
J_e	J'_e
B_e	B'_e
K_e	K'_e
J_{env}	
B_{env}	
K_{env}	

Table 6.3: The neural network will take six inputs including three impedance control parameters and three environment parameters and produce three outputs which will be suggested updates to the impedance control parameters.

In order to build the data set, we first sampled a large range of impedance controller values. The inertial element J_e was kept fixed at 0.05. The spring/stiffness value K_e was sampled in the range of $[1, 1000]$ with a step size of 1. The values for B_e were sampled as a function K_e and J_e in order to maintain critical damping of the system: $B_e = 2 \times \sqrt{J_e \times K_e}$ for each value of K_e. With these samples, we then run the Simulink model and curve fitting routines on all combinations of parameters to produce a sample of one thousand unique, critically damped impedance control parameters.

Next, we built a sample of environment parameters. K_{env} was sampled using the function $f(x) = 10^x$ from $[0.5, 10]$ with a step size of 0.5. This results in twenty unique values for K_{env}. J_{env} was sampled from $[1, 100]$ with a step size of 2, resulting in fifty unique values for J_{env}. B_{env} was kept fixed at 0 since the mass-spring model for the environment has no damping. Using the two samples of J_{env} and K_{env} with B_{env} fixed, we produced a set of one thousand unique environment parameters.

The final step is to generate every possible combination of impedance control parameters and environment parameters, resulting in a final data set of one million entries. In addition, the Nyquist stability analysis was performed on each of these entries and labeled to show if the critical point in the Nyquist plot was circled (unstable) or not circled (stable) in order to have a labeled data set for machine learning training.

Chapter 7

Results

This chapter summarizes the experimental results from the trained neural networks used for predicting impedance control parameters.

7.1 Neural Network Accuracy

Three different neural network architectures were tested in order to find a solution to the regression problem presented in this work. Three metrics were calculated during the training of the neural networks:

1. accuracy — This value represents the accuracy of a neural network to correctly generate a stable damping value for a system across all eleven thousand samples in the data set used to train the network. This accuracy is a percentage indicating the correct number of predictions across the entire data set after being trained.

2. MSE — This value represents the mean squared error across the predictions for the entire data set for the damping values generated versus the ideal damping

values calculated for the training data set. It is both helpful and hurtful that there are many damping values that may generate a stable system for a given set of parameters. We can use the MSE to measure the quality of damping values beyond simply generating a stable system but to indicate if it is generating the best possible stable system.

3. run time — This value represents the average amount of time in seconds that a neural network takes to produce a prediction. This value was calculated by having the neural network generate a prediction for every sample in the data set. Each prediction was timed and the average prediction time across all samples was taken.

All three neural network architectures were able to perform well, achieving accuracies that could reach or exceed 90%. The least performing neural network architecture was Fitnet, with a highest accuracy value of 92.9273%.

The Deep Neural Network package performed the best with a highest accuracy value of 98.1%. However, when comparing MSE values we can see that there is a trade off for this high accuracy. In general, this particular network overestimates damping values compared to other neural networks.

Looking at the results for the GRNN, we see that the best accuracy with this neural network architecture was 95.0091%. This is not as high as the DL Network, but there is once again a trade off. The MSE for the GRNN is orders of magnitude lower. This indicates that while the frequency of correct predictions will be lower, the quality of those predictions will be higher.

sample size	spread	accuracy	MSE	run time (sec)
100	20	83.5182	8.89 e1	7.80 e-3
100	15	82.7455	8.15 e1	7.89 e-3
100	10	81.9182	2.98 e2	7.50 e-3
100	5	86.7455	5.03 e2	7.34 e-3
100	1	86.7455	5.68 e2	7.27 e-3
100	0.1	86.6909	5.68 e2	6.65 e-3
100	0.01	86.6909	5.68 e2	5.93 e-3
100	0.001	86.6909	5.68 e2	6.07 e-3
100	0.0001	86.6909	5.68 e2	6.29 e-3
100	0.00001	86.6909	5.68 e2	6.29 e-3
200	20	87.7273	2.55 e1	7.70 e-3
200	15	88.3091	4.54 e0	7.63 e-3
200	10	88.0818	5.18 e-1	7.48 e-3
200	5	85.7364	3.01 e1	7.46 e-3
200	1	95.0091	3.10 e2	7.54 e-3
200	0.1	95.0091	3.10 e2	7.67 e-3
200	0.01	95.0091	3.10 e2	7.63 e-3
200	0.001	95.0091	3.10 e2	7.55 e-3
200	0.0001	95.0091	3.10 e2	7.68 e-3
200	0.00001	95.0091	3.10 e2	7.37 e-3
500	20	90.0636	2.38 e2	6.31 e-3
500	15	90.8636	6.64 e2	6.46 e-3
500	10	91.6727	4.58 e2	6.31 e-3
500	5	91.6727	1.58 e1	6.26 e-3
500	1	93.9909	1.83 e2	7.94 e-3
500	0.1	93.9818	1.82 e2	9.57 e-3
500	0.01	93.9818	1.82 e2	6.16 e-3
500	0.001	93.9818	1.82 e2	6.62 e-3
500	0.0001	93.9818	1.82 e2	5.49 e-3
500	0.00001	93.9818	1.82 e2	5.87 e-3
1000	20	90.8273	3.88 e0	8.54 e-3
1000	15	90.6727	1.92 e0	5.40 e-3
1000	10	90.7364	3.09 e-2	5.37 e-3
1000	5	91.6545	7.91 e-1	5.35 e-3
1000	1	92.2636	4.26 e1	6.10 e-3
1000	0.1	92.2636	4.35 e1	7.02 e-3
1000	0.01	92.2636	4.35 e1	6.61 e-3
1000	0.001	92.2636	4.35 e1	6.28 e-3
1000	0.0001	92.2636	4.35 e1	6.33 e-3
1000	0.00001	92.2636	4.35 e1	6.87 e-3

Table 7.1: (prediction accuracy) results for GRNN

Table 7.2: FITNET Results

Layers	***Neurons***	***% Stable***	***MSE***	***Run Time [ms]***
1	1	94.62%	1.82E+02	6.37
1	10	90.05%	2.83E+02	6.41
1	100	88.59%	7.44E+01	6.51
1	500	89.54%	2.56E+01	6.56
1	1000	88.20%	6.74E+02	6.64
2	1	94.74%	1.69E+02	6.97
2	10	88.89%	4.20E+02	7.06
2	100	88.98%	3.23E+02	7.41
2	500	88.39%	3.63E+02	13.87
2	1000	87.05%	1.14E+03	34.30
4	1	94.66%	4.26E+02	8.01
4	10	90.62%	1.17E+01	8.12
4	100	90.67%	1.32E+02	8.68
4	500	88.13%	9.44E+02	28.13
4	1000	88.25%	6.55E+00	73.67
8	1	94.29%	3.32E+01	10.14
8	10	92.13%	7.31E+02	10.28
8	100	89.45%	4.82E+01	11.39
8	500	89.77%	3.54E+01	52.54
8	1000	88.82%	5.16E+01	172.31
12	1	94.87%	1.58E+02	12.46
12	10	91.21%	9.23E+02	12.60
12	100	90.33%	1.29E+02	14.31
12	500	89.42%	4.45E+02	82.10
12	1000	89.20%	1.66E+03	298.95

Table 7.3: DL Network Results

Layers	***Neurons***	***% Stable***	***MSE***	***Run Time [ms]***
1	1	96.15%	6.74E+04	3.52
1	10	97.15%	3.13E+04	3.07
1	100	90.76%	1.66E+06	3.16
1	500	89.76%	1.22E+06	3.18
1	1000	86.19%	2.46E+06	3.45
2	1	96.00%	7.92E+04	3.02
2	10	97.71%	3.17E+03	2.99
2	100	89.20%	1.83E+06	3.16
2	500	84.58%	4.32E+05	3.75
2	1000	92.23%	3.54E+05	3.10
4	1	41.80%	4.94E+07	3.24
4	10	95.93%	1.95E+05	3.17
4	100	97.11%	1.74E+06	3.09
4	500	95.05%	3.16E+05	5.20
4	1000	94.81%	9.23E+05	3.66
8	1	41.92%	4.94E+07	3.06
8	10	93.72%	2.99E+05	3.19
8	100	96.54%	7.53E+05	3.00
8	500	95.31%	1.55E+06	4.13
8	1000	88.58%	3.16E+06	3.76
12	1	41.73%	4.94E+07	3.23
12	10	96.73%	3.77E+05	3.21
12	100	93.07%	6.51E+04	3.16
12	500	88.46%	1.33E+05	4.37
12	1000	93.06%	2.98E+05	4.42

Chapter 8

Discussion

1. Initial results show that this method may be useful for generating impedance control parameters for ensuring stability.
   (a) some networks didn't train very well, the lowest accuracy network was approximately 28%
   (b) the best performing networks were able to predict stable impedance control parameters for all 100% of the one million samples in the data set
2. the mse (mean squared normalized error) performance measure function was used during the training of the neural networks
   (a) the error for any particular network trained was always a very large value (need to look back at data in matlab to grab specific figures)
   (b) even with that being the case, what we want to measure is the prediction stability, which is NOT specifically trained for with the neural network
   (c) something of note: for any given prediction, there is a RANGE of possible stable values. In its current form, the analysis done here does not check for the optimal solution, only a sane and stable one. This is why the

prediction accuracy in these results is much better than the MSE measure would indicate

3. After training and testing several different networks of different sizes, some apparent optimal configurations emerge from the results.

   (a) the fastest neural networks had the fewest layers, but with the trade off in accuracy this indicates that a network with 6 to 9 layers may be best

   (b) the number of neurons per layer didn't affect the run speed meaningfully, therefore the only consideration for the number of neurons is the prediction accuracy and 2 to 6 neurons per hidden layer yielded the best stable prediction accuracies

4. Future work will include implementing a trained neural network into a robot control loop (both simulated and real hardware), to verify the efficacy of the method in hardware.

   (a) because of the necessity to integrate the neural network into a REAL TIME control loop, it needs to be as fast as possible or at least as fast as is necessary for a control loop to remain stable

Chapter 9

Conclusion

1. A robot actuator model was designed and developed along with an impedance controller in Matlab/Simulink. This model was used as the basis to test a neural network method for choosing impedance control parameters to ensure stability.

2. Experimental results show that this method is viable for use in a robot control system for maintaining stability

   (a) discuss the accuracy of the models

   (b) discuss the run speed of the models

   (c) perhaps an additional experiment to test on additional input outside of the original data set?

   (d) figure or picture to help?

3. some other ideas for future work may include:

   (a) doing additional experiments with different types of training functions

(b) there are many settings for these neural networks, perhaps working with these settings to test aditional configurations

(c) taking the existing trained networks and implementing them into a simulated or hardware control loop and analyze its effectiveness

Appendices

Appendix A

Data Set Generation Code

There are four Matlab scripts used to generate the data set:

1. Param_Sweep_1.m

   Generates all of the K_e and B_e values used to generate the data set. J_e is kept to a constant value.

2. Param_Sweep_2.m

   Generates all of the K_{env} and J_{env} values used to generate the data set. B_{env} is kept to a constant value.

3. Param_Sweep_3.m

   This script builds on data generated in the first two scripts by combining the two sets of variables into every permutation of values possible and recording the resulting stability.

4. Param_Sweep_4.m

   This script completes the data set by calculating a stable set of impedance control parameters to give as an "answer" for each sample when training.

A.1 Param_Sweep_1.m

```
clear;
close all;
setup;

% 1. sweep a selection of Ke values ranging from 1 to 1000
% 2. solve for Be using the critical damping ratio formula
% 3. keep Je at a fixed value for all experiments

% Select a set of values to sweep for Ke
Ke_sweep = 1:1000;

% Based on the Ke values, use the critical damping formula
% to solve for the critical damping value for Be
Be_sweep = 2 * sqrt(Je * Ke_sweep);

% Store the curve fitted impedance control values in this matrix
Bs = zeros(3, size(Ke_sweep, 2));

global io bo ko tscale theta3 Fapplied

% Keep the inertia element value fixed
Je = 0.05;
io = Je;

% Sample every "delta"-th point of model output in order to make
% the curve fitting process more efficient
```

```
delta = 10;

% options for lscurvefit():
%    don't display output on each iteration
opts = optimset('Display', 'off');

for i = 1:size(Ke_sweep, 2)
    fprintf("Model sim %d of %d\n", i, size(Ke_sweep, 2));

    % Update the impedance controller settings:
    Be = Be_sweep(i);
    Ke = Ke_sweep(i);

    % Run the model with updated Be, Ke, values
    model = sim('actuator_model.slx');

    % Update curve fit parameters:
    bo = Be;
    ko = Ke;
    tscale = model.tout(1:delta:size(model.tout, 1));
    theta3 = ...
        model.theta3.Data(1:delta:size(model.theta3.Data, 1));
    Fapplied = model.F_env.Data;

    % Curve fit with a second order model
    B = ...
    lsqcurvefit(@eom_uscf,[bo io ko],tscale,theta3,[],[],opts);
```

```
    % Store the curve-fitted impedance control parameters
    Bs(1, i) = B(1, 2); % io
    Bs(2, i) = B(1, 1); % bo
    Bs(3, i) = B(1, 3); % ko
end

% Write the curve fit data out to a file:
save('Dataset.mat', 'Bs');
```

A.2 Param_Sweep_2.m

```
clear;
close all;

load('Dataset.mat');

% sample K_env values exponentially with these formulas
f = @(x) 10 .^ x;

% Sweep every combination of a selection of
% Ke and Je values for a Mass-Spring system
% K_env_sweep = [1:99, 100:1e5:(1e5-1) 1e5:1e6:(1e7-1) 1e7:1e7:1e9];
K_env_sweep = f(0.5:0.5:10);
J_env_sweep = 1:2:100;

Es = zeros(3, size(K_env_sweep, 2) * size(J_env_sweep, 2));
```

```
j = 1;
for i = 1:size(J_env_sweep, 2)
    for k = 1:size(K_env_sweep, 2)
        fprintf("Environment sim %d of %d\n", j, size(J_env_sweep, 2) * size(K_env
        Es(1, j) = J_env_sweep(i); % io
        Es(2, j) = 0;              % bo
        Es(3, j) = K_env_sweep(k); % ko
        j = j + 1;
    end
end

save('Dataset.mat', 'Bs', 'Es');
```

A.3 Param_Sweep_3.m

```
clear;
close all;

load('Dataset.mat');

deltaB = 1;
deltaE = 1;

count = 1;
circled_count = 0;
max_count = size(1:deltaB:size(Bs, 2), 2) * size(1:deltaE:size(Es, 2), 2);
BEs = zeros(7, max_count);
```

```
for i = 1:deltaE:size(Es, 2)
    for j = 1:deltaB:size(Bs, 2)
        fprintf("Param sweep %d of %d\n", count, max_count);

        % generate the transfer function for the given impedance
        % control parameters and environment parameters
        Z_sys = tf(1, Bs(:, j)');
        Y_env = 1 / tf([1, 0] , Es(:, i)');
        G = Z_sys * Y_env;

        % produce nyquist plot data from the transfer function
        [re, im] = nyquist(G);
        re = floor(reshape(re, [], 1));
        im = floor(reshape(im, [], 1));

        % 1) calculate the angles of each re point from the
        %    critical point
        % 2) sample all points in 45 degree increments to test if the
        %    nyquist plot line circles the critical point
        deg = floor(abs(rad2deg(atan2(im, re + 1.0))));
        points = (sum(ismember(0:45, deg)) > 0 && ...
                  sum(ismember(136:180, deg))) > 0 ...
                  && ...
                 (sum(ismember(46:90, deg)) > 0 || ...
                  sum(ismember(91:135, deg))) > 0;

        % In order to catch some edge cases, also ensure that the
```

```
% nyquist plot line will cross the imaginary (X) axis at
% least once on each side of the critical point.
left = false;
right = false;
for k = 2:size(im, 1)
    if (   im(k) <= 0 && im(k - 1) >= 0 ...
        || im(k) >= 0 && im(k - 1) <= 0) ...
        && (re(k) < -1 || re(k - 1) < -1)

        left = true;
    end
    if (   im(k) <= 0 && im(k - 1) >= 0 ...
        || im(k) >= 0 && im(k - 1) <= 0) ...
        && (re(k) > -1 || re(k - 1) > -1)

        right = true;
    end
end

circled = left && right && points;

if circled
    circled_count = circled_count + 1;
end

BEs(1:3, count) = Bs(:, j);
BEs(4:6, count) = Es(:, i);
BEs(7, count) = circled;
```

```
        count = count + 1;
    end
end

fprintf("%d of %d critical points circled (unstable)\n", ...
    circled_count, max_count);

save('Dataset.mat', 'Bs', 'Es', 'BEs');
```

A.4 Param_Sweep_4.m

```
clear;
close all;

load('Dataset.mat');

one = sum(BEs(7, :));
zero = size(BEs, 2) - one;
all = one + zero;

BE_0 = zeros(10, zero);
BE_1 = zeros(10, one);

j = 1;
k = 1;
for i = 1:all
```

```
    % For all known stable systems, simply keep the original
    % impedance control settings
    if BEs(7, i) == 0
        BE_0(1:7, j) = BEs(1:7, i);
        BE_0(8:10, j) = BEs(1:3, i);
        j = j + 1;
    % For all known unstable systems, leave the prediction
    % blank for now, the next loop will calculate a new
    % stable output based on the unstable input by adjusting
    % the damping of the unstable impedance control settings
    elseif BEs(7, i) == 1
        BE_1(1:7, k) = BEs(1:7, i);
        % BE_1(8:10, k) = calculated down below
        k = k + 1;
    end
end

tic;
damping_scalar = 10;

for i = 1:one
    % error(1:zero) = sum(abs(BE_0(4:6, 1:zero) - BE_1(4:6, i)));
    % [~, newBE_index] = min(error);
    % BE_1(8:10, i) = BE_0(1:3, newBE_index);
    % disp([i one]);

    new = BE_1(1:3, i);
    env = BE_1(4:6, i);
```

```
circled = true;
attempts = 0;
damping_delta = damping_scalar ^ attempts;

% systematically find a stable configuration by increasing the
% damping value of the impedance controller, scaling it up
% exponentially with each attempt: f(n) = damping_scalar ^ n

while circled % && attempts <= 1000
    new(2) = new(2) + damping_delta;
    % generate the transfer function for the given impedance
    % control parameters and environment parameters
    Z_sys = tf(1, new');
    Y_env = 1 / tf([1, 0] , env');
    G = Z_sys * Y_env;

    % produce nyquist plot data from the transfer function
    [re, im] = nyquist(G);
    re = floor(reshape(re, [], 1));
    im = floor(reshape(im, [], 1));

    % 1) calculate the angles of each re point from the critical
    %    point
    % 2) sample all points in 45 degree increments to test if the
    %    nyquist plot line circles the critical point
    deg = floor(abs(rad2deg(atan2(im, re + 1.0))));
    points = (sum(ismember(0:45, deg)) > 0 && ...
              sum(ismember(136:180, deg))) > 0 ...
```

```
            && ...
           (sum(ismember(46:90, deg)) > 0 || ...
                                % this "or" is on purpose
            sum(ismember(91:135, deg))) > 0;

    % In order to catch some edge cases, also ensure that the
    % nyquist plot line will cross the imaginary (X) axis at
    % least once on each side of the critical point.
    left = false;
    right = false;
    for k = 2:size(im, 1)
        if (   im(k) <= 0 && im(k - 1) >= 0 ...
            || im(k) >= 0 && im(k - 1) <= 0) ...
            && (re(k) < -1 || re(k - 1) < -1)

            left = true;
        end
        if (   im(k) <= 0 && im(k - 1) >= 0 ...
            || im(k) >= 0 && im(k - 1) <= 0) ...
            && (re(k) > -1 || re(k - 1) > -1)

            right = true;
        end
    end

    circled = left && right && points;
    attempts = attempts + 1;
    damping_delta = damping_scalar ^ attempts;
```

```
    end

    disp([i one circled attempts]);
    disp(100 * i / one);
    BE_1(8:10, i) = new;
    toc;
end

Dataset = [BE_0 BE_1];

save('Dataset.mat', 'BEs', 'Bs', 'Es', 'Dataset');
```

References

[1] Neville Hogan. Impedance control: An approach to manipulation. In *1984 American control conference*, pages 304–313. IEEE, 1984.

[2] Charles C Kemp, Aaron Edsinger, and Eduardo Torres-Jara. Challenges for robot manipulation in human environments [grand challenges of robotics]. *IEEE Robotics & Automation Magazine*, 14(1):20–29, 2007.

[3] Aude Billard and Danica Kragic. Trends and challenges in robot manipulation. *Science*, 364(6446):eaat8414, 2019.

[4] D. Whitney. Historical perspective and state of the art in robot force control. In *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, volume 2, pages 262–268, 1985.

[5] Jonathon E. Slightam, Daniel R. McArthur, Steven J. Spencer, and Stephen P. Buerger. Passivity analysis of quadrotor aircraft for physical interactions. In *2021 Aerial Robotic Systems Physically Interacting with the Environment (AIRPHARO)*, pages 1–8, 2021.

[6] P. Bondi, G. Casalino, and L. Gambardella. On the iterative learning control theory for robotic manipulators. *IEEE Journal on Robotics and Automation*, 4(1):14–22, 1988.

[7] Danwei Wang and Chien Chern Cheah. An iterative learning-control scheme for impedance control of robotic manipulators. *The International Journal of Robotics Research*, 17(10):1091–1104, 1998.

[8] MathWorks Inc. *Deep Learning Toolbox*. Natick, Massachusetts, United States, 2020.

[9] Moshe Cohen and Tamar Flash. Learning impedance parameters for robot control using an associative search network. *IEEE Transactions on Robotics and Automation*, 7(3):382–390, 1991.

[10] Toshio Tsuji and Yoshiyuki Tanaka. On-line learning of robot arm impedance using neural networks. *Robotics and Autonomous Systems*, 52(4):257–271, 2005.

[11] Seul Jung and Tien C Hsia. On neural network application to robust impedance control of robot manipulators. In *Proceedings of 1995 IEEE International Conference on Robotics and Automation*, volume 1, pages 869–874. IEEE, 1995.

[12] Dusko Katic and Miomir Vukobratovic. Learning impedance control of manipulation robots by feedforward connectionist structures. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 45–50. IEEE, 1994.

[13] Katsuhiko Ogata. *System dynamics*. Pearson/Prentice Hall, Upper Saddle River, NJ, 4th ed. edition, 2004.

[14] Byungchan Kim, Jooyoung Park, Shinsuk Park, and Sungchul Kang. Impedance learning for robotic contact tasks using natural actor-critic algorithm. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 40(2):433–443, 2009.

[15] Suguru Arimoto, PTA Nguyen, and T Naniwa. Learning of robot tasks via impedance matching. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*, volume 4, pages 2786–2792. IEEE, 1999.

[16] Chien-Chern Cheah and Danwei Wang. Learning impedance control for robotic manipulators. *IEEE Transactions on Robotics and Automation*, 14(3):452–465, 1998.