ADAPTIVE NEURAL CONTROL OF A GIMBALED LASER TARGETING SYSTEM

A THESIS SUBMITTED TO
THE TEMPLE UNIVERSITY GRADUATE BOARD

IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in ELECTRICAL ENGINEERING

---

by
Salvatore Giorgi
May, 2014

Examining Committee Members:

Chang-Hee Won, Advisory Chair, Electrical and Computer Engineering
John Helferty, Electrical and Computer Engineering
Dennis Silage, Electrical and Computer Engineering

ABSTRACT

Space-based solar power is a better alternative to the ground-based solar system, because of its round-the-clock availability. Accurately pointing the laser from space to a ground receiver can increase transmission efficiency and decrease the receiving area. A superior pointing performance requires an accurate system model and controller. In this thesis, we investigate Adaptive Neural Control for a laser targeting application. A two-axis Gimbaled Laser Target System (GLTS) is used as hardware test bench of the space-based solar power transmission system.

The Adaptive Neural Control (ANC) system, first proposed by D.C. Hyland, is a neural control system within a Model Reference Control (MRAC) architecture. It is composed of five separate neural networks, two of which are used to replicate an unknown plant, while the remaining three are used to control the plant's output to match that of an ideal reference system. The system has been successfully used in hardware such as the NASA / LaRC Mini-MAST testbed and the ASTREX testbed at Airforce Philips Laboratory. It has been shown to be very effective in terms of robustness, fault tolerance, and optimality.

The objective of this research is to apply the ANC system to the problem of pointing and tracking the line of sight of a GLTS. A software model of the ANC system is built using Matlab / Simulink. We then simulate control of a linear stochastic model of our GLTS test bed and compare the ANC system's performance to that of a Proportional Integral Derivative (PID) controller. Next, we consider a separate nonlinear stochastic model of a two-axis gimbal, and consider the problem of platform stabilization using the ANC system.

Next, we examine the ANC system's resiliency, defined in terms of how the controller maintains operational normalcy in response to anomalies, both unexpected and malicious. These anomalies will be in the form of added latencies, plant parameter changes, false data injec-

tion, and sensor data alteration. We simulate the attacks on the GLTS model and determine the system's resiliency to each attack through four metrics. These metrics are recovery time, performance degradation, protection time, and degrading time. We then compare these results to that of a PID controller subjected to the same attacks.

Finally, the ANC system software model is translated to a fixed point hardware model for implementation on a Field Programmable Gate Array (FPGA) using Xilinx / System Generator. This software to hardware translation considers special attention to floating point to fixed point conversion, division, representing nonlinear neural functions, and hardware resource allocation. System replication and control simulations are run for the linear stochastic GLTS model. These simulations include "hardware in the loop" simulations, where the actual FPGA is used within the Matlab simulation.

Simulations show that the software model of the ANC system is able to replicate and control the linear GLTS model as well as a separate nonlinear gimbal model in the presence of process and measurement noise, with no prior modeling information. Additionally, the simulations demonstrate the resiliency of the ANC system when exposed to attacks, in terms of recovery time, performance degradation, and degrading time. Lastly, hardware simulations confirm the ANC system's ability to control the GLTS model using the FPGA.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my adviser, Dr. Chang-Hee Won of the Electrical and Computer Engineering Department at Temple University for his continuous patience and support. His guidance helped me in all times of research, writing, and organizing this thesis.

I would like to thank the rest of my thesis committee: Dr. John Helferty and Dr. Dennis Silage for their motivation, insight, and inspiration throughout my academic career at Temple University.

In addition, I would like to thank my labmates in CSNAP. In particular, I would like to thank Firdous Saleheen for his friendship and technical assistance, not only during this project, but since first starting at CSNAP.

I am eternally indebted to my four parents Charlene and Michael Perregrino and Donna and Salvatore Giorgi for help, love, and encouragement through this time. Most of all, I want to thank my girlfriend, Kathryn Moran, for loving and supporting me throughout the development of this work.

Finally, I would like to extend my thanks for the financial support from National Science Foundation Grant AIS ECCS-0969430, REU Supplement, and the College of Engineering Teaching Assistant support. Through this teaching assistant support, I was able to grow both as a student and as a mentor.

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Motivations

The motivation for this project is twofold. First, we wish to address the problem of controlling a laser line of sight to follow a particular reference signal. This is done to address the issue of space-based solar power, which is the collection of the sun's energy in space and the wireless transmission from space to earth. This idea was first proposed by P.E. Glaser (Glaser, 1968). The Department of Energy and NASA started the Solar Power Satellite Concept Development and Evaluation Program in 1976 (Dietz et al., 1981). This program developed gimbaled, microwave transmission antenna in space and a ground based rectenna for receiving the transmission. More recently, NASA revisited this concept in order to determine its commercial viability (Mankins, 1997). Fig. 1.1 shows a possible power satellite system. When this problem is viewed from a control system's perspective, we see that power transmission is essentially a pointing and tracking problem.

Second, we wish to examine the resiliency of the Adaptive Neural Control (ANC) system. The resilient concept, which is loosely defined as the system's ability to withstand attacks, has only begun to take shape within the past few years (Rieger, 2010; Rieger and Villez, 2012; Wei and Ji, 2010; Rieger et al., 2009). With the ANC system being developed during the 1990s, the idea of resiliency of this particular control system was not extensively examined. By implementing this system in software and hardware, we can then subject the system to a series of "malicious attacks" and monitor the system's ability to maintain operational normalcy or gracefully degrade in response to the attacks.

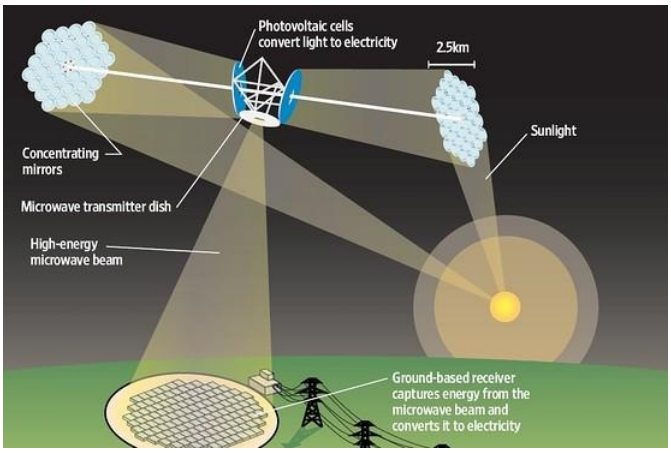

Figure 1.1: Solar Power Satellite System

1.2 Problem Statement

This thesis will investigate the control capabilities of the ANC system when applied to the GLTS. The controllers performance will be quantified through two control metrics and four resilient metrics. The control metrics are root mean square point error and standard deviation of point error. The resilient metrics are recovery time, performance degradation, protection time, and degrading time.

The robustness, adaptiveness, and optimality of the ANC system were first studied through hardware experiments, and later through analytical results. The resiliency of the controller was not. Preliminary resiliency results have shown the system to be resilient to certain attacks, though these results were not quantified through the above metrics (Giorgi et al., 2012). We use a Proportional Integral Derivative (PID) controller for a baseline performance. Since this type of controller does not consider a stochastic system, we expect their performance to degrade when subjected to disturbances and anomalies.

Ultimately, our objective is to show that we will achieve better control performance and resiliency with the ANC system.

1.3 Contributions

The major contributions of this investigation will be as follows:

- Develop a software implementation of the ANC system in Matlab/Simulink.
- Simulate system replication and control of the GLTS and a nonlinear gimbal model using the ANC system software.
- Examine the resiliency of the ANC system to added latencies, plant parameter changes, false data injection, and sensor data alterations.
- Examine the resiliency of ANC system through resilient metrics: recovery time, performance degradation, protection time, and degrading time.
- Develop a hardware model of the ANC system for implementation on an FPGA using Xilinx/System Generator.
- Perform software simulations, as well as "hardware in the loop" simulations (where the FPGA is used within the simulation), of the hardware model for system replication and control of the GLTS model using the hardware model of the ANC system.
- Implement a Replicator Unit on a sequential processor, via a dSPACE control board, and an FPGA for real time system identification of the GLTS hardware test bench output.

1.4 Thesis Organization

This document is composed of five chapters. Chapter 2 gives a brief background on the theory behind neural networks and issues surrounding FPGA implementation of neural networks. Next, we outline the theory of Model Reference Adaptive Control and Neural Control. We then give a brief background on adaptive neural networks for control. This chapter then gives a brief description of the gimbaled laser targeting system, where an overview of the entire system is given. Finally, we give a linearized model of the GLTS, which is then decoupled giving linear models for both the pitch and yaw axis.

Chapter 3 provides a detailed description of the ANC system. In this chapter, first an overview of the neural architecture is given. Then, each subsystem is presented in detail. The major subsystems are the single neuron, ganglia, Replicator Units, and the control architecture.

The adaptive update laws, as well as convergence results, are given. Next, we simulate control of the linear stochastic GLTS model, using both a PID controller and the ANC system. Also, we simulate control of a separate nonlinear stochastic two axis gimbal model using the ANC system. Finally, we implement the ANC system in hardware using a dSPACE control board and preform real time system replication of the GLTS test bed.

Chapter 4 introduces Resilient Control. First, we discuss the theory and methodologies of Resilient Control. Next, we present metrics by which we will measure the ANC system's resiliency. Finally, we examine the ANC system's resiliency through simulations. Here, we subject the model of the GLTS to four types of attacks: added latencies, false data injection, sensor data alteration, and plant parameter changes. Again, we use a PID controller for a baseline performance comparison.

Chapter 5 outlines our implementation of the ANC system on an FPGA. First, we introduce computational preliminaries by discussing our floating point to fixed point conversion and how we handle division and the nonlinear functions within the neural network. Next, we preform simulations and "hardware in the loop" simulations of our division, nonlinear neural functions, various Replicator Unit designs, and control of the linear GLTS model using the ANC system.

Chapter 6 considers the problem of hardware implementation of the ANC system for real time system replication of the GLTS test bed. We first consider real time data transfer and shared memory between the FPGA and the PC running our image processing algorithm. Then, as a simple hardware example, we implement a proportional controller on the FPGA and compare this performance against that of the dSPACE proportional controller. Next, we implement a single Replicator Unit on the FPGA for real time system identification. This, again, is compared to a similar Replicator Unit running on the dSPACE board. This chapter concludes with a discussion of the hardware resources needed for our Replicator Unit implementation.

Finally, Chapter 7 provides a summary of the major results and our conclusions. We finish this chapter with possible solutions to the problem of a full hardware implementation of the ANC system for real time control.

CHAPTER 2

LITERATURE REVIEW AND BACKGROUND

In this chapter we give the necessary background information for this thesis. First, we discuss neural networks and define their associated terminology. We also discuss problems associated with implementation of neural networks on FPGAs. Neural networks for control are then discussed. Next, we briefly define Model Reference Adaptive Control. Finally, we describe the laser targeting system test bench. An overview of the entire system is given, with details of each subsystem. We conclude this chapter with a linearized system model, which will be used for simulations.

2.1 Neural Networks

The ANC system sits within the intersection of neural network control, neural network pattern recognition, and model reference control, as seen in Fig. 2.1. Since the ANC system is a specific neural network architecture, it is important to first understand how neural networks work and define terms commonly used in neural network literature. Therefore, in this section, we define a general neural network and show how it is used for replication and control, whereas, the ANC system is treated in great detail in Chapter 3. We also introduce parallelisms within neural networks and examine how FPGAs have been used in the past to implement neural networks.

It is possible to think of a neural network as function which is formed from compositions and superpositions of neural functions. Thus, the output of a neural network is the value of the function that results from that particular composition and superposition of the neural function (Cybenko, 1989). Therefore, if we can represent the plant (either exactly or approximately) by a finite combination of weighted neural functions, we can use a neural network to identify

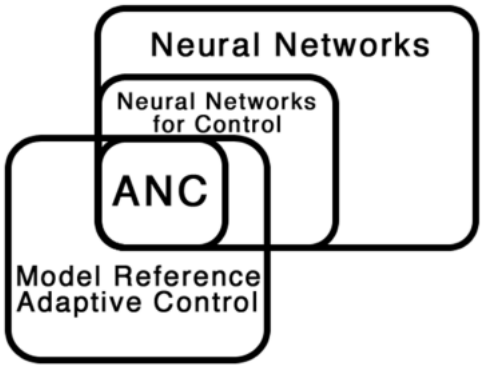

Figure 2.1: The ANC System Within a General Neural Network Architecture

and control the plant. In the case of linear systems, or nonlinear systems whose linearizations around an equilibrium point are well behaved, we can exactly represent the plant dynamics with this type of sum (Narendra, 1996). In the case of a certain nonlinear systems, specifically, any system modeled by a continuous function of n variables with support on a unit hypercube, we can always approximate the plant dynamics with this type of sum (Cybenko, 1989).

2.1.1 General Neural Network Architecture

A general neuron has an arbitrary number of inputs and a single output, as seen in Fig. 2.2. All inputs x_i are multiplied by an associated weight w_i and are summed together with a bias θ. The neuron then takes this sum and propagates it to the output y via a neural function f_{trans}. Thus, each neuron can be viewed as the following function

$$y = f_{trans}(\sum_{i=1}^{n} x_i w_i + \theta). \tag{2.1}$$

The most common transfer functions are the linear, threshold, and sigmoid functions. The linear transfer function is the simplest case and is usually the identity map. The threshold function, also called the hard limit function, gives a binary output. The sigmoid functions are a class of continuous, differentiable, monotone functions with limited range, usually $[0, 1]$ or $[-1, 1]$ (Lange, 2005). Examples are the logsig and tansig functions.

Neural networks usually contain multiple layers of neurons, as seen in Fig. 2.3. A layer consists of a group of neurons which are not interconnected. The simplest neural networks

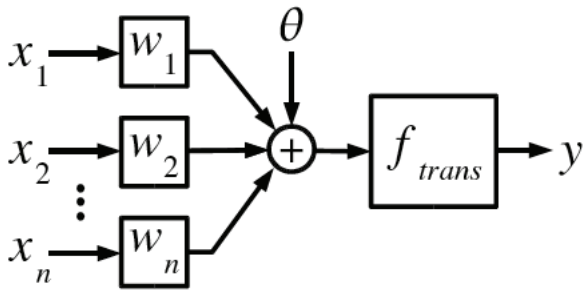

Figure 2.2: General Neuron Architecture

contain two layers, an input and output layer, whereas most networks contain at least one hidden layer.

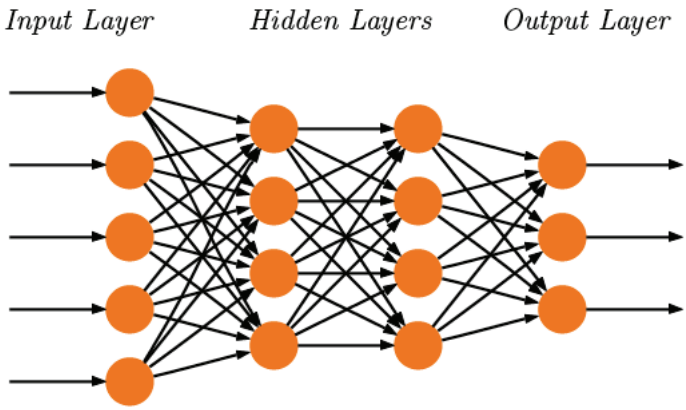

Figure 2.3: Layers of a Neural Network

An important feature of a neural network is its ability to learn. Details on various learning algorithms are detailed in (Anthony, 1999). Learning can be divided into two classes: supervised and unsupervised learning (Lange, 2005). In supervised learning the network is trained via input and output pairs. The difference between the current output and the desired output is what drives the learning process. In unsupervised learning the network is trained only with input, as the output is not known in advance. Here, the learning takes place as the network autonomously reconfigures to classify the input.

One of the more popular types of neural networks is the backpropagation network, first

introduced by Werbos in 1974 (Werbos, 1974). In order to deal with disjointed classification areas, hidden layers are needed (Lange, 2005). But without the aid of backpropagation, there is no method for training the neurons within the hidden layers. The backpropagation method is based on recursive error propagation throughout the network. Each neuron receives an error signal proportional to that neuron's contribution to the total output and the associated weight is adjusted accordingly (Freeman and Skapura, 1991).

2.1.2 Parallelism in Neural Networks

Neural networks exhibit several types of parallelism, and a careful examination of these is required in order to both determine the most suitable hardware structures as well as the best mapping from the neural network structures onto given hardware. The types of parallelism are (Omondi and Rajapakse, 2006):

- **Layer parallelism**: When using a multilayer neural network, the different layers can be processed in parallel. Typically, each layer contains tens of neurons, and therefore the value of layer parallelism is less significant than other parallelism.

- **Training parallelism**: Multiple training sessions can be run in parallel. In general, this type of parallelism results in hundreds of neuronal processes executing simultaneously, and therefore this type of parallelism is of medium importance.

- **Node parallelism**: This level of parallelism deals with individual neurons. It is the most important level of parallelism, since, if one can attain node parallelism, then all parallelisms at a higher level can be attained. Generally, the number of neurons in a neural network ranges from thousands to millions, and thus, this type of parallelism is difficult to attain.

- **Weight parallelism**: This type of parallelism only considers the multiplication of the neuron's output by a weight.

Two things are apparent from the above. First, different types of parallelism are better suited for different hardware implementations. For example, because of routing connections,

weight parallelism might not be suited for implementation on an FPGA. Indeed, most FPGA implementations first consider the problem of minimizing the number of neurons available to the network in order to minimize the number of interconnections (Omondi and Rajapakse, 2006). Second, the importance of each type of parallelism depends on the overall architecture of the neural network. For example, the ANC system contains five separate neural networks, and, thus, training parallelism should be high priority.

2.1.3 FPGA Implementation of a Neural Network

Researchers have been implementing neural networks on FPGAs for over two decades; see (Zhu and Sutton, 2003) for a survey. We summarize the main topics presented in the survey with a number of additional sources.

The parallel processing capabilities of an FPGA are often the foremost reason for their use with neural networks (Omondi and Rajapakse, 2006). How to properly take advantage of the FPGA parallelism is discussed in Chapter 2.

Another one of the main reasons for using an FPGA to implement a neural network in hardware is the reconfigurability of an FPGA. Since each neural network varies in terms of number of neurons, layers, learning algorithms, back propagation, and so on, reconfigurable hardware allows for rapid prototyping of an unlimited number of neural architectures.

Topology adaptation is another benefit of FPGA implantation. Often, during training and unsupervised learning, a neural network will modify its architecture in order to classify certain inputs. This change in topology can be attained through the FPGA's ability to reconfigure.

Many neural networks often use integer weights for training purposes (Jian et al., 2012; Plagianakos and Vrahatis, 1999). This is done since integer multiplication can be executed faster in fixed point notation than floating point. One study develops a learning algorithm where all computation use only numbers which are powers of two (Marchesi et al., 1993).

While there are significant advantages to implementing the neural network on an FPGA, it is not without its drawbacks. The first main issue is the precision of fixed point arithmetic. Usually, the designer must compromise weight precision with implementation cost (Zhu and

Sutton, 2003). Holt and Baker performed a minimum precision study for a class of benchmark classification problems (Holt and Baker, 1991). They found that in order for the neural network to maintain proper learning capabilities, the minimum allowable fixed point representation is 16-bits.

Implementation of the neural function is also an issue, especially when considering a nonlinear sigmoid function. It is often not possible to represent a nonlinear function exactly in hardware. This, combined with limited fixed point precision, can often introduce significant errors. One way to mitigate this error is to represent the nonlinear function with a lookup table (Colavito and Silage, 2009), though this significantly increases the amount of hardware needed to process the neural network.

2.2 Adaptive Neural Control

It is important to distinguish between Adaptive Neural Control and the Adaptive Neural Control system. In general, Adaptive Neural Control is the control of a dynamic system with the aid of a neural network (Hagan and Demuth, 1999). The ANC system, on the other hand, is a specific type of Adaptive Neural Controller, with constrained neural connections and weight update law, developed by D.C. Hyland. To avoid confusion, we will refer to Adaptive Neural Control, the control technique, as Adaptive Neural Control, and we will refer to Hyland's specific design as the ANC system.

Within Adaptive Neural Control there are multiple control frameworks, such as Model Reference Control, Model Predictive Control, Adaptive Inverse Control, etc. (Hagan and Demuth, 1999). In all of these systems a minimum of two neural networks are used, one for replicating the plant and another for control. We note that these frameworks say nothing of the type of neural networks being used, weight update laws, backpropagation, etc. They only determine the configuration of the neural network within the control system. The ANC system uses a Model Reference Control framework.

2.2.1 Adaptive Neural Control of a Gimbaled Laser Targeting System

While the ANC system has not been thoroughly applied to the problem of controlling the line of sight of a gimbaled laser, Adaptive Neural Control has been used for this type of problem in the past. One study uses a feedback Adaptive Neural Control system for a passive line-of-sight stabilization system with a 2 degree of freedom gimbal (Lee et al., 1998). Another article (Lin and Hsiao, 2001) uses a feedforward Adaptive Neural Controller to control a gimbaled antenna for disturbance rejection of a guided missile. Also, (MacKunis et al., 2008) uses an Adaptive Neural Controller for satellite attitude control. In addition to the ANC system not being applied to this type of problem, little effort has been made to study the resiliency of both Adaptive Neural Control and the ANC system.

2.2.2 Resilient Control within Adaptive Neural Control

Preliminary studies of the ANC system's resiliency were done (Giorgi et al., 2012). We restate those results here. In that study, we simulate an attack on a nonlinear plant whose model is given by

$$\dot{x}(t) = -f[x(t)] + u(t) \tag{2.2}$$

$$f[x(t)] = 2x(t) + 0.8x^3(t), \tag{2.3}$$

where $u(t)$ is the input to the plant, with the additional assumption

$$y(t) = x(t). \tag{2.4}$$

We simulate a Plant Parameter Change, Sensor Data Alteration, and False Data Injection attacks and the results are seen in Figs. 2.4, 2.5, and 2.6, respectively. For this study, we did not use any metrics to quantify the resiliency, but one can get a qualitative notion of resiliency through the figures. In each case, the ANC system was able return the system to operational normalcy within a finite amount of time.

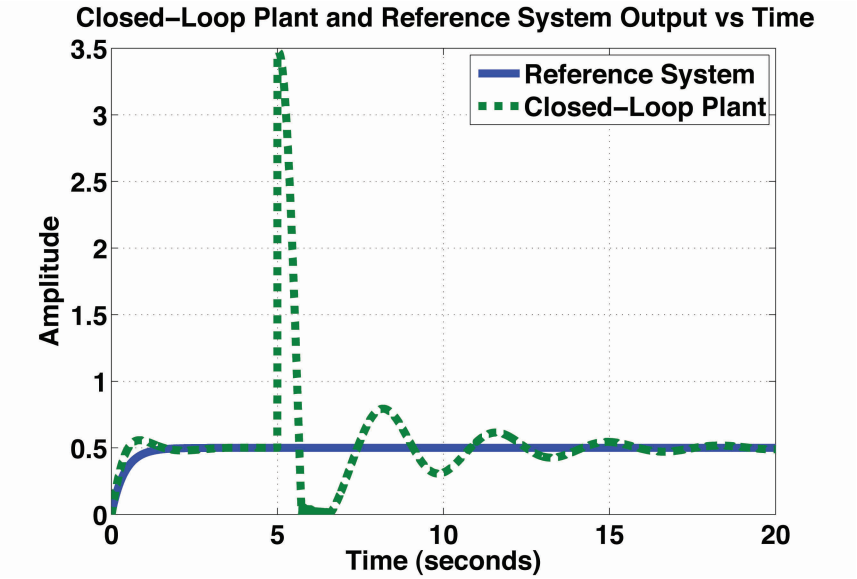

Figure 2.4: Plant Parameter Change Attack

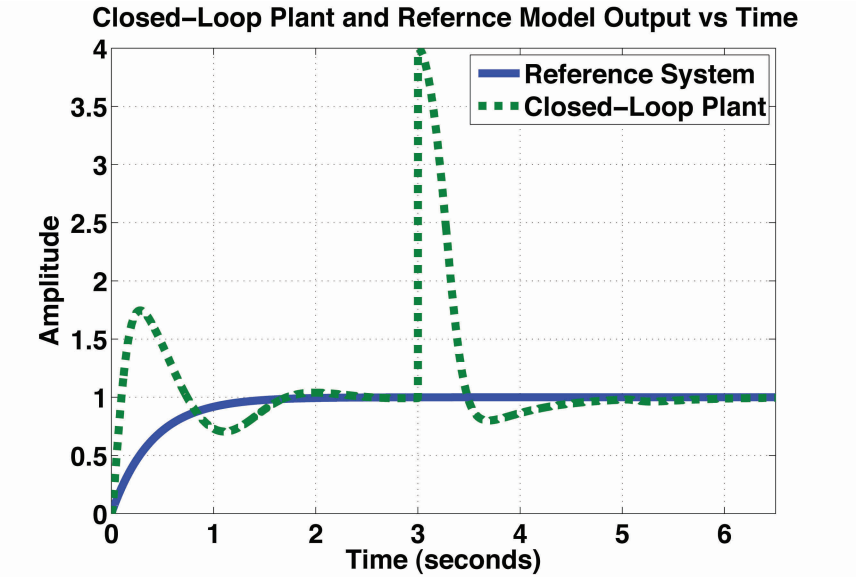

Figure 2.5: Sensor Data Alteration Attack

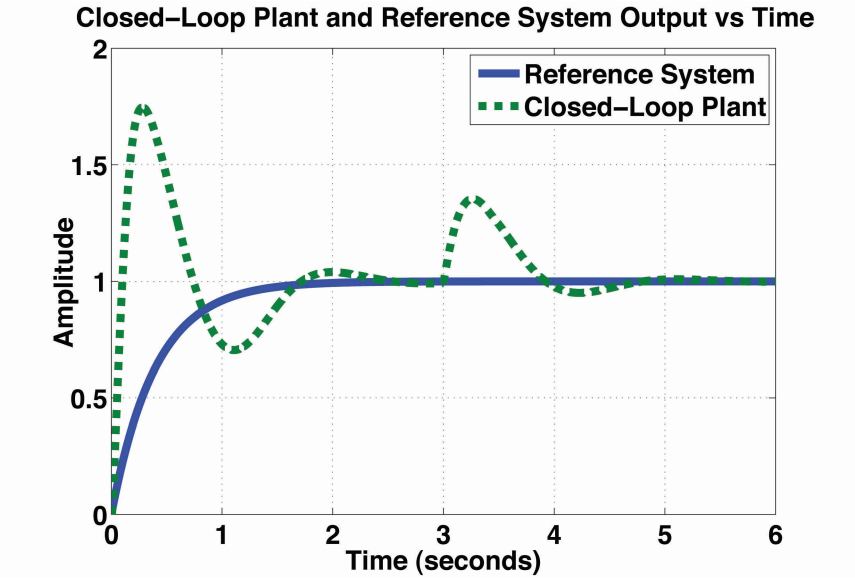

Figure 2.6: False Data Injection Attack

2.3 Model Reference Adaptive Control

In Model Reference Adaptive Control (MRAC) the desirable dynamic characteristics of the plant are specified in a reference model and the input signal or the controllable parameters of the plant are adjusted so that the plant's response will duplicate that of the ideal model (Hang and Parks, 1973). The two major designs within the MRAC paradigm are the M.I.T. design rule (Osburn et al., 1961) and the Lyapunov Synthesis (Parks, 1966). The first MRAC design was a performance index minimization method proposed at the M.I.T. Instrumentation Laboratory and is now referred to as the M.I.T. design rule. The main drawback of this method is that it doesn't guarantee global stability. The Lyapunov Synthesis method uses Lyapunov's second method to guarantee stability (Butchart and Shackcloth, 1966). Though this method addresses the issue of stability, it is not without its limitations, as the entire state vector must be available for measurement.

More recently, neural networks have been used with the MRAC framework (Hagan and Demuth, 1999). One sample architecture is seen in Fig. 2.7. While not the exact architecture of the ANC system it does highlight the most important features found in many neural network based MRAC systems. A critical feature is the reference model. As stated above, the goal of

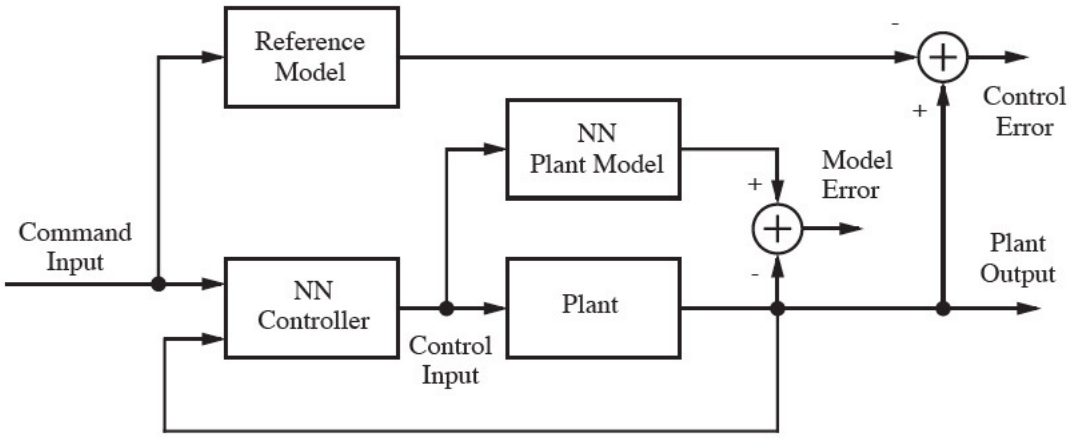

Figure 2.7: Model Reference Adaptive Control Architecture

MRAC is to control the plant's dynamics to match that of an ideal system. The same input is fed into the reference model and the controller, which in Fig. 2.7 and in our case is a neural network. The output from the reference system is then subtracted from the output of the plant, resulting in the control error. This error is used to drive the adaptation of the neural network controller.

A second neural network is used to replicate the plant, as it is often the case that we have no prior modeling information. Similar to the above, we feed the same input into the plant and the neural network replicating the plant. The two outputs are then compared resulting in the model error. It is this error that drives the adaptation of the neurons in this particular neural network.

Therefore, within the neural network MRAC framework, we have a minimum of two simultaneously adapting neural networks, one of which replicates the plant while the other controls the plant to match the reference model. Chapter 3 gives the complete details on the specific MRAC architecture of the ANC system.

2.4 Laser Targeting System: Hardware Description

The entire hardware setup of the laser targeting test bench can be seen in Fig. 2.8. Starting from the top middle and going in a clockwise direction, it consists of a two axis gimbal mounted on a disturbance table, a receiving screen with CCD camera, image processing running in real

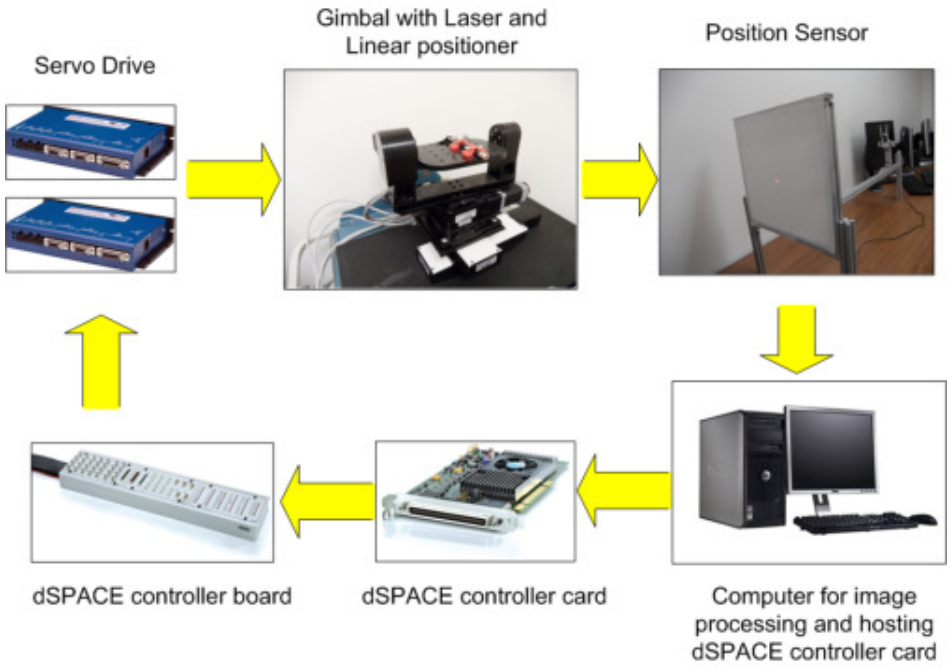

Figure 2.8: Laser Targeting System Hardware

time with Matlab, a dSPACE processor and control board and a servo amplifier. Position information is fed into the computer via images from a web camera. The computer then processes these images to determine the location of the laser on the screen. This information, in terms of a x and y coordinate, in cm, is then sent to the control algorithm running on the dSPACE board, which in turn powers the amplifiers.

The gimbal assembly and laser receiver subsystems are combined into a plant model shown in Fig. 2.9. Specifically, the plant model consists of a servo drive, brushless dc motor, gimbal dynamics, laser pointer kinematics and computer vision based position sensor. The position sensor's output $f_y(k), f_x(k)$ is the position, measure in cm, of the laser dot. A nonlinear model was derived and linearized in (Salaheen, 2013). The basic linear model is given below.

2.5 Laser Targeting System: Model

This model was found using (Ekstrand, 2001; Kennedy and Kennedy, 2003; Krishnan, 2001). After linearizing we have the following model. The full details of this are given in

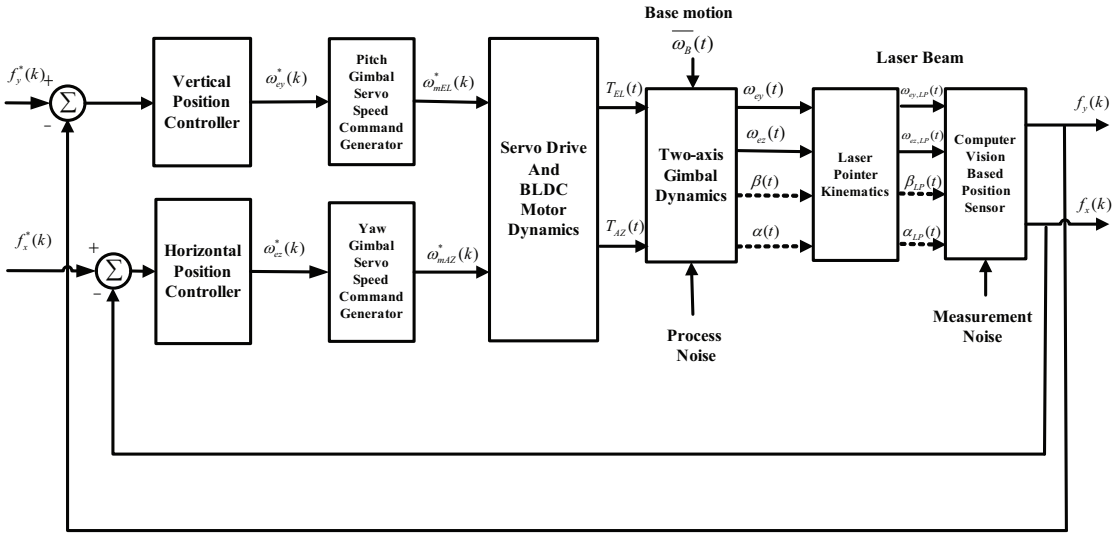

Figure 2.9: Gimbal Plant Model

(Salaheen, 2013). The system is modeled as a linear stochastic differential equation

$$dx(t) = (Ax(t) + Bu(t))dt + Gdw(t), \tag{2.5}$$

The state matrix A is

$$A = \begin{bmatrix} 0 & C_y & 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{K_{e\omega}}{C_y J_{ey}} & -\frac{K_{ef}}{J_{ey}} & \frac{k_b N}{J_{ey}} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{\tau_{iEL}} & \frac{K_{iEL}}{\tau_{iEL}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{\tau_{sEL}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & C_x & 0 & 0 \\ 0 & 0 & 0 & 0 & -\frac{K_{a\omega}}{J_a} & -\frac{K_{af}}{J_a} & \frac{k_b N}{J_a} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{\tau_{iAZ}} & \frac{K_{iAZ}}{\tau_{iAZ}} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{\tau_{sAZ}} \end{bmatrix}, \tag{2.6}$$

with state vector,

$$x(t) = \begin{bmatrix} f_y(t) & \omega_{ey}(t) & i_{aEL}(t) & i^*_{aEL}(t) & f_x(t) & \omega_{ez}(t) & i_{aAZ}(t) & i^*_{aAZ}(t) \end{bmatrix}^T. \tag{2.7}$$

The states are as follows:

- $f_y(t)$ = vertical position co-ordinate on screen,

- $\omega_{ey}(t)$ = pitch gimbal angular rate,
- $\omega_{ey}(t)$ = pitch gimbal angular rate,
- $i_{aEL}(t)$ = armature current for servo-motor drive for pitch gimbal,
- $i^*_{aEL}(t)$ = reference current command for servo-motor drive for pitch gimbal,
- $f_x(t)$ = horizontal position co-ordinate on screen,
- $\omega_{ez}(t)$ = cross-elevation gimbal angular rate,
- $i_{aAZ}(t)$ = reference current command for servo-motor drive for pitch gimbal,
- $i^*_{aAZ}(t)$ = reference current command for servo-motor drive for yaw gimbal.

The initial conditions of the horizontal and vertical position are chosen arbitrarily as $(0, 0)$; all other states are zero initially. This gives us the following initial conditions

$$x(0) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T \tag{2.8}$$

The input matrix B is given as

$$B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \frac{K_{sEL}C_{\omega EL}}{\tau_{sEL}} & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & \frac{K_{sAZ}C_{\omega AZ}}{\tau_{sAZ}} \end{bmatrix} \tag{2.9}$$

with input

$$u(t) = \begin{bmatrix} f^*_y(t) \\ f^*_x(t) \end{bmatrix}. \tag{2.10}$$

Here $f_y^*(t)$ and $f_x^*(t)$ are vertical and horizontal reference position co-ordinates, respectively.

Finally, $w(t)$ is the process noise, a vector of Wiener process or Brownian motion. It is introduced by the disturbance torque present in system dynamics. It follows that $dw(t)$ is a Gaussian random process with zero mean and covariance matrix Wdt. Additionally, we have

$$G = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & J_{ey}^{-1} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & J_a^{-1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{2.11}$$

The outputs of the system are $f_y(t)$ and $f_x(t)$, the vertical and horizontal position screen co-ordinates, respectively. The output equation is

$$y(t) = Cx(t) + v(t), \tag{2.12}$$

where,

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}. \tag{2.13}$$

Here $v(t)$ is the measurement noise, introduced during image processing, which is modeled as a Weiner process.

Thus, the linear model of the two-axis gimbaled laser targeting system is expressed in Eq. (2.5) and (2.12). This system can be decoupled into models for both the pitch and yaw gimbals. The pitch and yaw models will be indicated by 'EL' and 'AZ' subscript, respectively.

Linear Pitch GLTS Model

The pitch model can be written as

$$dx_{EL}(t) = (A_{EL}x_{EL}(t) + B_{EL}u_{EL}(t))dt + G_{EL}dw_{EL}(t), \tag{2.14}$$

with state matrix

$$A_{EL} = \begin{bmatrix} 0 & C_y & 0 & 0 \\ -\frac{K_{e\omega}}{C_y J_{ey}} & -\frac{K_{ef}}{J_{ey}} & \frac{k_b N}{J_{ey}} & 0 \\ 0 & 0 & -\frac{1}{\tau_{iEL}} & \frac{K_{iEL}}{\tau_{iEL}} \\ 0 & 0 & 0 & -\frac{1}{\tau_{sEL}} \end{bmatrix}, \tag{2.15}$$

and state vector

$$x_{EL}(t) = \begin{bmatrix} f_y(t) & \omega_{ey}(t) & i_{aEL}(t) & i^*_{aEL}(t) \end{bmatrix}^T. \tag{2.16}$$

Here the states are

- $f_y(t)$ = vertical position co-ordinate on screen,
- $\omega_{ey}(t)$ = pitch gimbal angular rate,
- $i_{aEL}(t)$ = armature current for servo-motor drive for pitch gimbal,
- $i^*_{aEL}(t)$ = reference current command for servo-motor drive for pitch gimbal.

We denote the initial conditions by $x_{EL}(0)$ is the initial condition and note that they are chosen arbitrarily. As above, the vertical position co-ordinate is set to 0 cm and all other states are zero initially, as follows

$$x_{EL}(0) = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}^T. \tag{2.17}$$

The input matrix B_{EL} is written as

$$B_{EL} = \begin{bmatrix} 0 & 0 & 0 & \frac{K_{sEL} C_{\omega EL}}{\tau_{sEL}} \end{bmatrix}^T \tag{2.18}$$

with input

$$u_{EL}(t) = \begin{bmatrix} f^*_y(t) \end{bmatrix}, \tag{2.19}$$

with $f^*_y(t)$ the vertical reference position co-ordinate.

The process noise $w_{EL}(t)$ is a Wiener process or Brownian motion and $dw_{EL}(t)$ is a Gaussian random process with zero mean and covariance matrix Wdt.

$$G_{EL} = \begin{bmatrix} 0 & J_{ey}^{-1} & 0 & 0 \end{bmatrix}^T. \tag{2.20}$$

The output $f_y(t)$ is the vertical position co-ordinate of the laser on the screen. The output equation is,

$$y_{EL}(t) = C_{EL}x_{EL}(t) + v_{EL}(t), \tag{2.21}$$

with

$$C_{EL} = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \tag{2.22}$$

The measurement noise $v_{EL}(t)$ is introduced during image processing and is modeled as a Weiner process.

Linear Yaw GLTS Model

The yaw model can be written as

$$dx_{AZ}(t) = (A_{AZ}x_{AZ}(t) + B_{AZ}u_{AZ}(t))dt + G_{AZ}dw_{AZ}(t). \tag{2.23}$$

Here, the state matrix is

$$A_{AZ} = \begin{bmatrix} 0 & C_x & 0 & 0 \\ -\frac{K_{a\omega}}{C_x J_a} & -\frac{K_{af}}{J_a} & \frac{k_b N}{J_a} & 0 \\ 0 & 0 & -\frac{1}{\tau_{iAZ}} & \frac{K_{iAZ}}{\tau_{iAZ}} \\ 0 & 0 & 0 & -\frac{1}{\tau_{sAZ}} \end{bmatrix}, \tag{2.24}$$

with state vector

$$x_{AZ}(t) = \begin{bmatrix} f_x(t) & \omega_a(t) & i_{aAZ}(t) & i^*_{aAZ}(t) \end{bmatrix}^T. \tag{2.25}$$

The states are as follows:

- $f_x(t)$ = horizontal position co-ordinate on screen,

- $\omega_{ez}(t)$ = cross-elevation (z component of pitch axis) gimbal angular rate,
- $i_{aAZ}(t)$ = reference current command for servo-motor drive for pitch gimbal,
- $i^*_{aAZ}(t)$ = reference current command for servo-motor drive for yaw gimbal.

The initial conditions $x_{AZ}(0)$ are chosen arbitrarily, with the horizontal position equal to 0 cm and all other states zero. This gives us

$$x_{AZ}(0) = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}^T. \tag{2.26}$$

The input matrix is

$$B_{AZ} = \begin{bmatrix} 0 & 0 & 0 & \frac{K_{sAZ}C_{\omega AZ}}{\tau_{sAZ}} \end{bmatrix}^T \tag{2.27}$$

with input

$$u_{AZ}(t) = \begin{bmatrix} f^*_x(t) \end{bmatrix}, \tag{2.28}$$

Here, $f^*_x(t)$ is the horizontal reference position co-ordinates.

The process noise $w_{AZ}(t)$ is a scalar of Wiener process or Brownian motion. This gives us that $dw_{AZ}(t)$ is a Gaussian random process with zero mean and covariance matrix Wdt. Additionally,

$$G_{AZ} = \begin{bmatrix} 0 & J_a^{-1} & 0 & 0 \end{bmatrix}^T, \tag{2.29}$$

The output $f_x(t)$ is the horizontal position co-ordinate on screen. The system output equation is

$$y_{AZ}(t) = C_{AZ}x_{AZ}(t) + v_{AZ}(t), \tag{2.30}$$

where

$$C_{AZ} = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}. \tag{2.31}$$

The measurement noise $v_{AZ}(t)$ is introduced during image processing and is modeled as a Weiner process.

All numerical values can be found in the Table 2.1.

Table 2.1: Laser Targeting System Model Parameters

Parameter	Description	Value	Unit
k_b	back emf constant	0.0678	V/rad/s
K_{sEL}, K_{sAZ}	Speed filter gain	10, 40	
τ_{sEL}, τ_{sAZ}	Speed filter time constant	0.0052,0.0052	s
J_{ex}, J_{ey}, J_{ez}	pitch diagonal element(inertia)	0.00565, 0.036725, 0.0226	
J_{ax}, J_{ay}, J_{az}	yaw diagonal element(inertia)	0.0113, 0.0452, 0.0678	
K_{ef}, K_{af}	viscous friction coefficient	0.0006328,0.0006328	Nm/rad/s
N	gear ratio	90	Nm
$K_{e\omega}, K_{a\omega}$	Cable restraint co-efficient	0.000113,0.000113	rad/s

2.6 Laser Targeting System: Control Objective

Our control objective is as follows. Given a reference signal $f_y^*(k), f_x^*(k)$, in cm, we wish to control the GLTS so that the output $f_y(k), f_x(k)$ matches the reference signal. Specifically, our sensor screen measures 30.3×30.3 cm. Our reference signal will remain constant throughout each simulation and correspond to the center of the screen at $(15.15, 15.15)$. In every control simulation and experiment, we wish to control the laser's position on the sensing screen output to remain at $(15.15, 15.15)$ despite any added noise or disturbances, starting from $(0, 0)$. Fig. 2.10 shows the exact configuration of the sensing screen.

2.7 Summary

Now that we have defined neural networks, Adaptive Neural Control, and Model Reference Control, in addition to defining our system model and outlining our control objective, we are now in a position to give a full description of the ANC system. The next chapter will give a detailed explanation of each subsystem of the ANC system and show its control capabilities through simulation and hardware implementation.

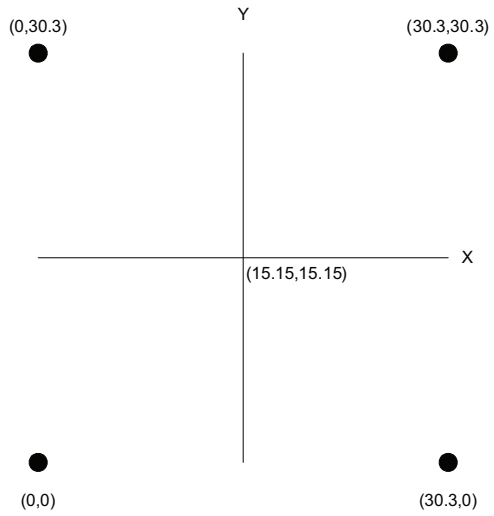

Figure 2.10: Sensor Coordinate System

CHAPTER 3

ADAPTIVE NEURAL CONTROL: THEORY AND SIMULATIONS

In this chapter we briefly outline the hierarchy of the ANC system (Hyland, 1991; Hyland, 1998; Hyland, 1995; Davis and Hyland, 1997). Then each subsystem within the ANC hierarchy is discussed in detail. The main subsystems within the hierarchy are individual neurons, synaptic connectors, memory units, ganglia, Toeplitz synapses, Replicators Units, and the ANC system. Next, the weight update law is presented along with convergence results. Then we present control simulations using the ANC system with the linear GLTS model. Here the control performance is compared against a PID controller. Finally, we implement the ANC system in hardware and control the GLTS test bench, comparing our results to a Proportional (P) controller.

3.1 Hierarchy of Control System

The ANC System can be viewed as a hierarchical modular system with each new level being built of pieces from the previous level, as shown in Fig. 3.1. The lowest level contains three devices: a memory unit, an individual neuron, and a synaptic connector. The next level up in the hierarchy consists of dynamic ganglia, or groups of neurons, and Toeplitz synapses, or constrained groups of synaptic connectors. Next are Replicator Units which are sets of ganglia. Finally, at the top of Fig. 3.1, is the ANC system, which consists of several Replicator Units.

3.2 Memory Units, Individual Neurons and Synaptic Connectors

A memory unit takes a scalar time-series input and produces an L-dimensional vector consisting of the current value of the input and the *L-1* delayed values. The training signal ϕ is first

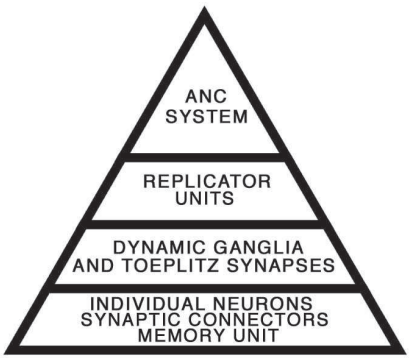

Figure 3.1: Hierarchy of Adaptive Neural Control System

sent through the memory unit and results in the output, for $n = 0, 1, 2, ..., N$,

$$\bar{\phi}(n) = \begin{bmatrix} \phi(n) \\ \phi(n-1) \\ \vdots \\ \phi(n-L-1) \end{bmatrix}. \tag{3.1}$$

Throughout the paper we will use a bar notation (e.g. $\bar{\lambda}$) to denote a column vector with *L* past signals augmented as in (3.1).

Each neuron is defined as in Fig. 3.2; a dual channel device with a forward signal flow path and a backward signal flow path. Both the forward and backward signal flow paths consist of a sum of a series of input signals and a bias signal. The input signals are usually received from other neurons via synaptic connectors. The neuron sums *K* forward path inputs $(x_1, x_2, ..., x_K)$ and the forward bias I_k to form α_k which is then operated on by a neural function $g(\cdot)$ (usually a linear or sigmoid function). Here $k = 1, ..., K$ and represents the k-th neuron. The output of the neural function forms the output of the forward path of the neuron y_k. The signal α_k is also sent into the backward path of the neuron, where it is operated on by the derivative of the neural function $g'(\cdot)$. This gives us the following

$$y_k = g(\sum_{i=1}^{K} x_i + I_k). \tag{3.2}$$

In the backward flow path, the *K* backward inputs $(x_1^*, x_2^*, ..., x_K^*)$ are added to a backward path bias ϵ_k. This sum is then multiplied by the derivative of the neural function evaluated at

α_k, resulting in the output of the backward flow path y_k^*. This gives us the following

$$y_k^* = (\sum_{i=1}^{K} x_i^* + \epsilon_k) g'(\sum_{1}^{K} x_i + I_k). \tag{3.3}$$

Fig. 3.3 is a simplified version of Fig. 3.2 and shows the *K*-th input and output of the *K*-th neuron. The hexagon represents everything within the forward path and backward path blocks in Fig. 3.2. Only the forward path signals are shown explicitly (all backward path signals are inferred). As a rule, forward path bias signals are shown entering the top of the hexagon and backward path bias signals are shown entering the bottom.

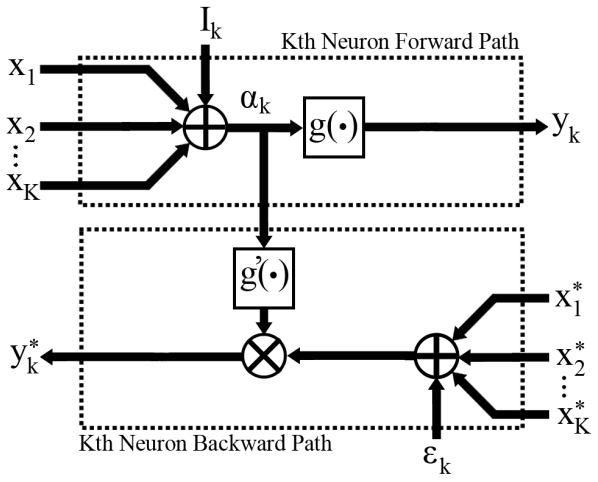

Figure 3.2: Explicit Neuron Diagram

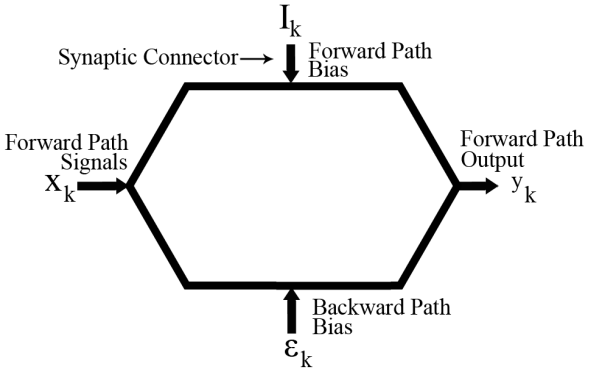

Figure 3.3: Simplified Neuron Diagram

Figure 3.4 shows a single synaptic connector with weight adaption law. A synaptic connector is a connection between the output of one neuron and the input of a separate neuron. Each synaptic connector has an associated scalar weight $w(n)$ at any instant n, which is multiplied with the output from the neuron. This product is then fed into the input of the next neuron. Defined this way, the neuron itself stays static and only the synaptic weights change. The synaptic connectors are also two-way devices, with a forward and backward path input and a forward and backward path output. Both the forward and backward path inputs are multiplied by the weight and the resulting products form the forward and backward path outputs, respectively.

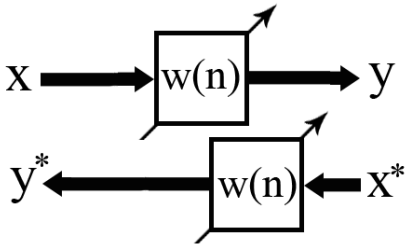

Figure 3.4: Synaptic Connector

Fig. 3.5 shows a simplified synaptic connector. Again, only the forward paths are shown, while the backward paths are inferred.

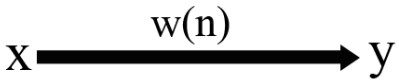

Figure 3.5: Simplified Synaptic Connector

3.3 Dynamic Ganglia

Next in the hierarchy are the dynamic ganglia, or ganglia, and Toeplitz synapses, both seen in Fig. 3.6. The ganglia, which are collections of neurons, are shown within the dotted lines. In this figure there are four neurons per ganglia, but in general there is no limit to the number of neurons contained within a single ganglia. Toeplitz synapses are constrained groups of synaptic connectors and they are shown within the solid box in Fig. 3.6. Specifically, Toeplitz synapses

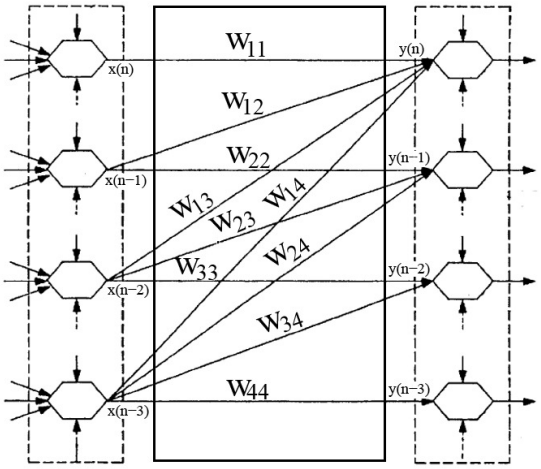

Figure 3.6: Ganglia with Toeplitz Synapse

constrain connections in the following way: any neuron m places from the top of the ganglia receive signals from all neurons m or more places from the top of the adjacent ganglia. It is important to note that any set of synapses constrained in such a way is considered one Toeplitz synapse. For example, the single synapses originating from $x(n), x(n-1), ..., x(n-3)$ and terminating at $y(n)$ are all considered one Toeplitz synapse. Sets of Toeplitz synapses are constrained by Toeplitz matrices (3.4) and the numbering for each synaptic weight uses the following convention: weight w_{ij} is the weight associated with the output from the j-th neuron in the first ganglia going to the i-th neuron in the second ganglia.

$$W = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1K} \\ 0 & w_{22} & \dots & w_{2K} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & w_{KK} \end{bmatrix} \tag{3.4}$$

These types of interconnection impose a temporal ordering and causality on the neurons. The position of the neurons determine the "age" of the data, where top level neurons represent current data and lower level neurons represent past data. Since top level neurons do not feed signals into lower level neurons, past data points do not depend on future inputs. Therefore, these types of constrained interconnections are particularly suited for replicating causal systems.

In view of Fig. 3.6 we get the following equation:

$$\begin{bmatrix} y(n) \\ y(n-1) \\ y(n-2) \\ y(n-3) \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ 0 & w_{22} & w_{23} & w_{24} \\ 0 & 0 & w_{33} & w_{34} \\ 0 & 0 & 0 & w_{44} \end{bmatrix} \begin{bmatrix} x(n) \\ x(n-1) \\ x(n-2) \\ x(n-3) \end{bmatrix}. \tag{3.5}$$

Since each synaptic connector is a two-way device, it is important to examine how the backward path signals are constrained. In light of Figures 3.4 and 3.5, each arrow in Figure 3.6 contains a forward path signal and a backward path signal. Therefore Figure 3.6 can also be viewed as Figure 3.7.

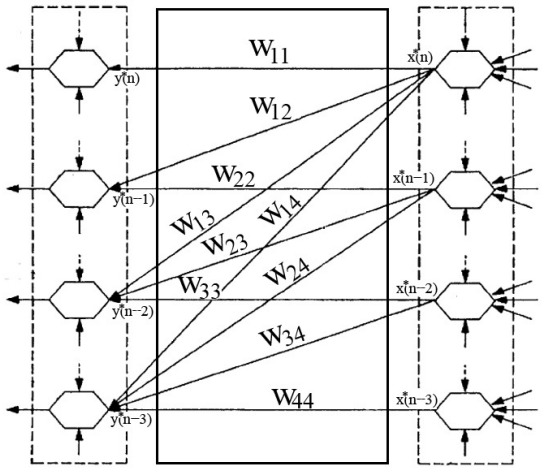

Figure 3.7: Ganglia with Toeplitz Constrained Backward Paths

In Fig. 3.7, the two ganglia are connected via weighted synapses exactly as in Figure 3.6. The only difference between the two figures is the direction of the synapses. Constraining the synapses in matrix form leads to:

$$\begin{bmatrix} y^*(n) \\ y^*(n-1) \\ y^*(n-2) \\ y^*(n-3) \end{bmatrix} = \begin{bmatrix} w_{11} & 0 & 0 & 0 \\ w_{12} & w_{22} & 0 & 0 \\ w_{13} & w_{23} & w_{33} & 0 \\ w_{14} & w_{24} & w_{34} & w_{44} \end{bmatrix} \begin{bmatrix} x^*(n) \\ x^*(n-1) \\ x^*(n-2) \\ x^*(n-3) \end{bmatrix}. \tag{3.6}$$

Thus, the backward path signals are constrained by the transpose of the Toeplitz weight matrix in (3.5).

Fig. 3.8 shows a simplified version of Fig. 3.6. Ganglia are denoted by the double hexagon and double lined arrows denote sets of Toeplitz synapses. Again, only forward path signals are shown.

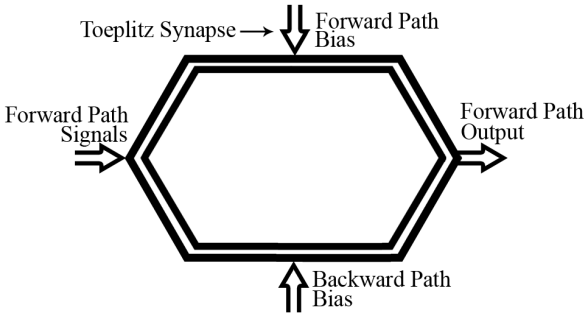

Figure 3.8: Simplified Ganglia with Toeplitz Synapse

3.4 Replicator Unit

The ganglia and Toeplitz synapses are then combined into Replicator Units, which are used to replicate unknown system. Depending on the structure of the system, some combination of either linear / nonlinear and infinite impulse response (IIR) / finite impulse response (FIR), we must use different Replicator Unit configurations. Each configuration is discussed below.

3.4.1 Linear / Nonlinear FIR

The output of a general FIR system can be written as follows

$$y(n) = \mathcal{F}\big(\xi(n), \xi(n-1), \cdots, \xi(n-N)\big), \tag{3.7}$$

were $\mathcal{F}$ is some unknown function, $\xi(n)$ is the system input at time $t = n$ and $N \in \mathbb{Z}$. If $\mathcal{F}$ is linear, then (3.7) can be written as a linear combination of weighted inputs as follows

$$y(n) = \sum_{i=1}^{L} \mathcal{L}_i \xi(n-i). \tag{3.8}$$

The constants $\mathcal{L}_i \in \mathbb{R}$ in (3.8) are referred to as the Markov parameters of the system.

Replication of the system described by (3.8) reduces to identifying the Markov parameters $\mathcal{L}_i$. The neural architecture needed for this type of replication is shown in Fig. 3.9. To replicate a general unknown FIR system a linear input ganglia, two layers of nonlinear ganglia in the hidden layer, and one linear output ganglia are needed (Hyland, 1997). This architecture is shown in Fig. 3.10. The number of ganglia in the hidden layer is not restricted, though three are shown in the figure for simplicity.

Replication of each type of system follows the same procedures. A series of training impulses ξ are sent into both a memory unit and the unknown system. The output from the memory unit is then fed into the forward path bias of the input ganglia. At the same time, the output from the unknown plant is also fed into a memory unit. The output from the plant's memory unit is then compared to the output from the output ganglia. The difference is then fed into the backward path bias of the output ganglia, which is then back propagated through the ganglia driving the weight update laws.

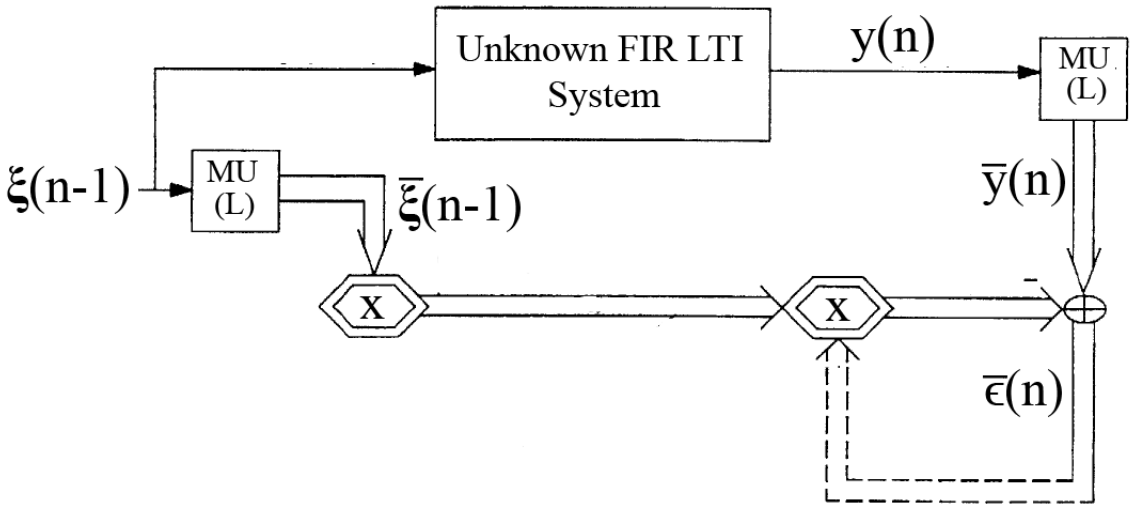

Figure 3.9: Linear FIR System with Replicator Unit

The Replicator Unit is responsible for system replication and is thus a fundamental part of the ANC system. It is important to note the difference between system replication and system identification. By replication we mean to match, as closely as possible, the input and output of

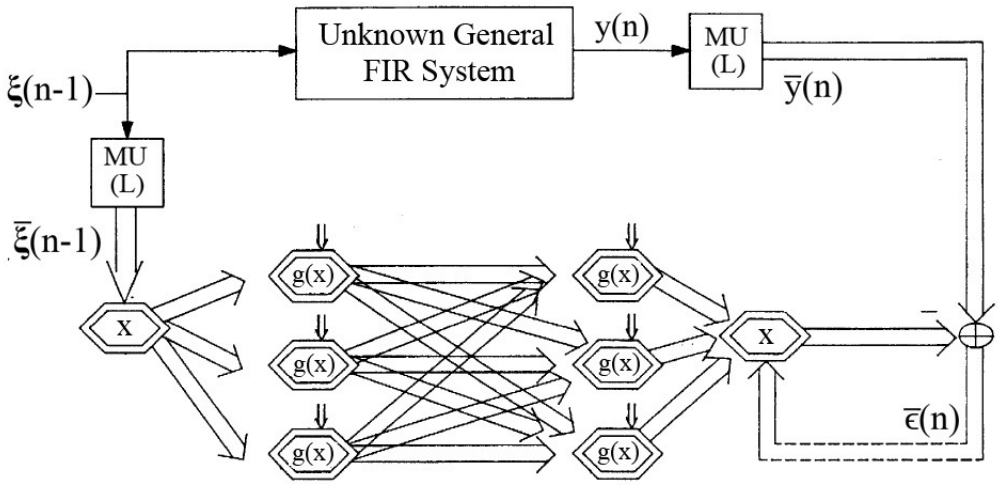

Figure 3.10: General FIR System with Replicator Unit

an unknown system. System identification not only matches input and output but also requires a mathematical model of the system dynamics.

3.4.2 Linear / Nonlinear IIR

First, we note that a general IIR system can be described by

$$y(n) = \mathcal{F}\big(\xi(n), \xi(n-1), \cdots, \xi(n-N), y(n-1), \cdots, y(n-M)\big), \tag{3.9}$$

were $\mathcal{F}$ is some unknown function, $\xi(n)$ is the system input at time $t = n$, and $M, N \in \mathbb{Z}$. When the system is linear (3.9) becomes a linear combination of weighted sums of past inputs and outputs. Replicator Units for linear IIR systems can be based on the following three models (Hyland, 1998). The first, is the Autoregressive Moving Average (ARMA):

$$y(n) = \sum_{i=1}^{M} \mathcal{P}_i y(n-i) + \sum_{j=1}^{N} \mathcal{L}_j \xi(n-j). \tag{3.10}$$

Next, we have the ARMarkov form:

$$y(n) = \sum_{i=1}^{M} \mathcal{P}_i y(n-L-i) + \sum_{j=1}^{N+L} \mathcal{L}_j \xi(n-j). \tag{3.11}$$

We note that the first L $\mathcal{L}_i$ are Markov parameters defined in (3.8). With $L = 0$ (3.11) reduces to (3.10) the ARMA model. The final form is called the Batch ARMarkov form and is given by

$$\overline{y}(n) = \text{Toep}\big[\overline{\mathcal{P}}\big]\,\tilde{y}(n-1) + \text{Toep}\big[\overline{\mathcal{L}}\big]\,\tilde{\xi}(n-1), \tag{3.12}$$

where

$$\overline{y}(n) = \begin{bmatrix} y(n) \\ \vdots \\ y(n-R+1) \end{bmatrix} \in \mathbb{R}^{R\times 1}, \;\; \tilde{y}(n) = \begin{bmatrix} y(n-L-1) \\ \vdots \\ y(n-L-M-R+1) \end{bmatrix} \in \mathbb{R}^{(M+R-1)\times 1}, \tag{3.13}$$

$$\tilde{\xi}(n) = \begin{bmatrix} \xi(n) \\ \vdots \\ \xi(n-L-M-R+1) \end{bmatrix} \in \mathbb{R}^{(L+M+R-1)\times 1},$$

$$\overline{\mathcal{P}} = \begin{bmatrix} \mathcal{P}_1 \\ \vdots \\ \mathcal{P}_M \end{bmatrix} \in \mathbb{R}^{M\times 1}, \;\; \overline{\mathcal{L}} = \begin{bmatrix} \mathcal{L}_1 \\ \vdots \\ \mathcal{L}_{M+L} \end{bmatrix} \in \mathbb{R}^{(L+M)\times 1} \tag{3.14}$$

We define Toep$[W] \in \mathbb{R}^{M\times(M+R-1)}$, for $W = [w_1, w_2, \cdots, w_R]^T \in \mathbb{R}^R$, as follows

$$\text{Toep}[W] = \begin{bmatrix} w_1 & w_2 & \cdots & w_n & 0 & \cdots & 0 \\ 0 & w_1 & \cdots & w_{n-1} & w_n & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & w_1 & w_2 & \dots & w_n \end{bmatrix}. \tag{3.15}$$

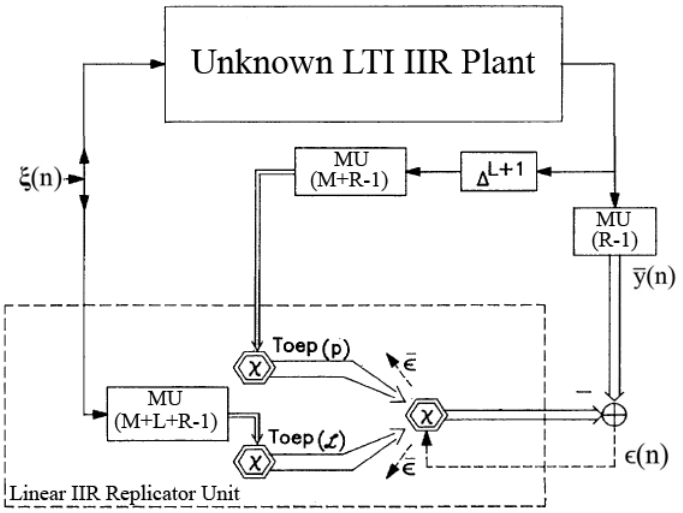

Figure 3.11: General IIR System with Replicator Unit

This Replicator Unit can also be used for nonlinear systems if we make the following modifications. For a linear model, we use three linear ganglia: one input ganglia for the plant output, one input ganglia for the system input, and one output ganglia. If we place a number of hidden layers, made of nonlinear ganglia, in between the input and output ganglia, then we can replicate a nonlinear IIR system. Note, these modifications are essentially the same modifications made to the FIR Replicator Unit above.

Fig. 3.12 shows a simplified version of everything within the dotted line in Fig. 3.10. Each signal is identified by its location: forward path bias signals are shown entering the top, backward path bias signals enter the bottom, forward path input enters the left, forward path output exits the right, backward path input signals enter the right, and backward path output signals exit the left. Usually, only the bias and forward path signals are shown.

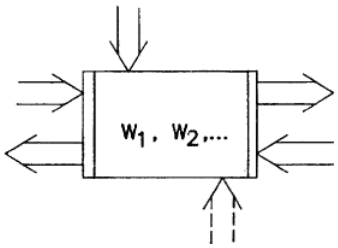

Figure 3.12: Simplified Replicator Unit

3.5 Adaptive Neural Control System

There are two main parts to the Model Reference ANC system in Fig. 3.13 the closed-loop modeler and the control adaptor. Both parts contain a minimum of two Replicator Units, W_M and W_C. Note that each Replicator Unit only shows the input and output layers, the hidden layers are not shown for simplicity. The closed loop modeler uses the training signals $\bar{\xi}$ and the plant sensor outputs to adapt the weights so that the closed-loop system is replicated. When ϵ_M approaches zero the modeler has replicated the plant within the closed-loop system. The control adaptor uses a training signal, its own output, and the output of an ideal reference system to adjust its weights so that the reference system is replicated. Note that in the closed-loop

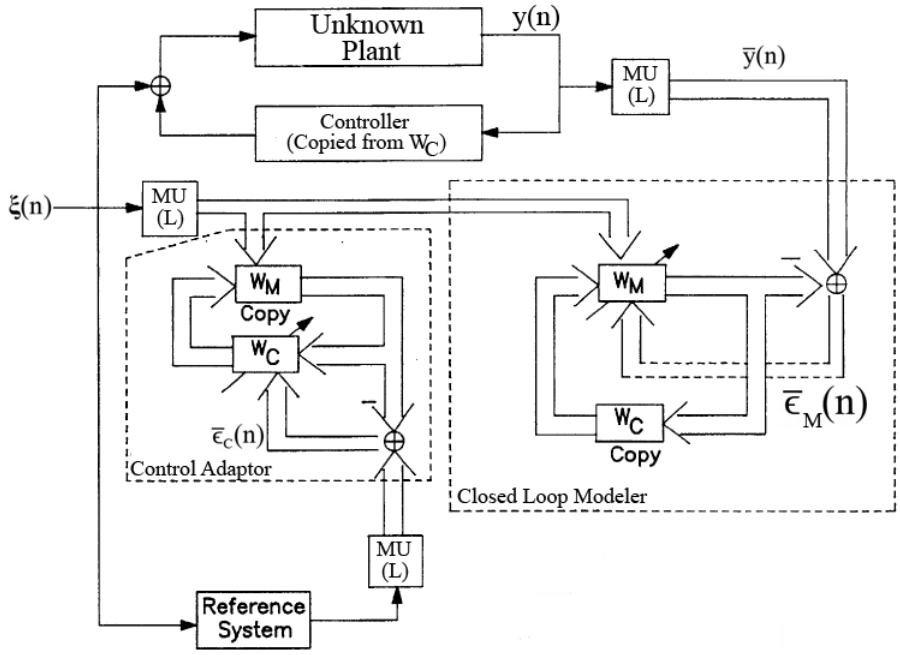

Figure 3.13: Adaptive Neural Control System

modeler, W_C is a copy of the current values being updated in the control adaptor and W_M is left unconstrained. Similarly, in the control adaptor, W_M is a copy of the current values adapted by the modeler and W_C is left unconstrained. Thus, only one Replicator Unit is adapting in both the closed-loop modeler and control adaptor. Though both the control adaptor and closed-loop modeler use the same training signal, each has its own error signal, $\bar{\epsilon}_C$ and $\bar{\epsilon}_M$, respectively.

3.6 Adaptive Update Law

The adaptive update law at time $n+1$ depends on the weight $W_k(n)$, forward path signal $\bar{x}_k(n)$, backward path signal $\bar{x}_k^*(n)$, and the global error $\bar{\epsilon}(n)$, all at time n. $W_k(0)$ must be initialized prior to running the system. For most purposes initializing all matrices to upper diagonal matrices with all nonzero entries equal to 0 or 1 will suffice. The law for the k-th Toeplitz synapse at time $n+1$, for $n = 0, 1, ..., N$ is as follows

$$W_k(n+1) = W_k(n) + \mu_k(n) U_0 * (\bar{x}_k^*(n) \bar{x}_k^T(n)) \tag{3.16}$$

where

$$\mu_k(n) = \beta_k F(n), \tag{3.17}$$

$$U_0 = \begin{bmatrix} 1 & 1 & \dots & 1 \\ 0 & 1 & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix} \in \mathbb{R}^{K\times K}, \tag{3.18}$$

and $W_k \in \mathbb{R}^{K\times K}$ with K being the number of neurons in the ganglia connected via the Toeplitz synapses. The symbol * denotes the Hadamard product, which is term by term matrix multiplication. Additionally,

$$F(n) = \frac{P(n)}{A(n)}, \tag{3.19}$$

$$P(n) = L(\frac{1}{2}\|\bar{\epsilon}(n)\|^2 - J), \tag{3.20}$$

$$L(\sigma) = \begin{cases} \sigma : \sigma > 0 \\ \\ 0 : \sigma \leq 0 \end{cases}, \tag{3.21}$$

and

$$A(n) = \sum_{\omega} \|\bar{x}_k^*(n)\|^2 \|\bar{x}_k(n)\|^2, \tag{3.22}$$

where $\|\cdot\|$ denotes the Frobenius norm.

Note that in Fig. 3.11, we have two ganglia feeding into a third ganglia, where the size of one ganglia depends on $\mathcal{L}$ and the size of the other ganglia depends on $\mathcal{P}$. Thus, $\bar{x}_k^*(n)\bar{x}_k^T(n)$ in (3.16) is not always properly defined. In this case, we define the following update law.

$$\text{Toep}\big[W_k(n+1)\big] = \text{Toep}\big[W_k(n)\big] + \mu_k(n)\text{avd}\big(|\bar{x}_k^*(n)\bar{x}_k^T(n)|\big), \tag{3.23}$$

where we define $\text{avd}\big(|\Gamma|\big) \in \mathbb{R}^N$ with

$$\text{avd}\big(|\Gamma|\big)_k := \frac{1}{M}\sum_{i=1}^{M} \Gamma_{i,k+i-1} \tag{3.24}$$

for $k = 1, \cdots, N$ and $\Gamma \in \mathbb{R}^{M\times(M+N-1)}$.

In the above equations, P is referred to as the Performance Function, $J \in \mathbb{R}$ is the desired mean square error level, $A(n)$ is the neural activity level, and ω is the set of all Toeplitz synapses connecting the two ganglia in question. The matrix (3.18) constrains the entire equation to Toeplitz form. The global error $\bar{\epsilon}(n)$ is the error of the system being replicated. For example, in Fig. 3.13, the closed-loop modeler replicates the unknown plant and uses the difference between the Replicator Unit and the closed-loop system output to drive the adaptation. So for this part of the system the global error is considered $\bar{\epsilon}_m$.

In (3.17), $\mu_k(n)$ is a time varying adaptive speed. This time varying adaptive speed, along with the Toeplitz synapses, is one of the defining features of the ANC architecture. β_k is the learning rate constant. This constant is chosen before adaptation begins and is programmed into the system.

3.7 Convergence Results for the ANC System

Convergence results for a Toeplitz network with adaption described by (3) through (9) can be found in (Hyland, 1997). First, we note that $\frac{1}{2}\|\bar{\epsilon}(n)\|^2$ is a function of the training history and of all of the weight matrices W_k for $k = 1, ..., N_a$, where N_a is the number of neural arrays, at time n. Therefore, we can write

$$\Lambda(W(n)) = \frac{1}{2}\|\bar{\epsilon}(n)\|^2. \tag{3.25}$$

We say that (3.25) is bounded by a homogeneous function of degree M if and only if

$$0 < \frac{\Lambda - \Lambda_0}{\frac{1}{M}(W - W_0)^T \frac{\partial \Lambda}{\partial W}} < 1, \tag{3.26}$$

for $\forall (W - W_0) \neq 0_{N_s}$ and $W_0 \in \mathbb{R}^{N_s}$, where N_s is the total number of independent, nonzero, synaptic weights, and Λ has a single global minimum Λ_0 at $W = W_0$. It is proven that if we assume that all vectors $\bar{x}_k^*(n)$ and $\bar{x}_k(n)$ are uniquely determined, $\Lambda_0 \leq J$, $\Lambda(n)$ is bounded by a homogeneous function of degree M, and $\beta_k < 2M$, that

$$\lim_{n \to \infty} \Lambda(n) \leq J. \tag{3.27}$$

3.8 Simulations

In this section, we perform control simulations, first using a PID control, and then using the ANC system. We use the linearized yaw axis model of our laser targeting system. This model is given in Chap. 2 and restated here for clarity:

$$dx_{AZ}(t) = (A_{AZ}x_{AZ}(t) + B_{AZ}u_{AZ}(t))dt + G_{AZ}dw_{AZ}(t). \tag{3.28}$$

$$y_{AZ}(t) = C_{AZ}x_{AZ}(t) + v_{AZ}(t), \tag{3.29}$$

Here

$$A_{AZ} = \begin{bmatrix} 0 & C_x & 0 & 0 \\ -\frac{K_{a\omega}}{C_x J_a} & -\frac{K_{af}}{J_a} & \frac{k_b N}{J_a} & 0 \\ 0 & 0 & -\frac{1}{\tau_{iAZ}} & \frac{K_{iAZ}}{\tau_{iAZ}} \\ 0 & 0 & 0 & -\frac{1}{\tau_{sAZ}} \end{bmatrix}, \quad x_{AZ}(t) = \begin{bmatrix} f_y(t) \\ \omega_a(t) \\ i_{aAZ}(t) \\ i^*_{aAZ}(t) \end{bmatrix} \tag{3.30}$$

$$B_{AZ} = \begin{bmatrix} 0 & 0 & 0 & \frac{K_{sAZ}C_{\omega AZ}}{\tau_{sAZ}} \end{bmatrix}^T, \quad u_{AZ}(t) = \begin{bmatrix} f^*_x(t) \end{bmatrix}, \tag{3.31}$$

$$G_{AZ} = \begin{bmatrix} 0 & J_a^{-1} & 0 & 0 \end{bmatrix}^T, \quad C_{AZ} = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}. \tag{3.32}$$

For all of the following simulations, we use the following control architecture and simulation values. With reference to Fig. 3.13, we use a linear IIR Replicator Unit for W_M and a nonlinear FIR Replicator Unit for W_C. The IIR Replicator Unit has two linear input ganglia, one with $M + R - 1$ neurons and the other with $M + L + R - 1$ neurons and one linear output ganglia with $R-1$ neurons. The nonlinear FIR Replicator Unit has a linear input layer with one ganglia, a single nonlinear hidden layer with three ganglia, and a linear output layer with one ganglia. Each ganglia contains ten neurons. For β in (3.17), we use different values depending on the location of the ganglia. We use β_C for neurons in W_C, β_M for neurons in W_M, $\alpha_{\mathcal{P}}$ or $\alpha_{\mathcal{L}}$ for neurons in the IIR Replicator Unit, and α for neurons in the FIR Replicator Unit. Then we write β as a product of the above. For example, for a linear neuron in W_M, which is an IIR Replicator Unit, we write $\beta = \beta_M \alpha_{\mathcal{P}}$. All simulation values are found in Table 3.1.

Table 3.1: Similation Values for ANC Control of GLTS

Parameter	Value
M	5
R	10
L	20
β_C	0.05
β_M	0.2
$\alpha_{\mathcal{P}}$	0.1
$\alpha_{\mathcal{L}}$	0.1
α	1
J	10^{-8}
Initial values of nonlinear neurons	10^{-4}
Initial values of linear neurons	10^{-2}
sample time	10^{-4}
W_m	10^{-6}
W_p	10^{-3}

3.8.1 Control: PID

We use the closed loop architecture seen in Fig. 3.14 to simulate a PID control for the yaw axis of the GLTS, where

$$C(s) = K_P + K_I\frac{1}{s} + K_D\frac{Ns}{s+N}. \tag{3.33}$$

The following values were used for the controller: $K_p = 1.04 \times 10^{-2}$, $K_I = 2.43 \times 10^{-4}$, $K_D = 4.73 \times 10^{-2}$, and $N = 6.45$ (Salaheen, 2013). We runs simulations for the GLTS without noise and with both measurement and process noise. Fig. 3.15 shows the result of the simulation without noise, whereas Fig. 3.16 shows the result of the simulation with noise. For the noise simulation we Gaussian white noise for both the measurement and process noise, with variances $W_m = 10^{-6}$ and $W_p = 10^{-3}$, respectively.

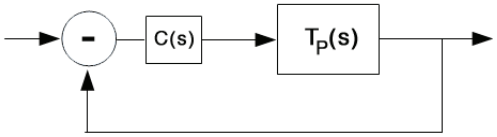

Figure 3.14: PI Control Loop

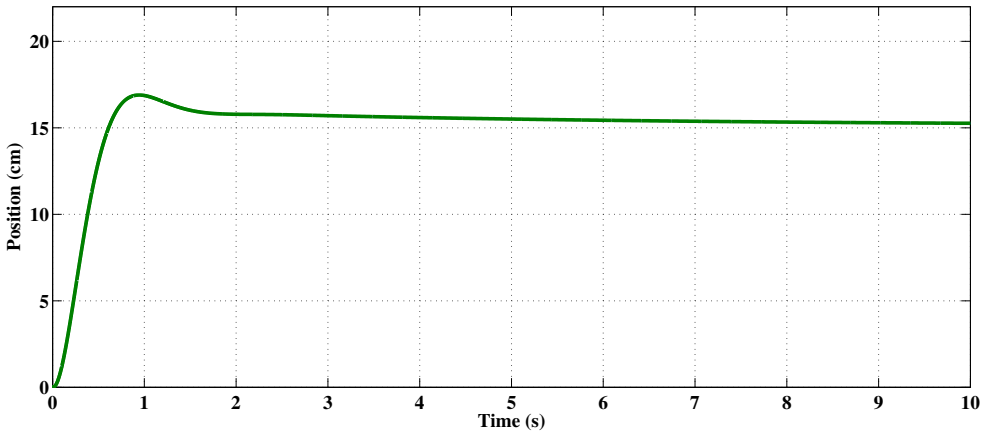

Figure 3.15: PID Control of GLTS

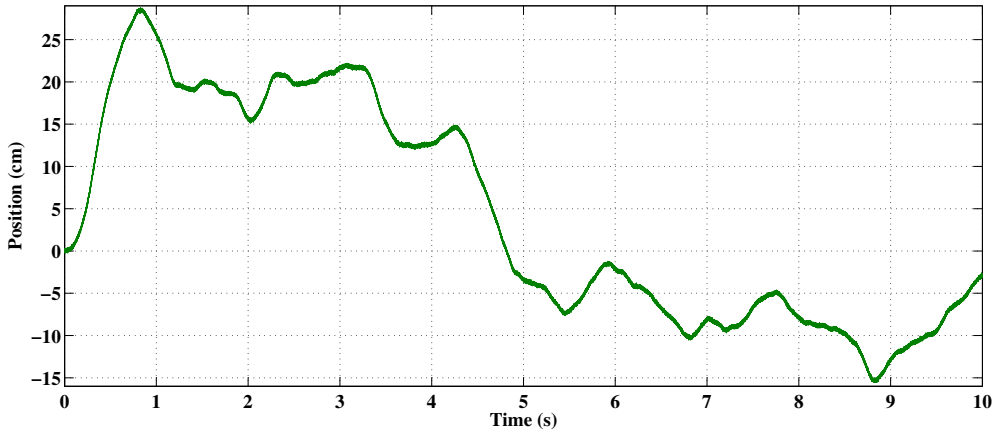

Figure 3.16: PID Control of GLTS with Process and Measurement Noise

3.8.2 Control: ANC

Next, we control the yaw axis of the GTLS using the linear model. The following input is applied to the system at $t = 0$

$$u(t) = 15.15, \tag{3.34}$$

which corresponds to the center position of the screen, in cm. The ideal reference system is chosen arbitrarily and described by

$$T_I(s) = \frac{9}{s+9}. \tag{3.35}$$

Figs. 3.17 and 3.18 shows the results of this simulation without any added noise.

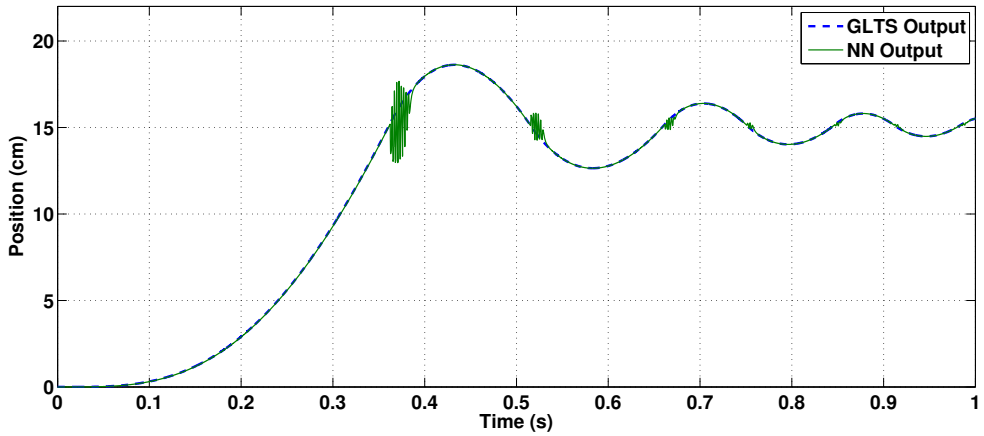

Figure 3.17: ANC System Replication of GLTS

As in the PID case, we simulate the ANC system with both measurement and process noise, with the same noise variances as the PID simulation, $W_m = 10^{-6}$ and $W_p = 10^{-3}$, respectively. Fig. 3.19 shows the result of this simulation.

3.8.3 Control Performance: PID and ANC

Table 3.2 gives shows the maximum absolute value, root mean square pointing error (RMSPE), and standard deviation of pointing error (SDPE) of the PID and ANC controllers, for the simulations with added measurement and process noise. The pointing error is the difference between the reference point and the actual position co-ordinates. The standard deviation of the

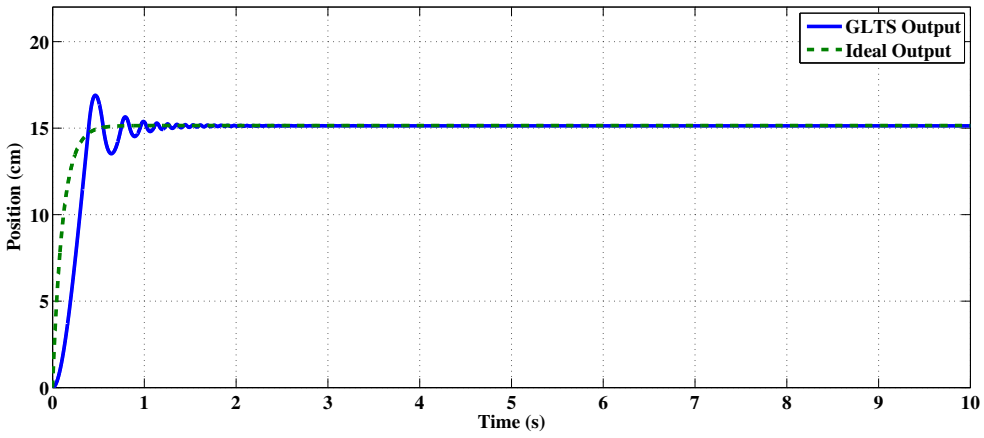

Figure 3.18: ANC Control of GLTS

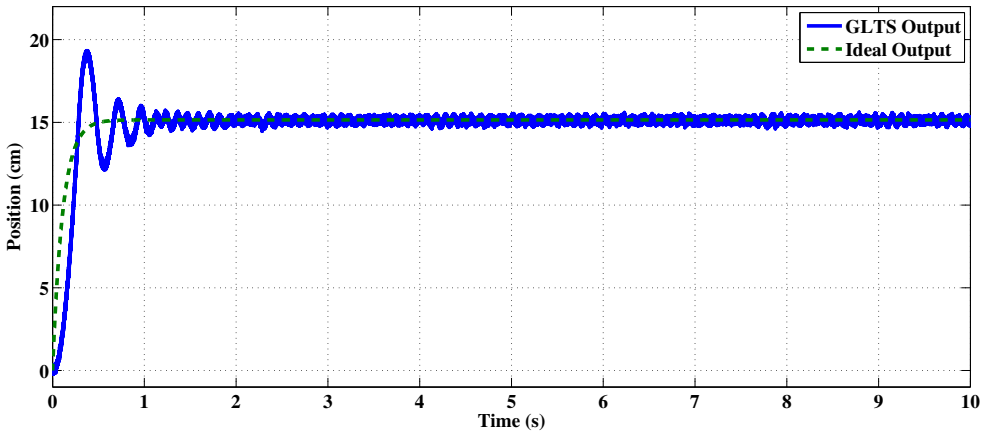

Figure 3.19: ANC Control of GLTS with Process and Measurement Noise

pointing (or reference) error indicates the variation of the error from the mean error value. We calculated the RMSPE and SDPE using the following equations, respectively,

$$\begin{aligned}\mu_\varepsilon &= \sqrt{\frac{\sum_{t=t_0}^{t_f} \varepsilon^2(t)}{N_\varepsilon}}, \\ &= \frac{\sqrt{\sum_{t=t_0}^{t_f} \varepsilon^2(t)}}{\sqrt{N_\varepsilon}}, \\ &= \frac{\|\varepsilon(t)\|_2}{\sqrt{N_\varepsilon}},\end{aligned} \tag{3.36}$$

$$\sigma_\varepsilon = \sqrt{\frac{\sum_{t=t_0}^{t_f} (\varepsilon(t) - \mu_\varepsilon)^2}{N_\varepsilon - 1}}. \tag{3.37}$$

Here μ_ε is the root mean squared error, ε is the error between the set-point position co-ordinate and the current position co-ordinate, N_ε is the number of samples. We do not consider the rise time when calculating the above metrics and therefore set $t_0 = 1$ second.

Table 3.2: PID and ANC Controller Statistics for Linear GLTS Model

Controller	Maximum Value (cm)	RMSE	SDPE
PID	36.32	17.56	32.36
ANC	19.3	0.15	0.24

3.8.4 Control: ANC for Nonlinear Gimbal System

In this section we present a nonlinear gimbal model and perform control using the ANC system. For the two-axis gimbal is given as (Kennedy and Kennedy, 2003)

$$\dot{x} = A(t)x(t) + F(x,t) + J\big[T_{\text{rate}} + T_{\text{dist}}\big] \tag{3.38}$$

where

$$\begin{aligned} x_1(t) = \int_0^t \omega_{Iy}(\tau)d\tau, \quad x_3(t) = \int_0^t \omega_{Iz}(\tau)d\tau, \\ \dot{x}_1(t) = x_2(t) = \omega_{Iy}(t), \quad \dot{x}_3(t) = x_4(t) = \omega_{Iz}(t), \\ F(x,t)^T = \Big[0, \;\; J_{\Delta z}\omega_{Iz}^2 \tan(\epsilon), \;\; 0, \;\; \frac{g_{yz}(t)\omega_{Iy}\omega_{Iz}}{J_S}\Big], \end{aligned}$$

$$g_{yz}(t) = \big(J_{Oz} + J_{Ix}\big)\tan(\epsilon) - \cos(\epsilon)\sin(\epsilon)\big(J_{Iz} - J_{Ix}\big),$$

$$J_{\Delta z} = \frac{J_{Ix} - J_{Iz}}{J_{Iy}},$$

$$J = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & J_{Iy}^{-1} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & J_S^{-1} \end{bmatrix},$$

$$J_S = J_{Oz} + \cos^2(\epsilon)J_{Iz} + \sin^2(\epsilon)J_{Ix},$$

$$A(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{-K_{I\omega}}{J_{Iy}} & \frac{-K_{If}}{J_{Iy}} & 0 & -J_{\Delta z}\omega_{Ox} \\ 0 & 0 & 0 & 1 \\ 0 & \frac{-g_y(t)\omega_{Ox}}{J_S} & \frac{-K_{O\omega}}{J_S} & \frac{-K_{Of}}{J_S} \end{bmatrix},$$

$$g_y(t) = \cos(\epsilon)\big(J_{Iy} - J_{Ix}\big) - \frac{1}{\cos(\epsilon)}\big(J_{Oz} + \sin^2(\epsilon)J_{Ix}\big),$$

$$T_{\text{rate}}^T = \Big[0,\ T_{I,\text{rate}},\ 0,\ 0\Big],$$

$$T_{\text{dist}}^T = \Big[0,\ T_{I,\text{dist}},\ 0,\ T_{O,\text{dist}}\Big].$$

The control objective here is to stabilize the angular velocity in the presence of added disturbances. Thus, we would like an ideal reference system whose output is zero. The issue with this is that the ANC System cannot use an all zero signal. Thus, we stabilize the angular velocity at a nonzero reference, which we will take to be 1 rad/sec. The simulation values are found in Table 3.4.

Figs. 3.20 and 3.21 show the results of the simulation with and without added noise, respectively. From these figures we can see that without any modeling information, the ANC System is able to control the nonlinear gimbal model in the presence of added noise. Additionally, we include the maximum value, root mean square error and standard deviation of reference error in Table 3.3.

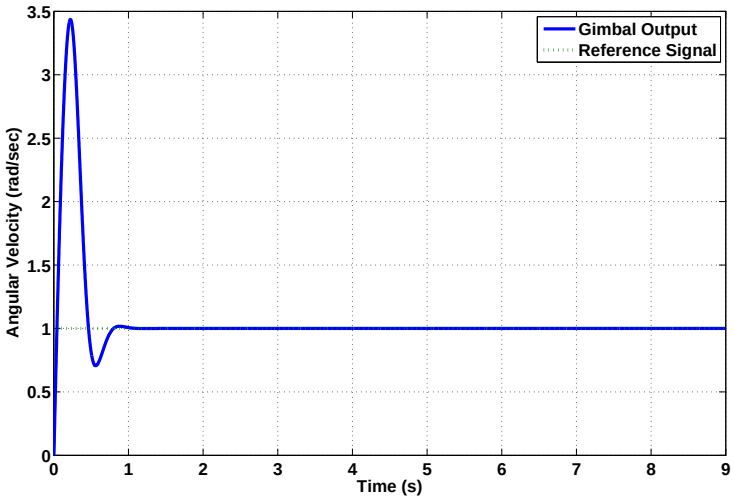

Figure 3.20: Control of Nonlinear Gimbal without Disturbances

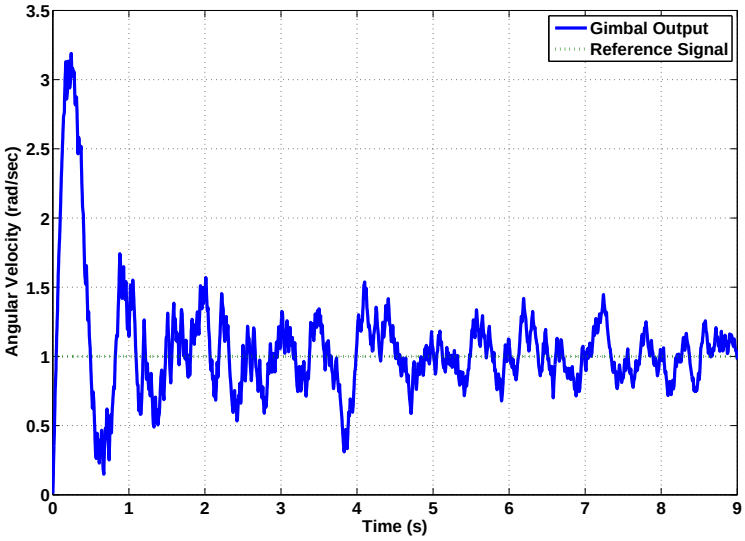

Figure 3.21: Control of Nonlinear Gimbal with Disturbances

Table 3.3: ANC Controller Statistics for Nonlinear Gimbal

Maximum Value (rad/sec)	RMSE	SDRE
3.19	0.19	0.28

3.9 Hardware Implementation

Each of the following experiments uses the same disturbance. We use a Newmark linear stage movers to create a disturbance. Fig. 3.22 shows the disturbance with no control.

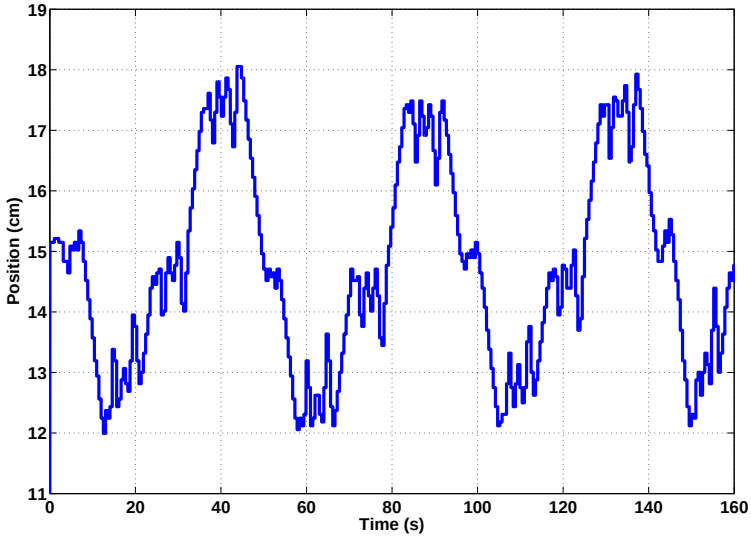

Figure 3.22: Yaw Axis Disturbance

3.9.1 Proportional

Figs. 3.23 and 3.24 show the result of a proportional controller without noise and with noise, respectively. Since the screen size ranges from $0, 30.3$ cm, our position error ranges from $-15.15, 15.15$. To map this into the proper duty cycle signal, we use the following equation

$$u_{DC}(t) = \frac{0.5}{15.15} e_p + 0.5 \tag{3.39}$$

where $u_{DC} \in [0, 1]$ is the duty cycle, e_p is the pixel error, which is calculated as $15.15 - p_c$ where p_c is the current pixel value.

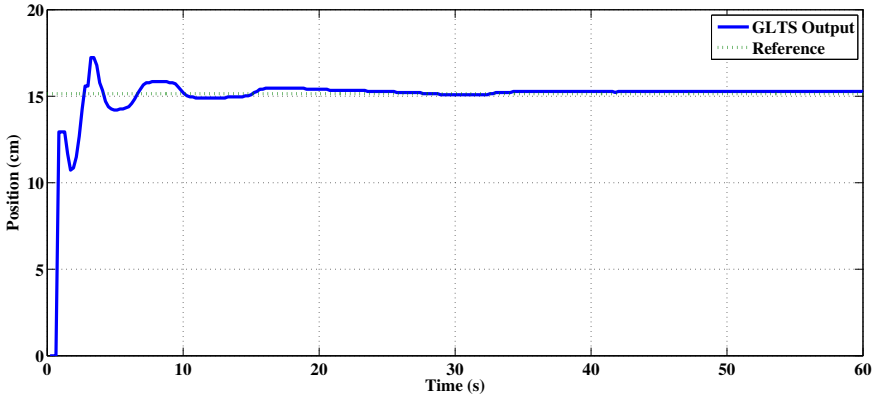

Figure 3.23: Proportional Control of GLTS Without Disturbance

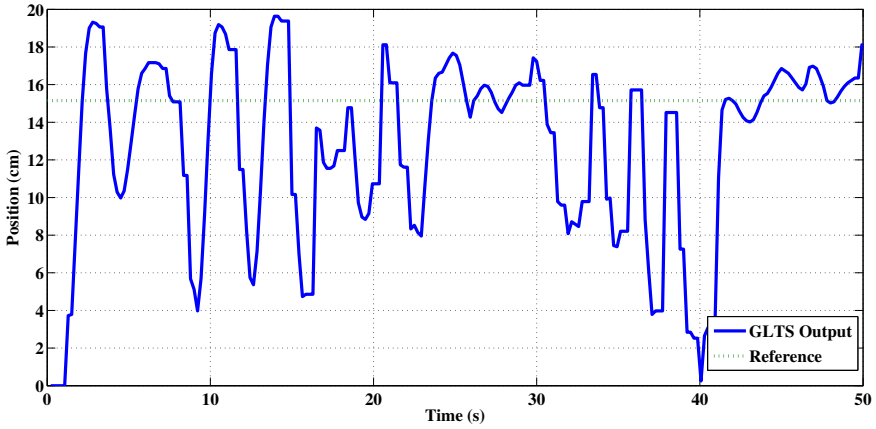

Figure 3.24: Proportional Control of GLTS With Disturbance

Table 3.4: Nonlinear Gimbal Parameters and Simulation Values

Parameter	Description	Simulation Value
ω_{Iy}	inner gimbal angular velocity, elevation axis	
ω_{Iz}	inner gimbal angular velocity, azimuth axis	
ω_{Ox}	outer gimbal angular velocity	1
ϵ	relative elevation angle between inner and outer gimbal frame	5
J_{Ix}	inner gimbal inertia coefficient	0.05
J_{Iy}	inner gimbal inertia coefficient	0.325
J_{Iz}	inner gimbal inertia coefficient	0.2
J_{Oz}	outer gimbal inertia coefficient	0.6
$K_{I\omega}$	inner gimbal cable restraint coefficient	0.1
$K_{O\omega}$	outer gimbal cable restraint coefficient	0.1
K_{If}	inner gimbal viscous friction coefficient	0.56
K_{Of}	outer gimbal viscous friction coefficient	0.56
$T_{I,\text{rate}}$	control signal	
$T_{I,\text{dist}}$	Gaussian white noise	10^{-3}
$T_{O,\text{dist}}$	Gaussian white noise	10^{-3}

3.9.2 ANC: System Replication

Figs. 3.25 and 3.26 show the results of the system replication experiment. We used the following parameters: 3 neurons per ganglia, α = 0.001, J = 10^{-5}, all initial weight values equal to 0, and a sample time of 10^{-2} seconds. We see that after 0.6 seconds the output of the neural network matches that of the GLTS.

Fig. 3.27 shows the control results. We used the following parameters: 10 neurons per ganglia, α = 0.01, J = 10^{-8}, all initial weight values equal to 1.0×10^{-6}, $\beta_M = 0.01$, $\beta_C = 2.36 \times 10^{-4}$, and a sample time of 0.01 seconds.

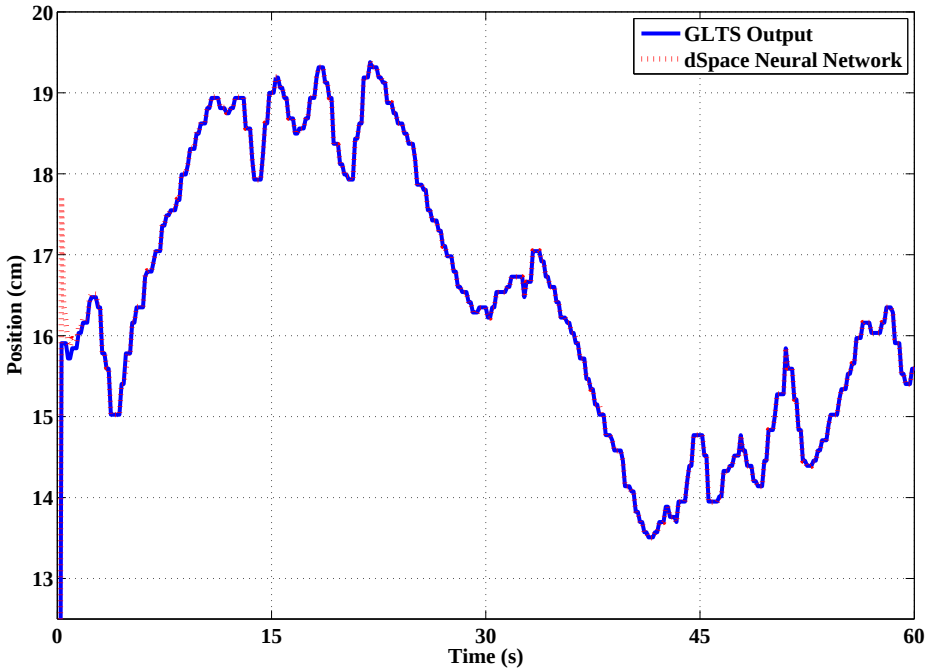

Figure 3.25: System Replication of Gimbal System

3.10 Summary

In this chapter we have given an outline of the architecture of the ANC system along with convergence results as formulated by Hyland. We simulated control of the linear GLTS model with the ANC system. Finally, we implemented the ANC system in hardware and compared its performance against a Proportional controller.

From these results of the simulations in Figs. 3.15-3.19, and the metrics given in Table 3.2, we can see that the added measurement and process noise have little effect on the ANC system's ability to control the GLTS, while the PID controller is not able to properly control the model. This is not surprising since the ANC system has been shown to be powerful in terms of noise rejection. The results of the PID controller are also not surprising since PID controllers do not consider stochastic systems.

From Figs. 3.24 and 3.27, we see that the Proportional controller out performs the ANC controller. We note that the ANC system implemented in hardware and the ANC system imple-

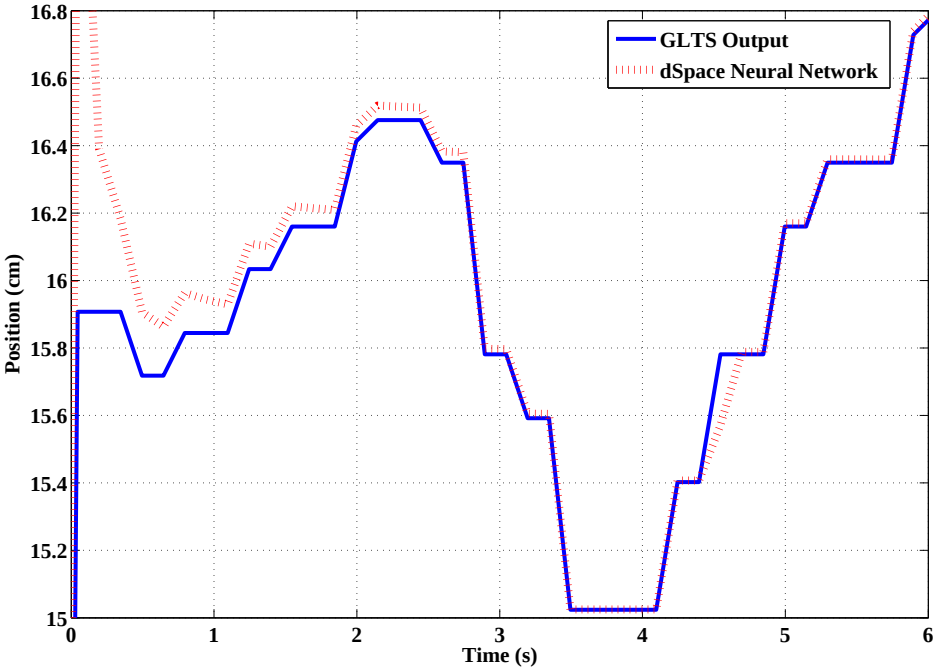

Figure 3.26: System Replication of Gimbal System

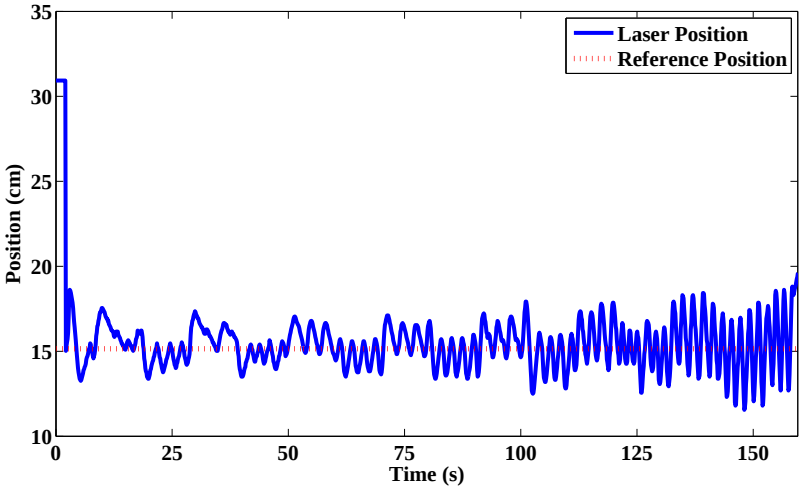

Figure 3.27: ANC Control of Gimbal System

mented in the simulations above differ in their sample time. The sample time for the hardware was 10^{-2} seconds while the sample time for the simulations was 10^{-4} seconds. Because of hardware limitations of the dSPACE board, we were not able to run the ANC system at the same sample time as the simulations. The sample time for the Proportional controller was set at 10^{-4} seconds. To properly compare these three controllers we would like to have both controllers running at the same speed.

We were, however, able to run a single neural network on the dSPACE board at a sample time of 10^{-4} seconds, for system replication as seen in Fig. 3.25. From this hardware experiment, we see that the Replicator Unit is able to properly replicate the GLTS output after a few seconds, which closely matches our simulation results. We would expect to have similar control results if our real time implementation was running at the same speed as our simulations. Because of the hardware limitations of the dSPACE board, we must look for other ways of implementing the ANC system in hardware. Thus, we propose to use an FPGA for implementation.

Having discussed the ANC system and shown its control capabilities, we now examine the resiliency of the ANC system, we exposed to malicious attacks.

CHAPTER 4

RESILIENT CONTROL USING ANC

Having examined the ANC system's control capabilities in the previous chapter, we now examine the resiliency of the ANC system. First, we define resilient control and discuss its methodologies. Next, we define the resilient metrics by which the ANC system's resiliency will be measured. Finally, we simulate four attacks on the linear GLTS model with both a PID controller and the ANC system. We compare the resiliency of both controllers.

Resiliency is defined as the system's ability to maintain operational normalcy when subjected to unexpected anomalies in the form of attacks. Thus, we subject the linear model of the GLTS to the following attacks: added latency, false data injection, sensor data alteration, and plant parameter changes. The added latency and plant parameter change attacks can be seen as a type of attack where the attacker is manipulating the physical structure of the plant and/or controller. The false data injection and sensor data alteration attacks can be seen as spoofing type attacks.

4.1 Resilient Control Theory

Resilient control is an emerging, multidisciplinary research area which combines cognitive psychology, computer science, and control engineering, in order to address the question of how to properly control highly complex, interconnected systems in the face of anomalies (Rieger, 2010). Resiliency is defined as the capacity of a control system to maintain state awareness and to proactively maintain a safe level of operational normalcy in response to anomalies. These anomalies include threats that counter normalcy and destabilize control through malicious attacks and complex latencies (Rieger, 2010). Additionally, a resilient control system should protect stability, efficiency, and security (Rieger and Villez, 2012). A resilient industrial control

system is defined as one that is designed and operate in a way that (Wei and Ji, 2010):

- the incidence of undesirable incidents can be minimized;
- most of the undesirable incidents can be mitigated;
- the adverse impacts of undesirable incidents can be minimized, if these incidents cannot be mitigated completely;
- it can recover to normal operation in a short time.

In order to distinguish resiliency from terms like robustness, adaptiveness, and fault-tolerance, we must first define all of the above. Robustness is defined as the ability to maintain satisfactory stability or performance characteristics in the presence of all conceivable system parameter variations (Stengel and Ryan, 1991). Fault-tolerance is the ability of a controlled system to maintain control objectives, despite the occurrence of a fault, such as defects in sensors, actuators, or in the process itself (Blanke et al., 2000). Adaptiveness is the ability of the controller to automatically adjust in real time, in order to achieve or maintain a desired level of control performance (Landau, 2011). We note that none of the above definitions consider how quickly the control system recovers to operational normalcy, yet all of the properties above are characteristics of resilience. Therefore, resilience is a superset of all of the above properties (Wei and Ji, 2010).

The current research on resilient control can be divided into three areas: unexpected condition adaptation, human interaction challenges, and goal conflicts.

The first area deals with highly complex and interconnected systems. Most control methods do not address multiple, often unknown, latencies that exist in these interconnected systems. Research has been done which mitigates these latencies, but often these studies do not address the presence of unknown latencies or latencies that result from unexpected failures or attacks on the plant (Rieger, 2010).

Cyber awareness falls into the human interaction challenges. Researchers in both computer science and cognitive psychology address such issues as mechanisms to detect attacks,

understand attacks, and develop threat models (Boring, 2009). To accomplish this task, some researchers suggest a combination of passive and active techniques to deflect the attack or attacker, such as decoys and randomization (passive) and traffic rerouting (active) (Sridhar et al., 2012).

Finally, goal conflicts can be addressed by considering multiple performance measures and state awareness. When dealing with an interconnected system, one must consider not only stability, but physical and cyber security, process efficiency, and process compliancy (Rieger, 2010). It is easy to see that these performance measures are not easily combined into a single frame of reference and are often not easily normalized for comparisons. One author considers a combination of operational safety and high performance for a nuclear power plant (Jin et al., 2010).

4.2 Resilient Control Metrics

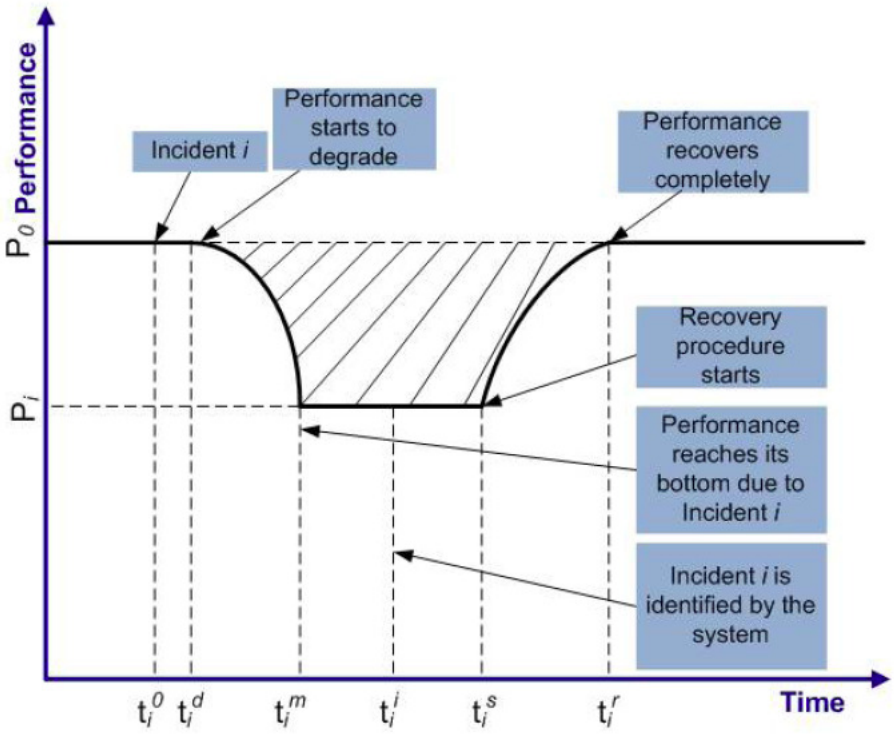

Figure 4.1: Resilience Curve

Since the concept of resiliency is still being developed, multiple authors have proposed different metrics for determining the system's resiliency. One initial study outlined the philosophy of a resilient system (Rieger, 2010) whereas other studies either propose resilient metrics (Wei and Ji, 2010) or show resiliency through stability metrics (Biswas et al., 2012). To better understand the resiliency of the ANC system we use the resilience curve shown in Fig. 4.1 and the following metrics (Wei and Ji, 2010):

- **recovery time** T_i^r - the time that the system needs to recover to normal operation from the incident i

$$T_i^r = t_i^r - t_i^s \tag{4.1}$$

- **performance degradation** P_i^d - maximal performance degradation due to the incident i

$$P_i^d = P_0 - P_i \tag{4.2}$$

- **protection time** T_i^p - the time that the system can withstand the incident i without performance degradation

$$T_i^p = t_i^d - t_i^0 \tag{4.3}$$

- **degrading time** T_i^d - the time that the system reaches its performance bottom

$$T_i^d = t_i^m - t_i^0 \tag{4.4}$$

To study the resiliency of the ANC system we will use all of the metrics defined above. This will be done in both simulation and in hardware. First, we will simulate the following anomalies: plant parameter changes, inter-system latencies, false data injection, and sensor data alteration. These simulations will use a linearized model of the laser targeting system and the above metrics will be measured. Second, we will introduce these same anomalies in hardware on the test bench, where, again, we will measure the above metrics.

Table 4.1: Performance Metrics

Notation	Description
t_i^0	the moment incident i occurs (s)
t_i^d	the moment the system performance starts to degrade (s)
t_i^m	the moment the system performance reaches the bottom (s)
t_i^s	the moment the system starts to recover (s)
t_i^r	the moment the system completely recovers (s)
P_0	the original system performance when the incident i occurs (cm)
P_i	the minimum performance due to incident i (cm)

4.3 Resiliency Simulations

In this section we examine the resiliency of the ANC system when controlling the gimbaled laser targeting system. We simulate the following types of attacks on the plant: added latencies, false data injection, sensor data alteration, and plant parameter changes. In each simulation, except for the added latency attack, we assume that the plant and control system having been running for some time and that the plant is being properly controlled. Since in both the PID and ANC simulations we have control by 4 seconds, we have the attacks occur at $t = 4$. For the added latency attack, we start the attack at $t = 0$. Since our control objective is to have the GLTS point at a single reference position, once control is achieved and the plant reaches steady state, the output from the plant will be constant. So adding at latency at this point will not effect the controller. Additionally, since the PID controller is not able to control in the presence of noise, we do not add any noise to these simulations, in order to properly compare the two controllers.

Added Latency

For this type of attack, we add a 25 sample delay in the feedback loop. As stated above, this attack occurs at $t = 0$. Figs. 4.2 and 4.3 show the result of the PID simulation and ANC

simulation, respectively. Table 4.2 shows the resilient metrics for this attack.

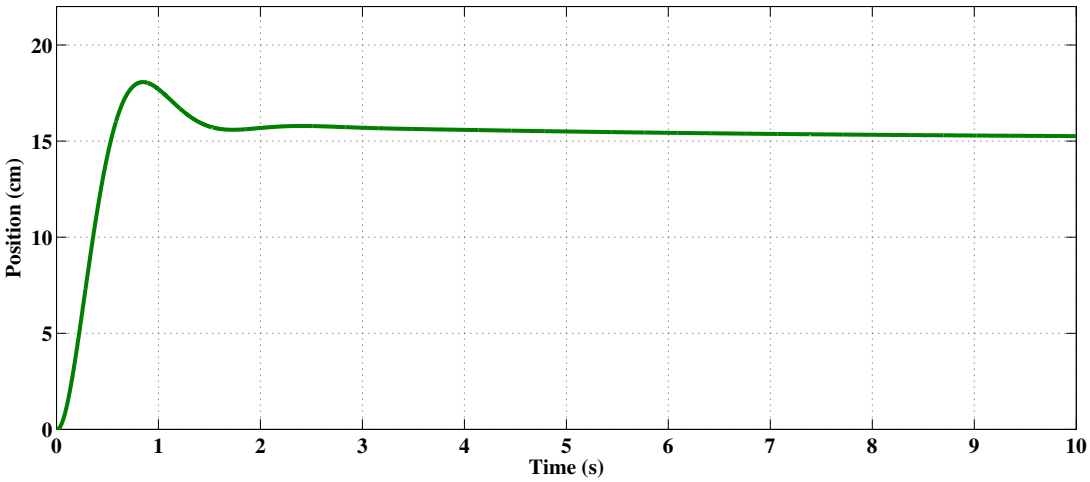

Figure 4.2: PID: Added Latency Attack

Table 4.2: Resilient Metrics for Added Latency Attack

Controller	t_{AL}^{0}	t_{AL}^{d}	t_{AL}^{m}	t_{AL}^{s}	t_{AL}^{r}	P_0	P_{AL}
PID	0 s	0 s	0.86 s	0.86 s	17.7 s	15.15 cm	18.07 cm
ANC	0 s	0 s	0.65 s	0.65 s	1.3 s	15.15 cm	16.07 cm

False Data Injection

The false data injection attack is a type of spoofing attack, where the attacker injects erroneous data into the plant. This attack occurs at $t = 4$ and we add a constant input of 10 cm to the controller output. Figs. 4.4 and 4.5 show the result of the PID simulation and ANC simulation, respectively. Table 4.3 shows the resilient metrics for this attack.

Sensor Data Alteration

This type of attack is also a spoofing attack, where the attacker alters the output of the plant or alternatively, attacks the plant sensor outputs. This attack occurs at $t = 4$ and we add a value of 25 cm to the plants output. Figs. 4.6 and 4.7 show the result of the PID simulation and ANC

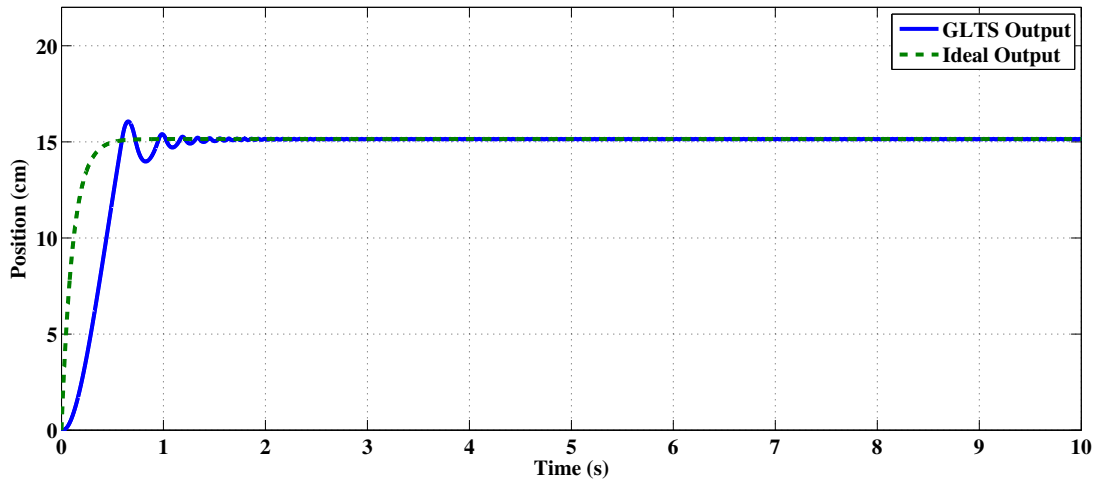

Figure 4.3: ANC: Added Latency Attack

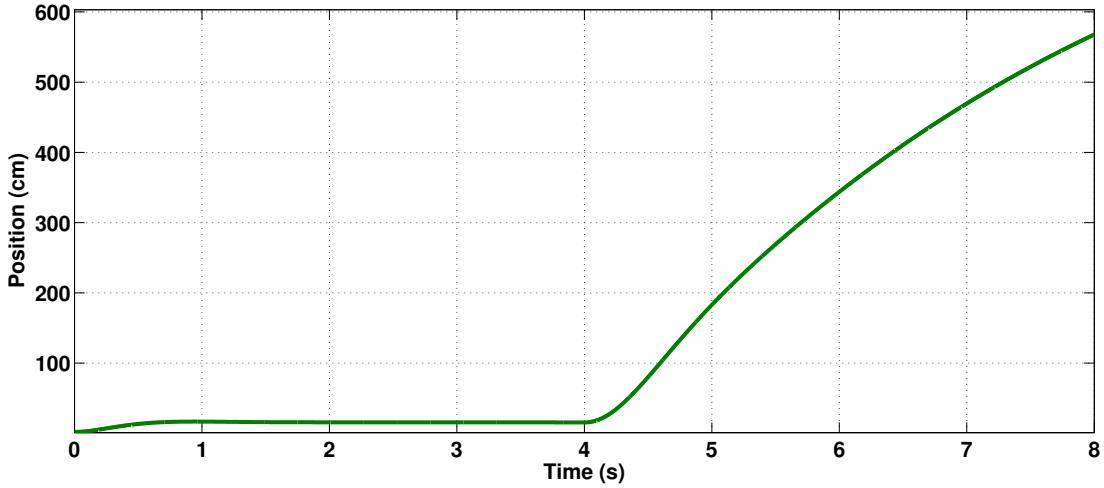

Figure 4.4: PID: False Data Injection Attack

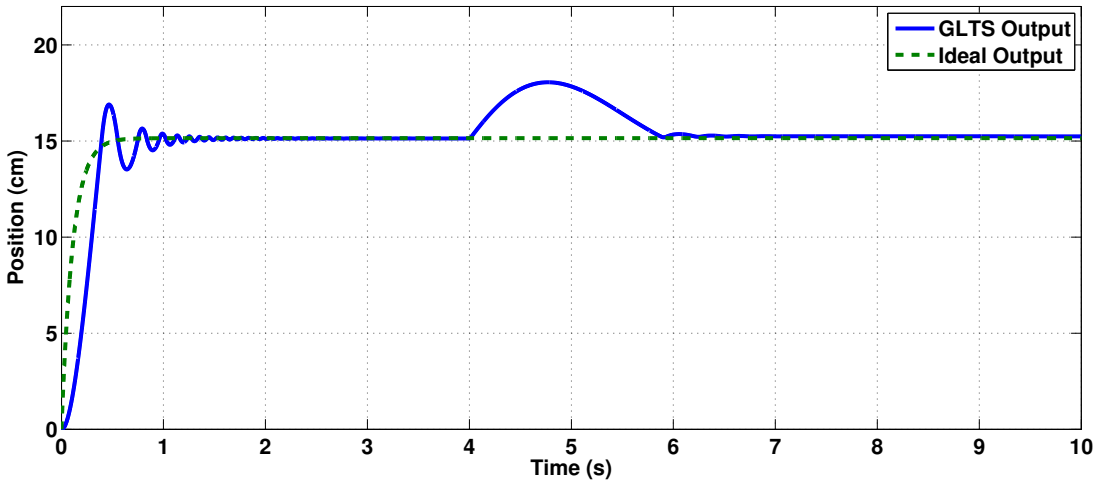

Figure 4.5: ANC: False Data Injection Attack

Table 4.3: Resilient Metrics for False Data Injection Attack

Controller	t^0_{FDI}	t^d_{FDI}	t^m_{FDI}	t^s_{FDI}	t^r_{FDI}	P_0	P_{FDI}
PID	4 s	4 s	-	-	-	15.15 cm	-
ANC	4 s	4 s	4.7 s	4.7 s	6.2 s	15.15 cm	18 cm

simulation, respectively. Table 4.4 shows the resilient metrics for this attack.

Table 4.4: Resilient Metrics for Sensor Data Alteration Attack

Controller	t^0_{SDA}	t^d_{SDA}	t^m_{SDA}	t^s_{SDA}	t^r_{SDA}	P_0	P_{SDA}
PID	4 s	4 s	4 s	4 s	24 s	15.15 cm	25.6 cm
ANC	4 s	4 s	4 s	4 s	5 s	15.15 cm	25.1 cm

Plant Parameter Changes

Here we consider an attack where the attacker is attempting to change the plant. To simulate this, we change the plant model A_{AZ}, B_{AZ}, and C_{AZ} to $\frac{A_{AZ}}{10}$, $10 \times B_{AZ}$ and $\frac{C_{AZ}}{10}$ at $t = 4$ seconds. Figs. 4.8 and 4.9 show the result of the PID simulation and ANC simulation, respectively. Table 4.5 shows the resilient metrics for this attack.

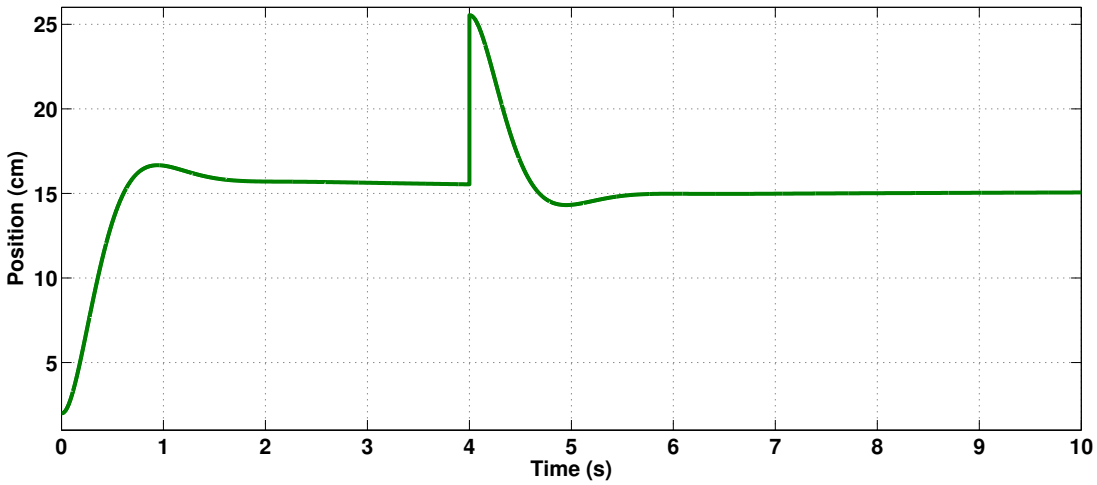

Figure 4.6: PID: Sensor Data Alteration Attack

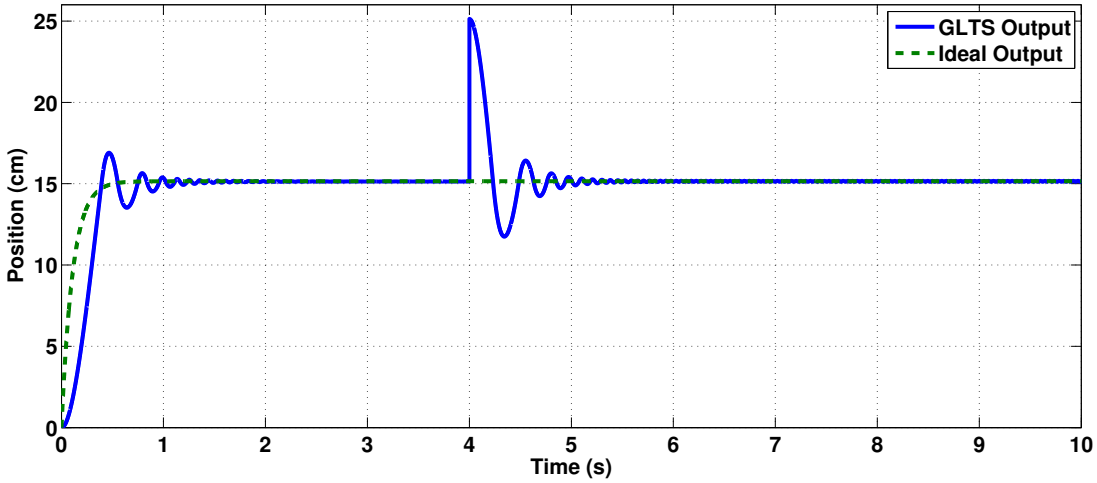

Figure 4.7: ANC: Sensor Data Alteration Attack

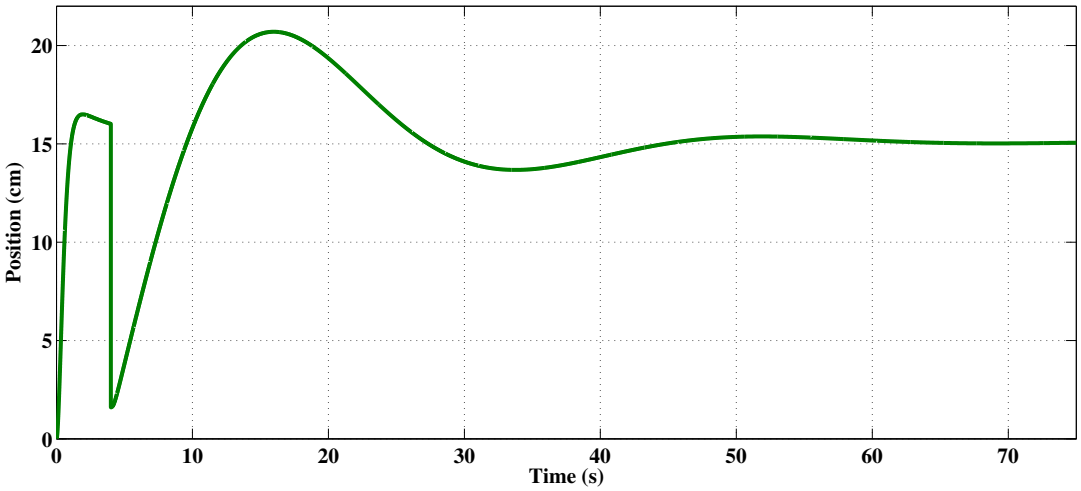

Figure 4.8: PID: Plant Parameter Change Attack

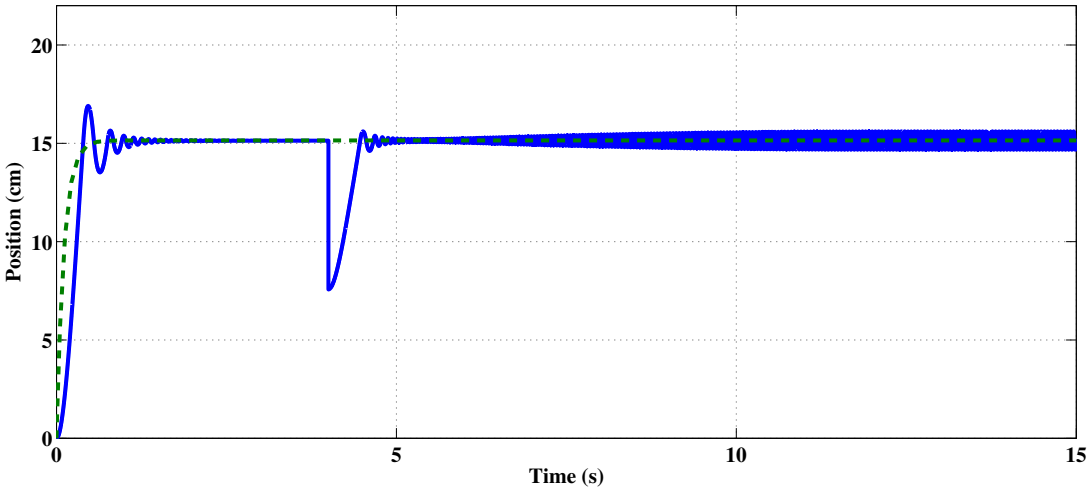

Figure 4.9: ANC: Plant Parameter Change Attack

Table 4.5: Resilient Metrics for Plant Parameter Changes Attack

Controller	t^0_{PPC}	t^d_{PPC}	t^m_{PPC}	t^s_{PPC}	t^r_{PPC}	P_0	P_{PPC}
PID	4 s	4 s	4 s	4 s	74 s	15.15 cm	1.6 cm
ANC	4 s	4 s	4 s	4 s	∞	15.15 cm	7.57 cm

The resiliency results of each attack simulation are given in Table 4.6.

Table 4.6: Resilient Metrics for PID and ANC Controllers

Metric	Added Latency		False Data Injection		Sensor Data Alteration		Plant Parameter Changes	
	PID	ANC	PID	ANC	PID	ANC	PID	ANC
T^r_i	16.84 s	0.65 s	∞	1.5 s	20 s	1 s	70 s	∞
P^d_i	-2.92 cm	-0.92 cm	∞	-2.85 cm	-10.45 cm	-9.95 cm	13.55 cm	7.58 cm
T^p_i	0 s	0 s	0 s	0 s	0 s	0 s	0 s	0 s
T^d_i	0.86 s	0.65 s	∞	0.7 s	0 s	0 s	0 s	0 s

4.4 Summary

In this chapter we outlined the theory and methodologies of resilient control. We also defined the metrics by which we determined the ANC system's resiliency. Finally, we simulated attacks on the GLTS, two of which were spoofing attacks and two of which were attacks against the physical plant and controller.

For the added latency attack, we added a latency of 25 samples within the feedback loop at $t = 0$ s. Both the PID and ANC controllers are able to properly maintain control. Examining the resilient metrics, we see that the PID controller has a much larger recovery time $T^r_{AL} = 16.84$ s while $T^r_{AL} = 0.65$ s for the ANC system. The performance degradation $P^d_{AL} = -2.92$ cm for the PID while $P^d_{AL} = -0.92$ cm for the ANC system. The protection time $T^p_{AL} = 0$ s for both systems. Finally, the degrading time for the PID controller is slightly higher than the ANC

system, with $T^d_{AL} = 0.86$ s for the PID and $T^d_{AL} = 0.65$ s for the ANC system. Thus, we see that for these metrics, the ANC system is more resilient to an added latency attack of this type.

For the false data injection attack, we inject a constant input of 10 cm to the controller output at $t = 4$ s. We see that the PID completely fails to maintain control for this attack, with the output of the GLTS diverging. The ANC system has a recovery time $T^r_{FDI} = 0.65$ s, performance degradation $P^d_{FDI} = -2.92$ cm, and degrading time $T^d_{FDI} = 0.86$ s. Again, we see that the ANC system outperforms the PID controller when subjected to this type of attack.

For the sensor data alteration attack, we inject a constant input of 25 cm to the plants output at $t = 4$ s. We see both systems are able to recover from this attack. Both the PID and ANC have equal protection time $T^p_{SDA} = 0$ and degrading time $T^d_{SDA} = 0$. Additionally, we have a similar performance degradation with $P^d_{SDA} = -10.45$ cm for the PID and $P^d_{SDA} = -9.95$ for the ANC. Finally, we see the ANC system outperforms the PID controller when considering recovery time T^r_{SDA}. Here $T^r_{SDA} = 1$ s for the ANC system and $T^r_{SDA} = 20$ s for the PID system. Thus, we conclude the ANC system is more resilient than the PID controller when subjected to this type of attack.

The final attack is a plant parameter change attack, in which we change the physical system at $t = 4$ s. We note here that, while the ANC system is able to regain some control, in a much shorter time than the PID controller, the system will oscillate after the attack. Thus, we conclude that the ANC system is not able to properly recover from the attack. The PID controller has a recovery time $T^r_{PPC} = 70$ s. The ANC system does outperform the PID when considering performance degradation, with $P^d_{PPC} = 7.58$ cm for the ANC system and $P^d_{PPC} = 13.55$ cm for the PID controller. Hence, we see that both systems are resilient to this tpe of attack only when considering limited metrics.

Now that we have examined the theory of the ANC system, its capabilities in terms of control and noise rejection, and the ANC system's resiliency, we now turn to an alternate hardware implementation. This alternate implementation will be on an FPGA.

CHAPTER 5

ANC USING FPGA

In this chapter we outline our implementation of the ANC system on an FPGA. First, we discuss the FPGA and it's need for implementation of the ANC system. Next, we discuss our computational preliminaries, such as floating point to fixed point conversion, division, and the nonlinear neural function. Next, we present simulations and "hardware in the loop" simulations of the division design, nonlinear neural functions, and various Replicator Unit designs. Finally, we simulate control of the linear GLTS model with our hardware model of the ANC system.

5.1 Introduction and Need for FPGA

With the hardware limitations of the dSPACE board, we must look elsewhere in order to properly implement the ANC system for real time control. One option is to run the controller on an FPGA. This hardware implementation will allow us to take advantage of the parallelisms within the neural network by using the parallel processing capabilities of the FPGA. We will use the Virtex-5 FPGA for our hardware.

The Xilinx Virtex-5 is a family of devices that consists of up to 240-by-108 configurable logic blocks (CLBs), 3420 Kb of distributed-RAM, 1032 Kb of block-RAM (arranged in 18-Kb blocks), 1056 DSP slices, and so forth. Note, not all stated maxima occur in any one device. We will be using the XC5VSX50T device. The clock rate which we will be running at is 100 MHz.

We propose to take advantage of Matlab / Simulink with Xilinx System Generator software for a graphical implementation of the controller. System Generator extends the Simulink block set to include functions specific to Xilinx hardware implementations. Additionally, Simulink and System Generator blocks can work in tandem with special interface blocks.

System Generator blocks can be used for both software simulations and in a hardware assisted mode, known as "hardware in the loop" or hardware cosimulation. During software simulation, the System Generator blocks perform just as Simulink blocks, and the simulation results are equivalent to those which would be produced by a hardware implementation. When in the hardware assisted mode, all System Generator blocks are converted into hardware implementation blocks. This hardware implementation is then downloaded to the FPGA and executed. The software simulation is essentially replaced by the hardware, running in real time. During the hardware assisted mode, one has the capability of sending and receiving data to and from the FPGA during its hardware execution. This gives us the capability of monitoring internal signals, which would otherwise prove difficult to access.

Using this software will allow us to do two things. First, we can implement the design in a fashion very similar to our dSPACE implementation. This allows us to compare the two designs and confirm their proper implementation. Additionally, we can easily simulate our system model before any hardware implementation occurs. Secondly, by exploiting the Simulink and System Generator relationships, we can easily interface our FGPA design with the current test bed.

5.2 FPGA Computation Preliminaries

Converting our design from Matlab / Simulink to something processable on an FPGA is not a trivial task. First, we must consider how to represent our data on the FPGA. Matlab uses a double-precision floating point data type according to the IEEE Standard 754 for double precision. This data type is not easily representable on an FPGA. Hence, we must convert all data to a fixed point representation.

Once the floating point to fixed point conversion is finished, we must modify all of our mathematical operations to be compatible with our fixed point representation. In some cases (addition, subtraction, multiplication, and special cases of division) this can be easily done through the use of the Xilinx / System Generator blockset. Within this blockset are blocks optimized for these specific tasks. Additionally, we note that Matlab is optimized for vector

operations, which was useful when implementing our first design. FPGAs, on the other hand, are not well suited for vector operations. Thus, the simple case of multiplying a matrix and a vector, which is handled in a single line of code in Matlab, must be handled by considering term by term multiplications and summations on an FPGA. In our case, we give a detailed explanation of our division implementation and how we handled the nonlinear neural tansig function.

5.2.1 Fixed Point Representations

Since the goal of this thesis is to implement a neural network on an FPGA, we must discuss the how the hardware represents data and handles arithmetic. We follow the exposition given in (Ruddy, 2012).

For an unsigned fixed point number we use the notation $U(a, b)$ where U represents the fact that the number is unsigned, a is a fixed number of integer bits, and b is a fixed number of fractional bits. The value of the unsigned word x is $a + b = N$ is given by

$$x = 2^{-b} \sum_{n=0}^{N-1} 2^n x_n, \tag{5.1}$$

where $a + b = N$ is the total number of bits. Thus, the weight of each bit x_n is 2^{n-b}. The resolution is set by the number of fractional bits, and is, therefore, equal to 2^{-b} and the range of representable numbers is 0 to $2^N - 1$. The decimal point is located between the bits weighted 2^1 and 2^0, giving us the following representation

$$x_a x_{a-1} ... x_1 x_0 . x_{-1} ... x_{-b}. \tag{5.2}$$

To represent an N-bit signed fixed point number, we use the two's complement of the above number (5.1) and use the notation $A(a, b)$, where A means signed. Since we are using two's complement, where the leading bit represents the sign, the number of integer bits is now $a = N - b - 1$. For a signed fixed point number the resolution is still equal to 2^{-b} but the range of representable numbers is now given by

$$-2^a \leq x \leq 2^a - 2^{-b}. \tag{5.3}$$

The value of an N-bit signed fixed point number is given by

$$x = 2^{-b}[-2^{N-1}x_{N-1} + \sum_{n=0}^{N-2} 2^n x_n]. \tag{5.4}$$

Throughout this paper we will use signed fixed point exclusively.

5.2.2 Fixed Point Arithmetic

To add two signed fixed point numbers $A(a, b)$ and $A(c, d)$ we must have either $a = c$ and $b = d$, or one number must be shifted so that the radix points are equal. If $a > c$ (or similarly $c > a$) one can easily attain $a = c$ by appending a string of $a - c$ zeros to $A(c, d)$, while maintaining the correct sign. If the number of fractional bits are not equal, one number must be right shifted by $|b - d|$ bits. To properly store the result of the operation we must consider a single bit of overflow, giving us a a final representation $A(a + 1, b)$.

Full precision signed multiplication is not always possible given hardware considerations. Often, one must round or truncate the resulting number, which can significantly affect accuracy. Ignoring this fact, full precision multiplication is given by

$$A(a, b) \times A(c, d) = A(a + c + 1, b + d). \tag{5.5}$$

As with multiplication, full precision division is not always possible in hardware and often results in loss of accuracy due to truncation or rounding. Full precision division is given by

$$\frac{A(a, b)}{A(c, d)} = A(a + b + 1, c + d). \tag{5.6}$$

Special consideration can be given to numbers which are integer of the form 2^k. Here multiplication reduces to a right shift by k bits and division reduces to left shifting by k bits.

5.2.3 Floating Point to Fixed Point Conversion

To convert from floating point to fixed point, we use an object in Matlab called the *NumericTypeScope*. This object provides information about the range of the data, in the form of a *log2* histogram. The histogram converts all floating point input data to binary, and then plots the bit weights in bins along the X axis, and the percentage of occurrence along the Y axis. The

histogram allows us to specify a word lengths, with integer and fractional part, and view how much of the data is within range, outside range, and below precision.

To use the NumericTypeScope, we first need to collect data from our simulation. Since the output of the plant matches that of the ideal system after approximately 2 seconds, we run each simulation for 2 seconds. During this time, we record the values of the weight matrices, the output from the linear and nonlinear neurons, and the error signals. In total, we have six weight matrices in the FIR Replicator Unit, which we will denote W_C^i for $i = 1, \cdots, 6$, two weight matrices in the IIR Replicator Unit, which are denoted W_M^i for $i = 1, 2$, five linear ganglia, denoted G_L^i for $i = 1, \cdots, 5$, three nonlinear ganglia, denoted G_N^i for $i = 1, 2, 3$, and two error signals e_C an e_M. The *log2* histograms for the W_M^1 and W_C^1 data sets are given below in Figs. 5.1 and 5.2. Columns in blue represent data that is representable by our fixed point choice, whereas red columns represent data outside the range, and yellow represents data below the precision. Note, we are presenting the results for a 32 bit word with 18 fractional bits. The reasoning for this choice is given below. We also give additional parameters for each data set in Table 5.1. These parameters include total number of data points, number of positive / negative data points, and minimum, maximum, and average values.

From the values seen in Table 5.1, the largest positive value is 1.9×10^4, the largest negative value is -4.04, and smallest magnitude is 5.2×10^{-9}. To eliminate all fixed point error our system should be able to represent that entire range. Using the *NumericTypeScopes* we can graphically change the word length and view the percentage of data outside of the range or below precision. Since the data in W_C^1 and W_M^1 contain the largest and smallest data, respectively, we can use these two plots to try to determine our fixed point representation.

From the plots, we can see that a 64 bit word with 32 fractional bits will allow us to properly represent all of our data. Given the size of the ANC system, this data type might not be optimal for hardware implementation with limited resources. Therefore, we choose a 32 bit word. We show the number of data points below precision and outside range in Table 5.2 for various fractional bits. From this table, we can see that as the number of fraction bits grows, the percentage of data out of range for W_M^1 shrinks, while percentage of data out of range for W_C^1

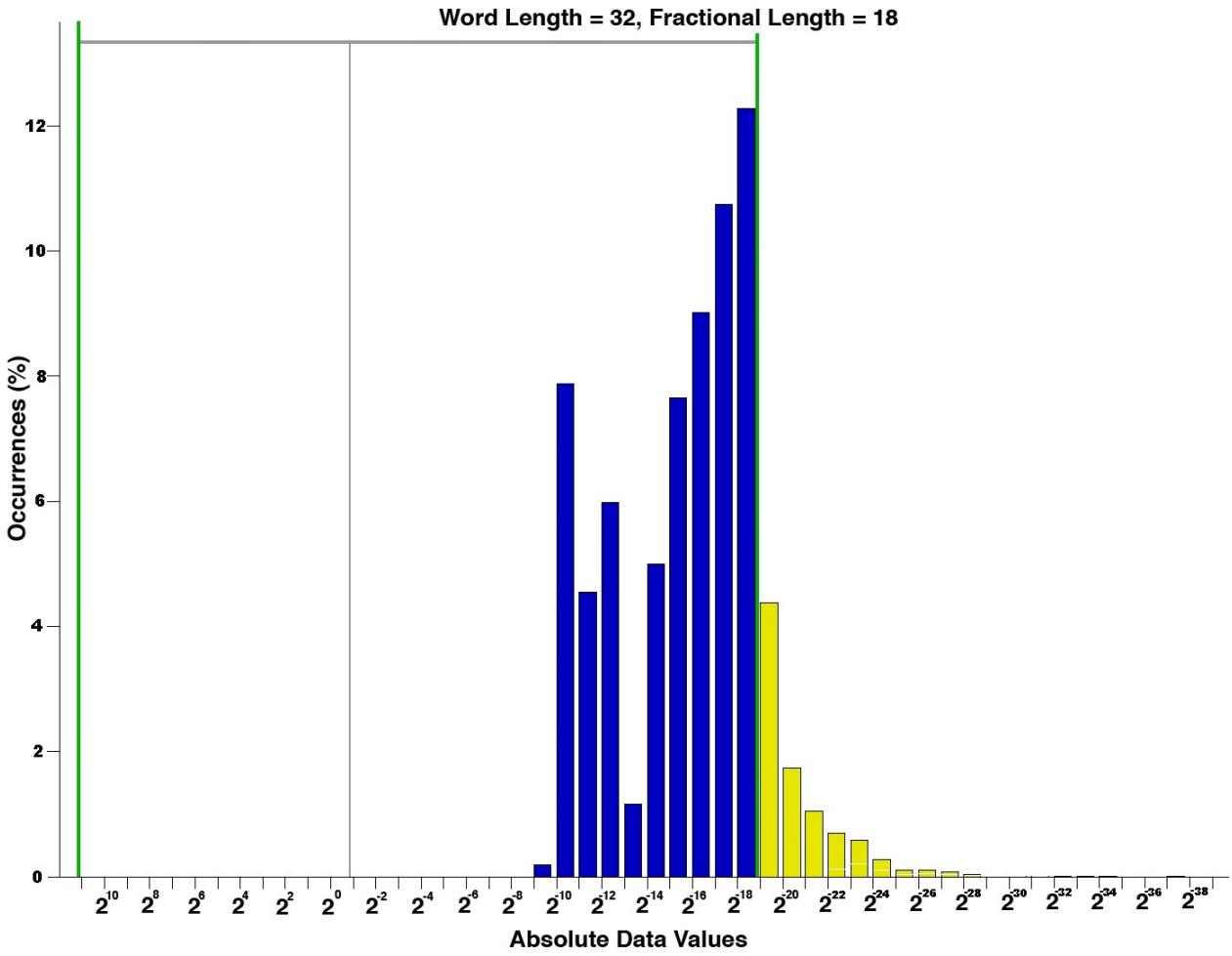

Figure 5.1: *log2* Histrogram: W_M^1

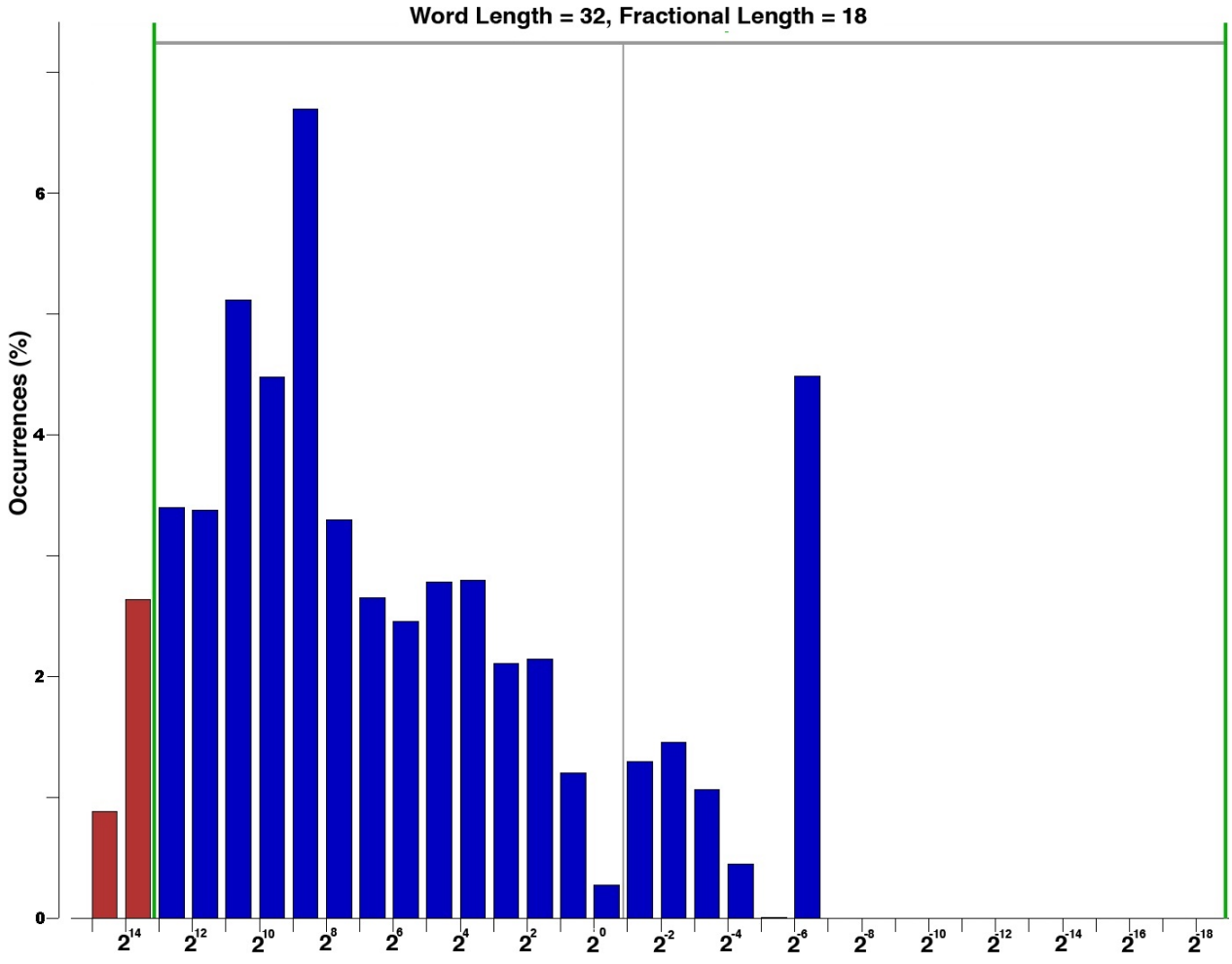

Figure 5.2: *log2* Histrogram: W_C^1

Table 5.1: Data Parameters for Fixed Point Conversion

Data Set	Number of Data Points	Positive Points	Negative Points	Maximum Value	Minimum Value	Average Value
W_C^1	2.0×10^6	1.1×10^6	0	1.9×10^4	0.01	9.7×10^2
W_C^2	2.0×10^6	1.1×10^6	0	1.9×10^4	0.01	9.7×10^2
W_C^3	2.0×10^6	1.1×10^6	0	1.9×10^4	0.01	9.7×10^2
W_C^4	2.0×10^6	1.1×10^6	0	8.2	0.01	2.06
W_C^5	2.0×10^6	1.1×10^6	0	8.2	0.01	2.06
W_C^6	2.0×10^6	1.1×10^6	0	8.2	0.01	2.06
W_M^1	6.8×10^6	4.9×10^6	39000	1.03×10^{-3}	-1.43×10^{-7}	9.07×10^{-5}
W_M^2	2.8×10^6	1.0×10^6	1.8×10^6	0.20	1.1×10^{-3}	6.8×10^{-2}
G_L^1	2.0×10^5	1.4×10^5	5.9×10^4	151	-4.04	10.6
G_L^2	2.0×10^5	1.4×10^5	5.9×10^4	80	-80	24.1
G_L^3	2.8×10^5	2.7×10^5	0	16.9	2.8×10^{-7}	13.2
G_L^4	6.8×10^5	4.7×10^5	2.0×10^5	80	-80	29.6
G_L^5	2.0×10^5	2.0×10^5	0	16.8	5.2×10^{-9}	13.2
G_N^1	2.0×10^5	1.4×10^5	5.9×10^4	1	-1	0.41
G_N^2	2.0×10^5	1.4×10^5	5.9×10^4	1	-1	0.42
G_N^3	2.0×10^5	1.4×10^5	5.9×10^4	1	-1	0.42
e_C	2.0×10^5	1.5×10^5	5.9×10^4	15.15	-0.51	1.9
e_M	2.0×10^5	1.4×10^5	6.2×10^4	2.6	-0.69	3.2×10^{-2}

grows. Choosing 18 and 20 fractional bits, and viewing the rest of the data, we see that either choice will be able to represent the remaining data set. Choosing 18 fractional bits will give us the ability to represent a larger range of data and lose precision for smaller values, whereas 20 fractional bits will give us more precision for small numbers and shrink the data range we can represent. In the end, we choose 20 fractional bits, since this will extend the range of our

representable data.

Table 5.2: Fixed Point Precision

W_M^1 Fractional Bits	Outside Range (%)	Below Precision (%)	W_C^1 Fractional Bits	Outside Range (%)	Below Precision (%)
16	0	32.0	16	0.0	0
18	0	9.0	18	3.5	0
20	0	2.9	20	10.3	0
22	0	1.2	22	19.9	0
24	0	0.3	24	29.9	0
26	0	0.1	26	35.0	0
28	0	0.0	28	40.5	0
30	0	0.0	30	44.8	0

5.2.4 Division

The above weight update law contains divisions in (3.19) and (3.20). Since we are using signed fixed point numbers, the division by 2 in (3.20) can be replaced by a single right shift. The division in (3.19) must be treated more carefully. This architecture is seen in Fig. 5.3. System Generator offers two LogiCore division blocks, both of which are optimized for maximum throughput and minimal latency. Only the 3.0 version is compatible with the Virtex-5 FPGA. This LogiCore block can only do integer division with operands up to 54 bits in length. With our 32 bit word size, we need not worry about truncation error when using this block. We note that instead of dividing P/A in (3.19), we first calculate the reciprocal of A and then multiply by F.

Before using the LogiCore Division 3.0 block we shift the entire number to the left, with the aid of a Bit Basher block, so that we have a 32 bit word with no fractional part. We use the division block to calculate the reciprocal of this number. Since we are calculating the reciprocal,

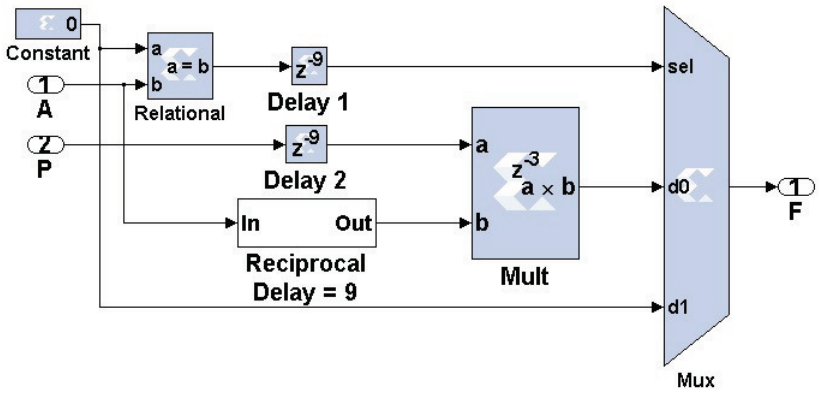

Figure 5.3: System Generator Implementation of (3.19)

we know the integer part of our division output will be 0. The fractional output is set to a 32 bit word. We then right shift this 32 bit word 32-18 times to obtain our result. The latency of the Divider Generator is 6 clock cycles and the output of the divider during the intermediate clock cycles is zero. Thus we must register the last division output. All of this is seen in Fig. 5.4.

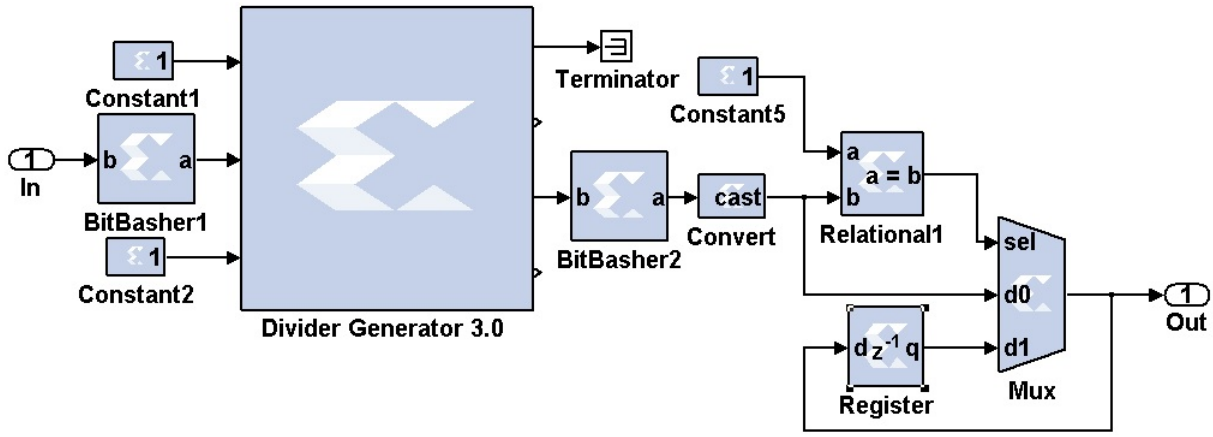

Figure 5.4: Reciprocal Calculations

5.2.5 Nonlinear Neural Function (Tansig)

The first step in building the nonlinear Replicator Unit is building the nonlinear neurons, which is shown in Fig. 5.5.

Next we build the tansig function, which is shown in Fig. 5.6. Analytically, the tansig

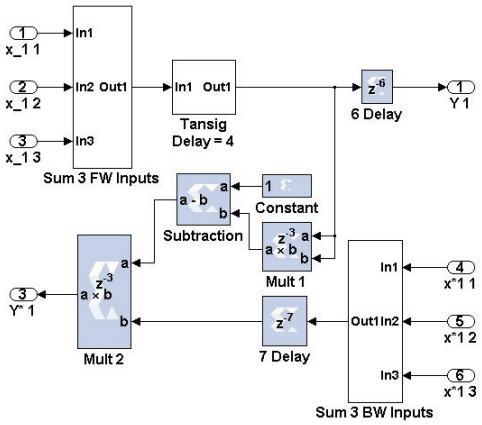

Figure 5.5: Nonlinear Neuron in System Generator

function and its derivative are as follows

$$f(x) = \frac{2}{1 + e^{-2x}} - 1, \tag{5.7}$$

$$f'(x) = 1 - f(x)^2. \tag{5.8}$$

Since the tansig function cannot be implemented exactly in hardware, we must use a lookup table via a single port read only memory (ROM) block. We take advantage of the fact that tansig is odd, i.e., $-f(x) = f(-x)$, and consider only $x \geq 0$. With the tansig function approximately equal to 1 for any value greater than 3 we consider only numbers in $0 \leq x \leq 3$. We break this interval up into three sub intervals: $[0, 1), [1, 2)$, and $[2, 3)$ and use a look up table for each interval. Fig. 5.7, 5.8, and 5.9 below shows the tansig architecture build in System Generator.

5.3 Simulations and "Hardware in the Loop"

In this section, we build the our controller within System Generator. First, we simulate our division and nonlinear neural function design. Next, we build a simple proportional controller. Then we begin building the ANC system. We start by implementing both a linear and nonlinear Replicator Unit. Finally, we build the ANC system. At each step, we simulate our design to ensure its accuracy.

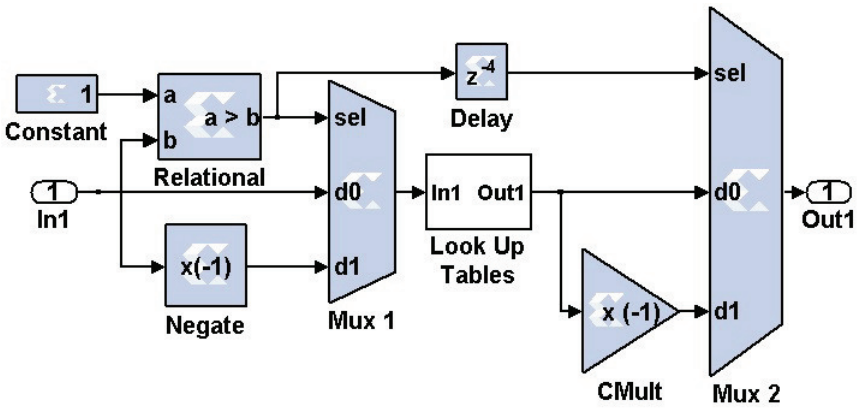

Figure 5.6: Tansig Function in System Generator

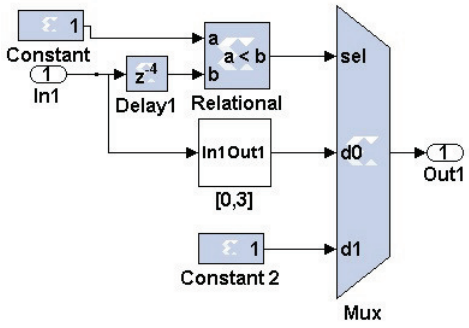

Figure 5.7: Tansig on $[0, 3)$ Interval

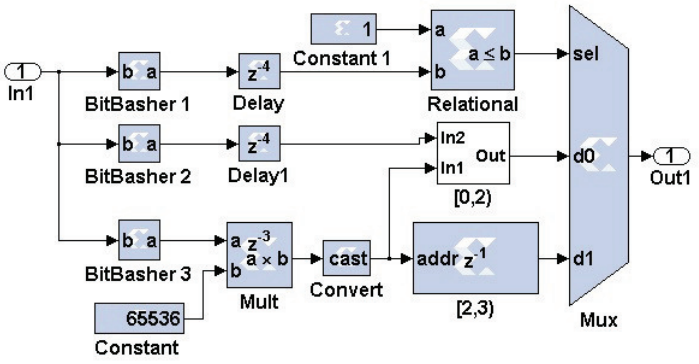

Figure 5.8: Tansig on $[2, 3)$ Interval

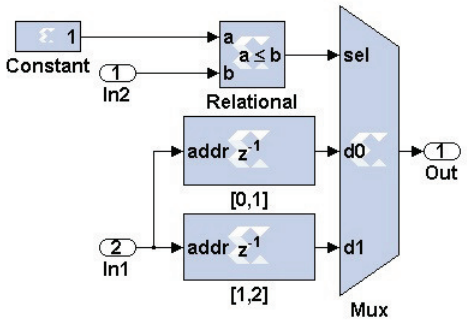

Figure 5.9: Tansig on $[0, 2)$ Interval

5.3.1 Division and Tansig Function

Before proceeding to the ANC system, we must verify that our division and nonlinear neural function design are correct. The results of our "hardware in the loop" division simulation are shown in Fig. 5.10. Here we are taking reciprocal values of the interval $[1, 2]$. We see that our design's output matches that of the actual reciprocal values, except on the delayed interval between computations.

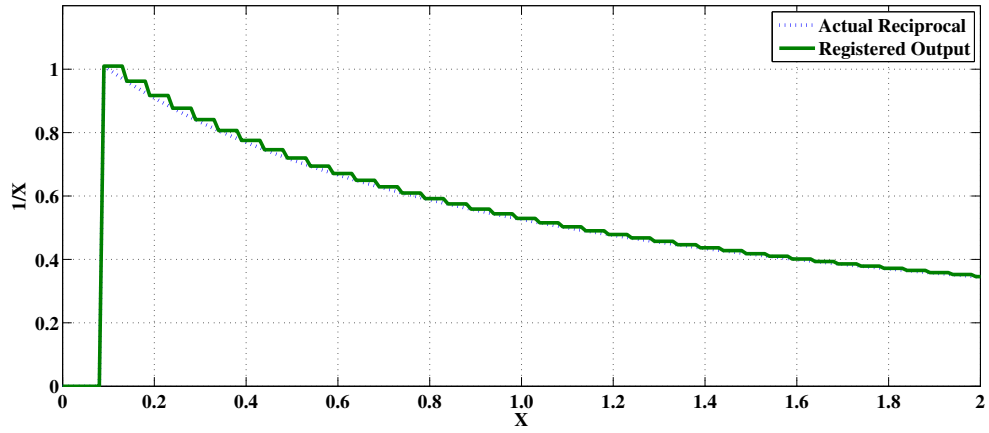

Figure 5.10: Reciprocal

Next, Figs. 5.11 and 5.12 are the "hardware in the loop" simulation output of our look up table shown against the actual values of the tansig function. From Fig. 5.11, we see that the output of our tansig function is almost indistinguishable from the floating point implementation,

except at two points approximately near $x = \pm 3$. Fig. 5.12 shows this discrepancy up close. From this figure we can see that our output differs from the actual output by less than 0.05.

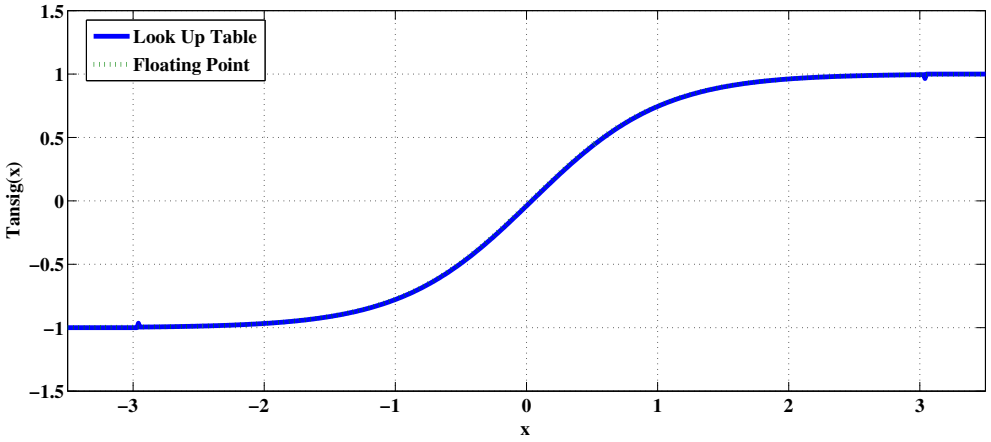

Figure 5.11: Over The Interval $[-3.5, 3.5]$

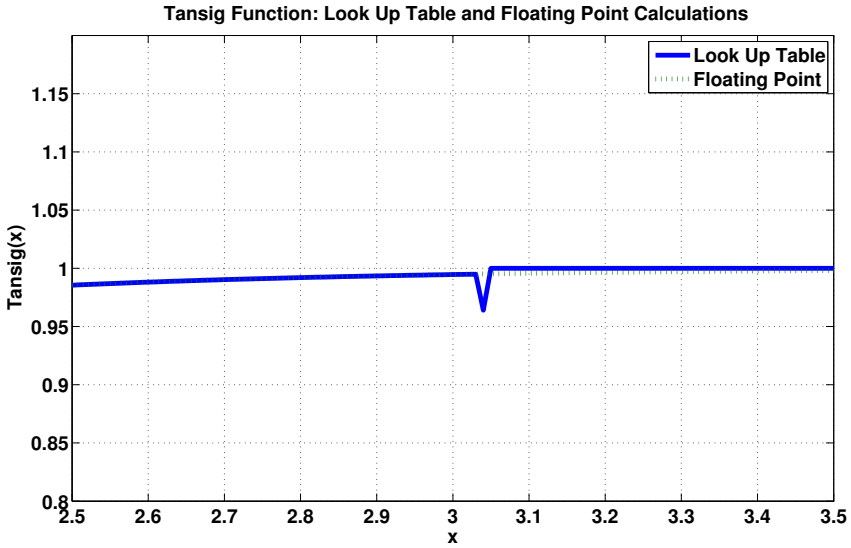

Figure 5.12: Over The Interval $[2.5, 3.5]$

5.3.2 Proportional Controller

Before implementing the ANC system on an FPGA, we first build a basic proportional controller. We us the same proportional controller as the one we implemented on the dSPACE

board. Note, instead of measuring the lasers position on the screen in cm we use pixels. Hence, we change 15.15 in (3.39) to 240 in (5.9). This allows us to simplify our controller design in hardware, since we can now use integer values for position instead of real numbers, which are harder to represent in hardware.

$$u(t) = \frac{0.5}{240} e_p + 0.5. \tag{5.9}$$

In this way, we have that if $e_p = 0$, then $u(t) = 0.5$, which tells the amplifier not to move the gimbal. If $e_p = \pm 240$, then $u(t) = 0$ or 1. So if our error is maximum, then we tell the system to move at full speed towards the center of the screen.

This control law (5.9) was put in hardware via a lookup table. If we ignore the y intercept value 0.5 in (5.9), we see that $u(t)$ is an odd function, i.e. $f(-x) = -f(x)$. Thus, we only need to consider if the input to $u(t)$ is positive or negative, and use the lookup table for the interval $[0, 240]$. This setup is shown in Fig. 5.13.

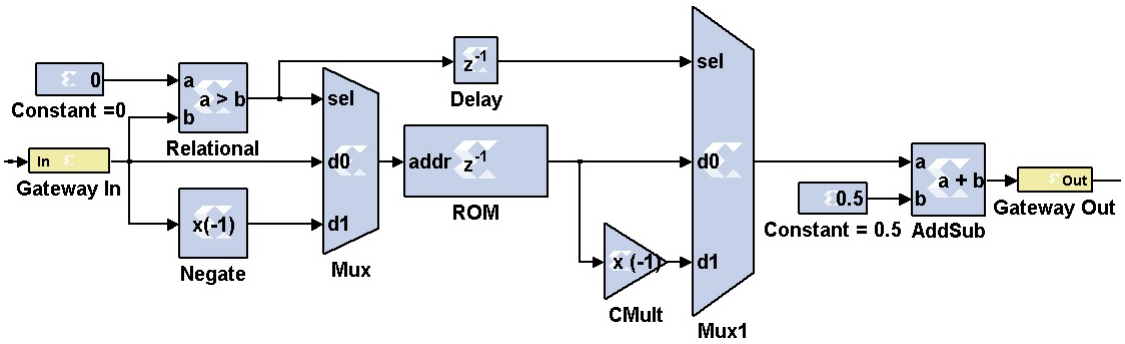

Figure 5.13: Proportional Controller Built in System Generator

To verify our design, we simulated the control law with Simulink, System Generator (via the design above), and then finally on the Virtex 5 with a "hardware-in-the-loop" simulation. Fig. 5.14 shows the result of the Simulink and System Generator simulations. Fig. 5.15 shows the result of the control law running on hardware. In all three, the input to the system was

$$u(t) = t, \;\; -240 \leq t \leq 240.$$

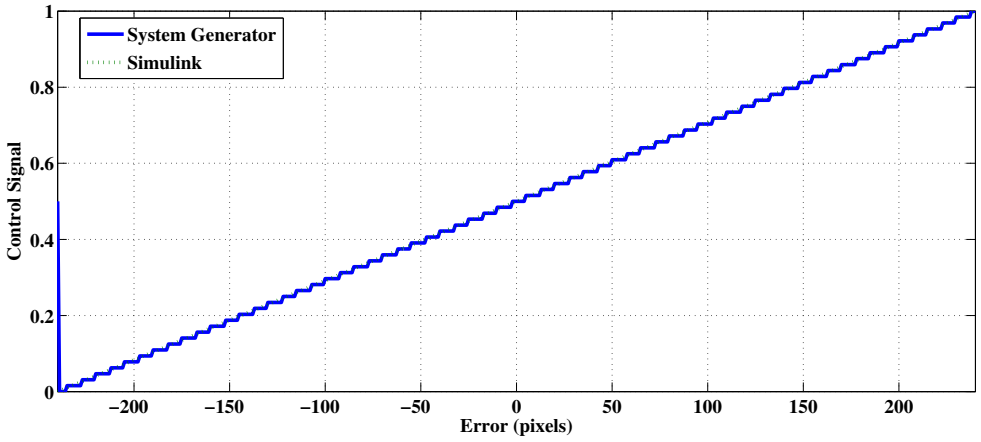

Figure 5.14: Simulink and System Generator Simulation of Proportional Control Law

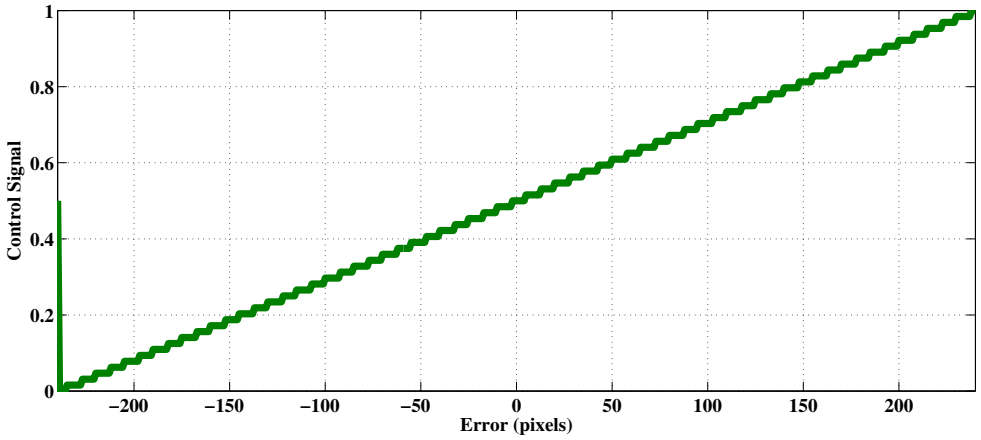

Figure 5.15: “Hardware in the Loop” Simulation of Proportional Control Law

5.3.3 Linear System Replicator Unit

We replicate a linear system using a modified version of the architecture shown in Fig. 3.10. Here we remove the nonlinear ganglia shown in the hidden layer and use only a linear input ganglia and a linear output ganglia. We use the following first order transfer function in our simulation:

$$T_I(s) = \frac{9}{s+9}. \tag{5.10}$$

Fig. 5.16 shows the results of our "hardware in the loop" simulation with the following parameters: 10 neurons per ganglia, $\alpha_L = 0.1$, $J = 10^{-5}$, and all initial weights set to 0.

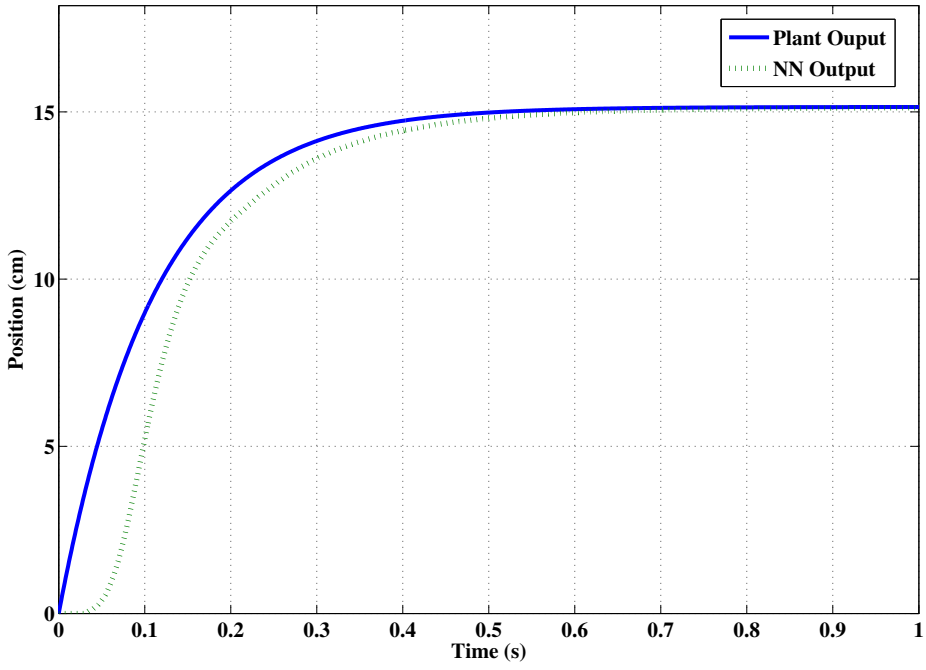

Figure 5.16: Linear System Replicator Unit for "Hardware in the Loop" Simulation

5.3.4 Nonlinear System Replicator Unit

Next we verify our design of the general system Replicator Unit shown in Fig. 3.10. Here we compare simulations run in System Generator to simulations done in our previous Simulink

model. We replicate the following transfer function:

$$T_I(s) = \frac{9}{s+9}. \tag{5.11}$$

This simulation uses both the tansig and reciprocal calculations described above. The results are seen in Fig. 5.17.

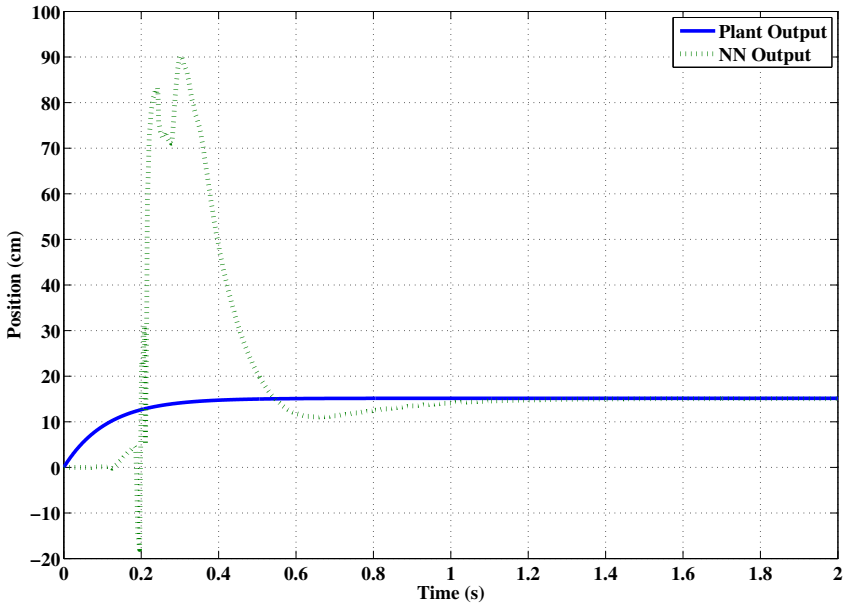

Figure 5.17: Nonlinear System Replication in System Generator

5.3.5 Linear Replicator Unit with Multiplexed Multipliers

Because of the limited hardware resources on the FPGA, we must try to minimize the hardware needed to implement a single Replicator Unit. The number of synapses, within a Toeplitz synapse, connecting two ganglia with n neurons each is $\frac{n(n+1)}{2}$. Since each synapse has a forward and backward signal, we have a total of $n(n+1)$ weights associated with any given Toeplitz synapse. Assume our Replicator Unit has a single ganglia input layer, single ganglia output layer, h hidden layers with k ganglia each, and further assume that all ganglia within the input layer are connected to all ganglia within the first hidden layer, each ganglia is within the ith hidden layer is connected to every ganglia within the $(i+1)$th hidden layer, and all ganglia

within the final hidden layer are connected to all ganglia within the output layer. This gives us a total of $2k + (h-1)k^2$ Toeplitz synapses within a single Replicator Unit, further giving us a total of $\big(2k + (h-1)k^2\big)\big(n(n+1)\big)$ weights per Replicator Unit, if each ganglia contains n neurons. Each one of these weights corresponds to a multiplication at each time step.

Also, for each Toeplitz synapse is a weight update at every time step. This weight update includes three vector norms as well as matrix multiplication. From equation 3.16 we have $n^2+n(n+1)$ multiplications. Equation 3.17 gives us a single multiplication, as does 3.19 if we consider division as multiplication by reciprocal. Equation 3.20 gives us n multiplications and Equation 3.22 gives us n^2+2n. Adding these together gives us $3n^2+4n+2$ total multiplications per Toeplitz synapse per time step.

Within the actual neuron, we see that we have a multiplication within the backward path, adding n more multiplications per time step. Thus, the total number of multiplications needed at each time step for a single Replicator Unit is given as

$$\big(2k + (h-1)k^2\big)\big(4n^2 + 5n + 2\big) + n.$$

For the simulation above of the linear Replicator Unit, we have $n = 10$, $h = 0$ and $k = 1$, giving us a total of 462 multiplications. Note, we take the convention that when $h = 0$, i.e., there are no hidden layers, we set $k = 1$. For the simulation above of the nonlinear Replicator Unit, we have $n = 10$, $h = 1$ and $k = 3$, giving us a total of 2722 multiplications.

In order to avoid maxing out hardware resources due to multipliers, we employ time-division multiplexing. For each design, if originally our ganglia contained n neurons, we condense n multipliers into 1 multiplier in the following way. First, we down-sample the signal n times, run each signal through the first block of a Xilinx Time-Division Multiplexer block pair, multiply the signal, delay the signal $(n-3)$ times, run the signal through the remaining block of the Time-Division Multiplexer block pair, and up-sample the signal n times. This design is seen in Fig. 5.18 with $n = 10$. Again, we replicate the following system

$$T_I(s) = \frac{9}{s+9}.$$

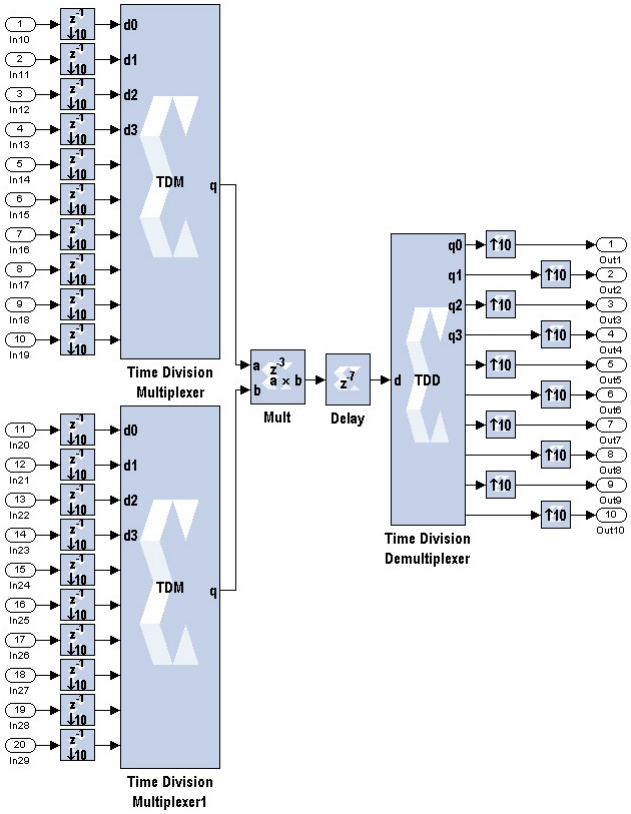

Figure 5.18: Multiplexed Multiplier

The results of this simulation are shown in Figs. 5.19 and 5.20, for $n = 3$ and $n = 10$ respectively. Note, that in the case of $n = 3$, this is a "hardware in the loop" simulation.

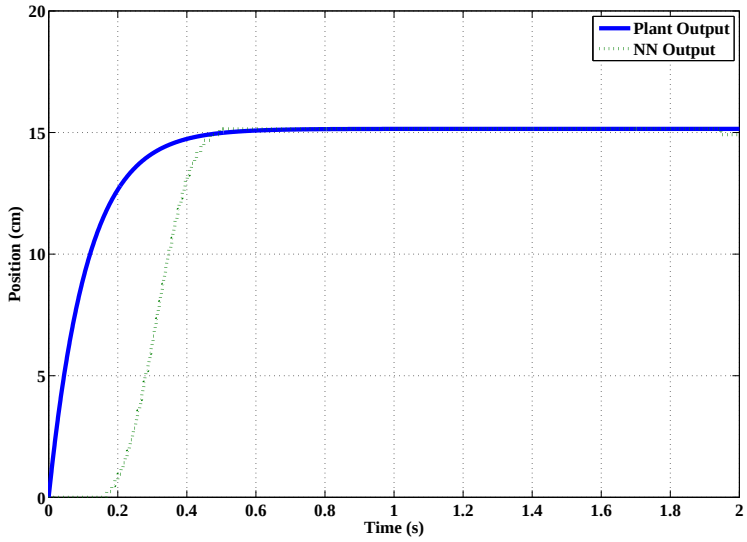

Figure 5.19: System Replication with Multiplexed Replicator Unit, $n = 3$

5.3.6 ANC System

We control the yaw axis of the linear GLTS model and use the following ideal reference system

$$T_I(s) = \frac{9}{s+9}. \tag{5.12}$$

The following parameters are used: all synapse weight values are initialized to 0.001, $\alpha = 0.02$, $\beta_m = 1$, $\beta_c = 0.02$, $J = 1.0 \times 10^{-5}$, and each value is represented as a 32 bit fixed point number with 18 fractional bits.

Fig. 5.21 and 5.22 show the results of our simulation.

All of the simulations above show that our FPGA implementation of the ANC system is able to control the linear model of the GLTS, despite an errors introduced from latencies and floating point to fixed point conversion.

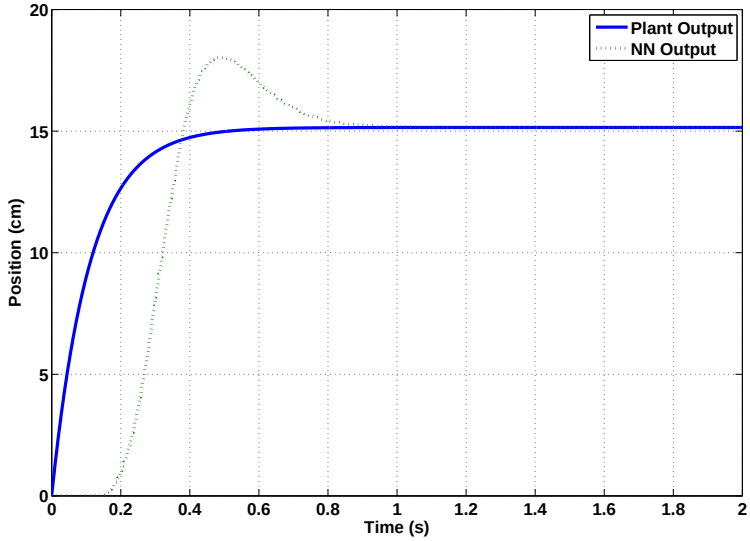

Figure 5.20: System Replication with Multiplexed Replicator Unit, $n = 10$

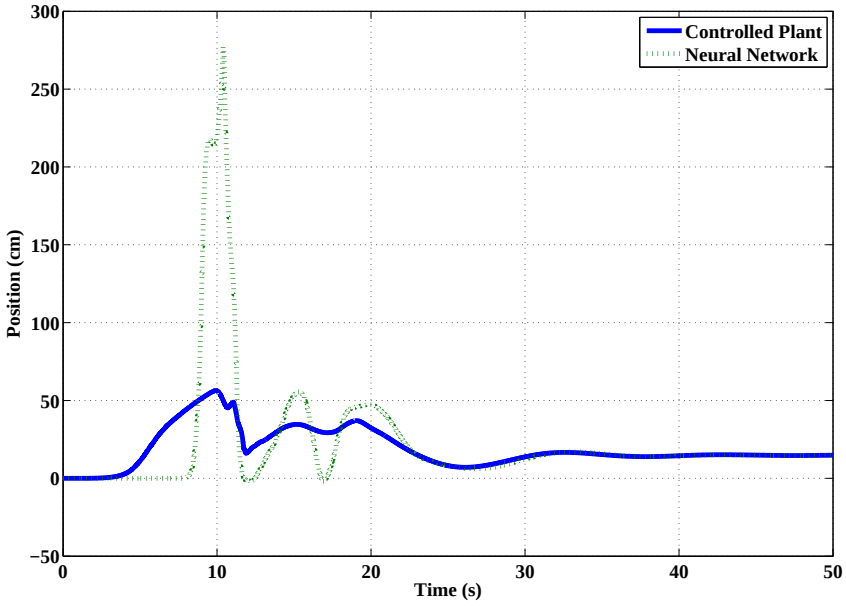

Figure 5.21: ANC in System Generator: System Replication

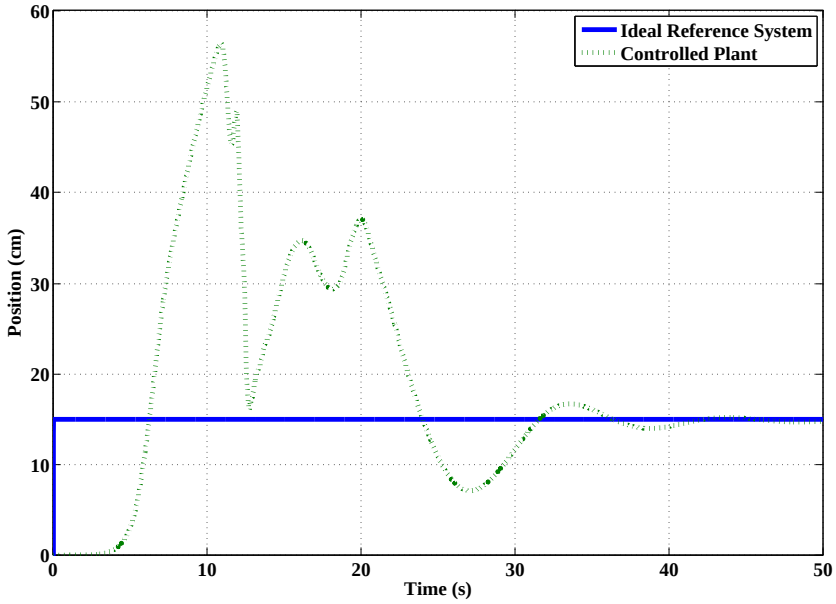

Figure 5.22: ANC in System Generator: Control

5.4 Summary

In this chapter we built the ANC System for implementation on an FPGA. To do this, we first converted our ANC software to handle fixed point arithmetic and properly addressed both division and the nonlinear neural function. From this we were able to build the full ANC system and simulate replication and control of the GLTS.

We see that the hardware model of the ANC system is able to properly identify and control the linear model of the GLTS. We note that this is a simulation of the hardware and not a "hardware in the loop" simulation and this hardware simulation is running with the Simulink clock and not the FPGA clock. Thus, given the the ability to run the hardware model at the same clock rate as the FPGA, we expect the system identification and control to converge faster than shown in these simulations.

Now that we have simulated our hardware model of the ANC System, and showed that we are able to properly control the linear GLTS model, we move on to the problem of hardware implementation for real time system replication of the GLTS test bed.

CHAPTER 6

HARDWARE IMPLEMENTATION OF ANC

6.1 Overview

Here we take initial steps needed to fully implement the ANC system in hardware for real time control of the GLTS. We begin with a description of where the FPGA fits into out test bed. This is followed by a discussion on data transfer and shared memory between the FPGA and our image processing algorithm running in real time in Matlab. Next, as a simple example of hardware implementation, we implement a proportional controller on the FPGA for real time control of the GLTS. After this, we implement a Replicator Unit on the FPGA for real time replication of the output of the GLTS test bed. This FPGA implementation is compared to a dSPACE implementation of the same Replicator Unit design. Finally, we look at the hardware resources needed to implement our Replicator Unit.

6.2 Hardware Test Bed with FPGA

Our hardware test bench is almost identical to the test bench used with the dSPACE processor. The only modification is now, instead of the controller running on the dSPACE processor, we have the controller running on the FPGA, as seen in Fig. 6.1. Note that we are still using the dSPACE board. As the figure shows, we have Matlab sending the current pixel information to the FPGA, which is then run through the control algorithm. The control output is then fed into the dSPACE board, via a serial RS-232 connection. We are using the dSPACE board as an interface to the amplifier. This was done only to easily interface the FPGA with the existing hardware. No processing is done to the control signal by the dSPACE board, other than receiving the control signal and sending it to the amplifier.

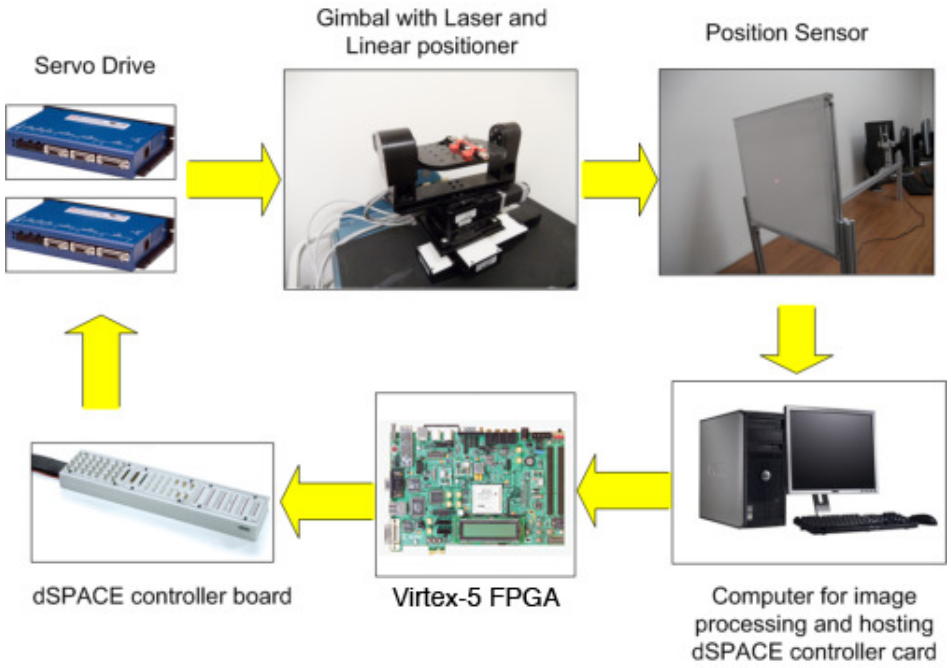

Figure 6.1: Laser Targeting System Hardware with FPGA

6.3 Data Transfer, Clock Mode, and Shared Memory

System Generator allows "hardware in the loop" simulation over an Ethernet connection. This type of connection allows for a high data rate transfer, as opposed to standard programming cables like USB. Additionally, we have the capability of using Point-to-point or network-based communication. The Point-to-point Ethernet connection, which we implement, uses a raw Ethernet connection between the FPGA and host PC, with no network routing equipment needed. The interface supports 10/100/1000 Mbps modes, of which we are using the 1000 Mbps mode. The network-based Ethernet connection supports the same modes but allows the FPGA to connect to the host PC remotely through an IPv4 network.

During a "hardware in the loop" simulation, we can run our FPGA using one of two clocks, a single stepped or a free running clock. A single stepped clock is synced and driving by the Simulink simulation clock. During this type of operational mode, the FPGA clock is completely in sync with the Simulink clock. Additionally, transferring data between the FPGA and host PC is completely synchronized. This is done by providing a single clock pulse to the FPGA

for each Simulink simulation cycle. Essentially, Simulink wakes up the hardware every clock cycle. Thus, if the amount of computation inside the FPGA is significant with respect to the Simulink processing, the FPGA will speed up the simulation.

In the free running clock, the FPGA runs asynchronously with respect to the Simulink simulation. In the single stepped mode, the Simulink clock controlled the hardware clock, whereas in the free running mode, the hardware clock is allowed to run independent of the Simulink clock. While this gives us the ability to process real time data, it also produces synchronization problems. In the single stepped mode, all hand shaking signals and data transfers are taken care of by System Generator. This is not the case with the free running clock. All synchronization mechanisms must be explicitly built into your model when using the free running clock.

Since we will be using the FPGA for real time control, we will use the free running clock. Therefore, we essentially have to separate time domains: the Simulink time domain and the hardware time domain. Within our test bench, Matlab is running real time image processing using data received by a CCD web camera. The sample time of the camera and the simulation time required for the image processing gives us hard bounds on the Simulink time domain. The sample period required to handle both the image processing and incoming data from the camera is 200 Hz. On the other hand, our neural network design has little to no timing constraints, so we can allow the FPGA to run at full speed, which is 100 Mhz.

In order to supply the neural network with real time data, and allow the host PC to receive the data immediately upon exiting the neural network, we take advantage of System Generator's ability to produce shared memory. This interface allows for mapping between FPGA memory and common address spaces on the host PC. We implement the shared memory using To FIFO and From FIFO blocks. When a shared FIFO block is generated for a "hardware in the loop" simulation, a single asynchronous FIFO core replaces the blocks. One side of the core is connected to the host PC while the other half of the core is connected to the FPGA. Fig. 6.2 shows a To FIFO block compiled for a "hardware in the loop" simulation. Here, the write side of the FIFO (shown in blue) is connected to the FPGA design while the read side is connected to the host PC. The opposite is seen in Fig. 6.3. Here the read side (shown in blue) is connected to the

FPGA while the write side is connected to the host PC. Thus, the "to" and "from" refer to the action of the FPGA, i.e, write to and read from, respectively.

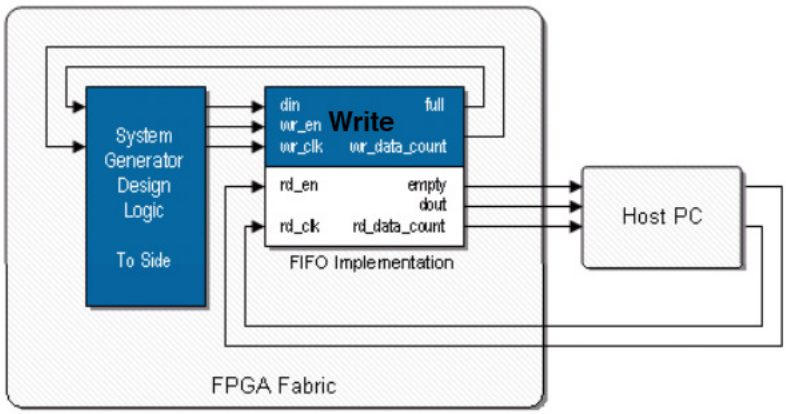

Figure 6.2: A To FIFO Block Compiled for a "Hardware in the Loop" Simulation

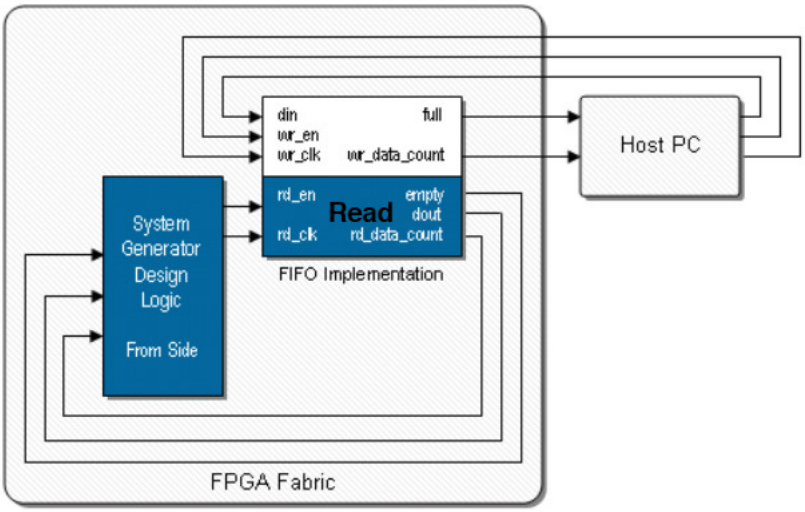

Figure 6.3: A From FIFO Block Compiled for a "Hardware in the Loop" Simulation

The To FIFO and From FIFO blocks compiled for a "hardware in the loop" simulation then have the other half of their pair (From FIFO and To FIFO, respectively) within the System Generator design. These two pairs then take care of any clocking issues when transmitting or receiving data. When data is written to the To FIFO block within the Simulink simulation, the data appears in the From FIFO block on the FPGA. Similarly, when data is written to the

To FIFO block on the FPGA, the same data appears on the From FIFO block running in the Simulink Simulation. The timing synchronization is taken care of during the System Generator compilation process.

6.4 Proportional Controller

We implement the proportional controller described above, with no added noise. Fig. 6.4 shows the of this experiment. We include the results from the same controller implemented in dSPACE for comparison.

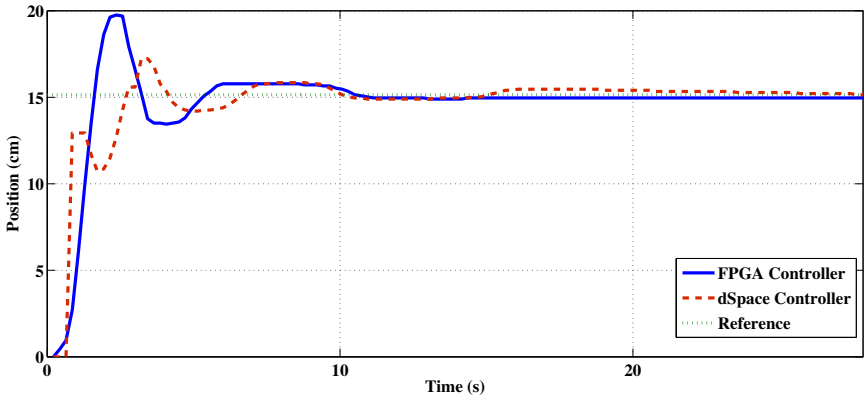

Figure 6.4: Proportional Control: FPGA and dSPACE

Next, we implement the same proportional controller, with an added disturbance. The results are seen in Fig. 6.5. Again, we include the results of the same controller implemented in dSPACE for comparison.

Table 6.1 gives shows the maximum value, root mean square pointing error (RMSPE), and standard deviation of pointing error (SDPE) of the two proportional controllers.

Table 6.1: Proportional Controller Statistics: FPGA and dSPACE

Controller	Maximum Value (cm)	RMSE	SDPE
dSPACE	30.93	10.17	9.44
FPGA	17.8	1.56	1.57

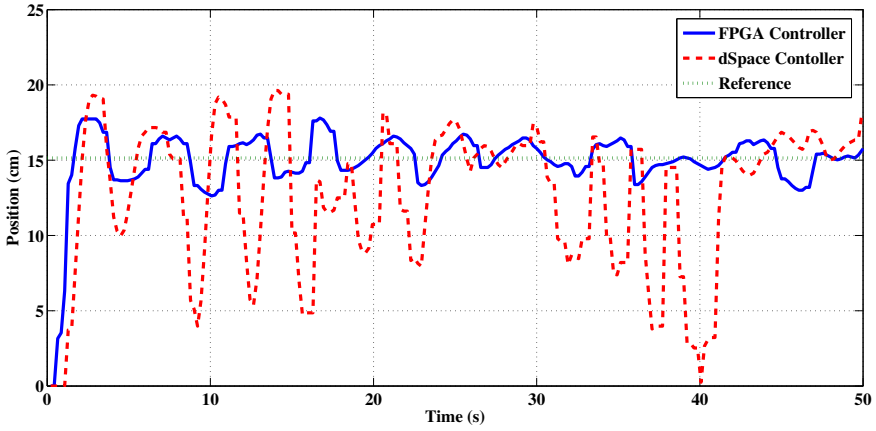

Figure 6.5: Proportional Control with Noise: FPGA and dSPACE

6.5 ANC: System Replication

We implement a single linear FIR Replicator Unit for real time system replication of the GLTS with added noise. In order to compare this experiment with the system replication done on the dSPACE board, we use the same parameters: 3 neurons per ganglia, $\alpha = 0.001$, $J = 10^{-5}$, and all initial weight values equal to 0. The one difference between this experiment and the dSPACE experiment, is that we are running on the 66.667 MHz FPGA clock. Figs. 6.6 and 6.7 show the result of this simulation. Note that Fig. 6.7 shows the initial response of the neural network and converges after approximately .

In order to compare the performance of the FPGA and dSPACE boards, we calculate the percent overshoot (PO), and the root mean square reference error (RMSE), from 3.36. We define percent overshoot as:

$$PO = \frac{\text{max value} - \text{step value}}{\text{step value}}.$$

Table 6.2 gives the results for these two experiments.

6.6 ANC: Hardware Resources

Before implementing the entire ANC system, we implement a linear FIR Replicator Unit, with 10 neurons per ganglia, in order to replicate the GLTS. While trying to implement this on

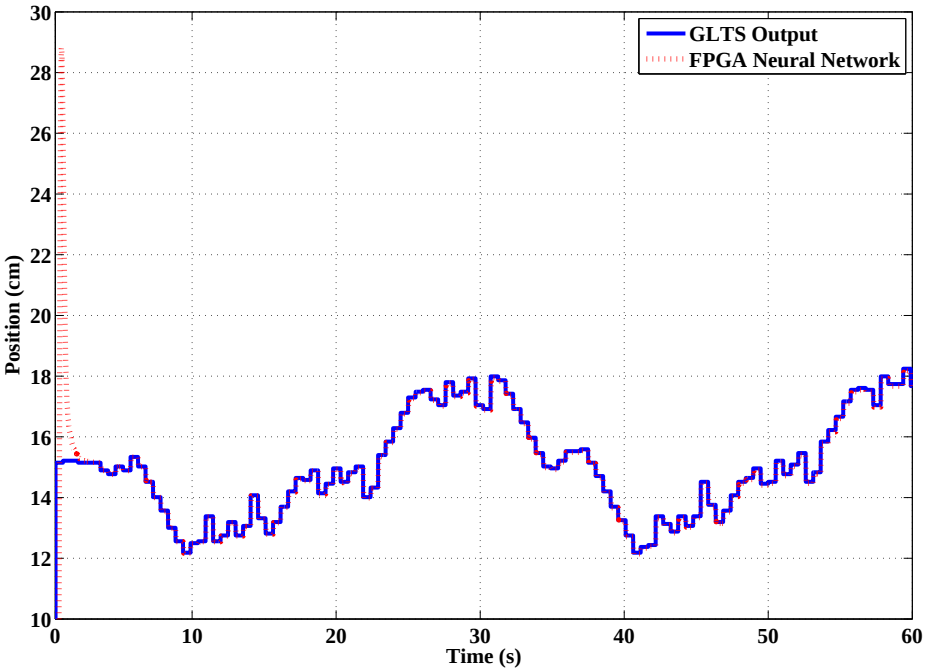

Figure 6.6: Proportional Control with Noise: FPGA and dSPACE

Table 6.2: Performance of System Replication on FPGA and dSPACE

Hardware	Max Value	Step Value	Percent Overshoot	RMSE
dSPACE	17.72	15.91	0.114	0.035
FPGA	28.8	15.15	0.901	0.030

the FPGA hardware, we begin to max out the FPGA's hardware resources, as a result of the number of multipliers used within the design. While each multiplier can be optimized for speed or area, both of these optimizations use more than 100%. The hardware resources used for this design is given in Table 6.3.

Next, we implement our multiplexed multiplier design for a linear FIR System Replicator Unit. While this design reduced the total number of multipliers needed, it also increased the delays throughout the system. The non-multiplexed design had a delay of 3 for each multiplication, while the multiplexed design had a delay of $3n$ for each multiplication, where n is the

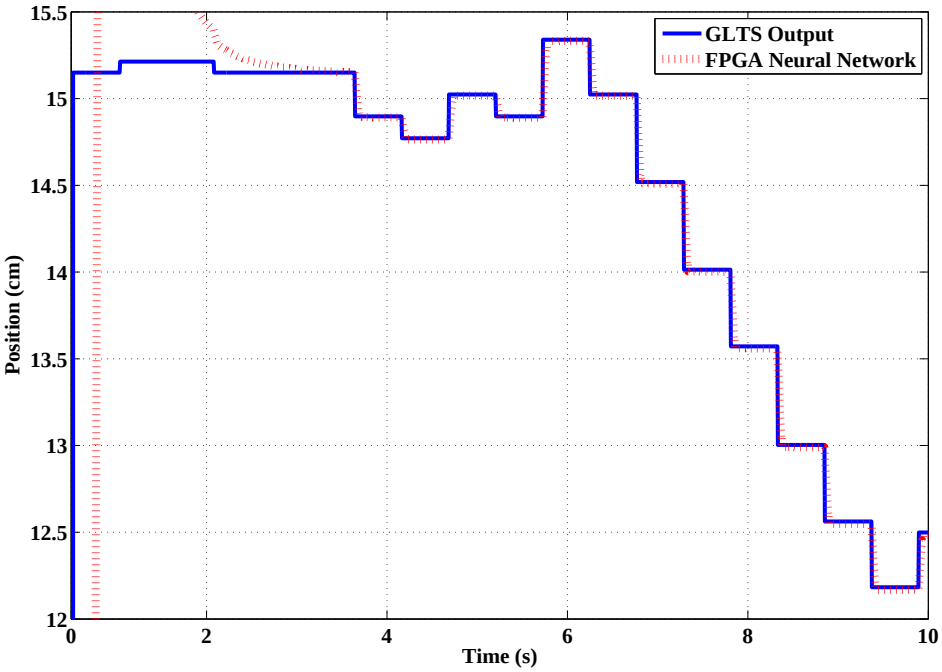

Figure 6.7: Proportional Control with Noise: FPGA and dSPACE

number of neurons per ganglia. These added delays begin to introduce timing errors within the compilation process, forcing us to reduce the clock speed of the FPGA. Tables 6.4 and 6.5 show the hardware resources needed and timing errors for a multiplexed linear FIR System Replicator Unit with 5 and 3 neurons per ganglia, respectively. Each of these tables considers two clock speeds (100 and 66.667 MHZ) and two optimization processes (area and speed).

Finally, we try to compile our design for implementation on devices other than the Virtex 5 FPGA. We consider both the Virtex 4 and Virtex 6 FPGAs and try to implement a linear FIR multiplexed Replicator Unit, with 3 neurons per ganglia at 100 MHz. For both devices we optimize for both speed and area. The results of these simulations are given in Table 6.6.

6.7 Summary

In this chapter we have taken the first steps to a full hardware implementation of the ANC system for real time control. We first considered data transfer and shared memory between the FPGA and a PC running our image processing algorithm. Next, we implemented a proportional

Table 6.3: Hardware Used in FPGA for a Linear FIR Replicator Unit, 10 Neurons per Ganglia, 100 MHz

Optimization	Hardware	Number Used	Total Available	Percentage Used
Area	Slice Registers	51253	32640	157%
	Slice LUTs	193887	32640	594%
	DSP48e Slices	317	288	110%
Speed	Slice Registers	16866	32640	51%
	Slice LUTs	12869	32640	39%
	DSP48s Slices	1210	288	420 %

controller in hardware via an FPGA and dSPACE control board. Finally, we implemented a Replicator Unit for real time system replication of the GLTS, again, via an FPGA and a dSPACE control board. Finally, we examined the hardware resourced used for various Replicator Unit design on different FPGA evaluation boards.

At this point, we have stated the results of all of our contributed work. In the remainder of the thesis, we will summarized these results and give our conclusions, and end with possible solutions to the problem of full hardware implementation of the ANC system for real time control.

Table 6.4: Hardware Used in FPGA for a Linear FIR Multiplexed Replicator Unit, 5 Neurons Per Ganglia

Clock (MHz)	Optimization	Hardware	Number Used	Total Available	Percentage Used (%)	Number of Timing Errors
100	Area	Slice Registers	21083	32640	64	3
		Slice LUTs	24687	32640	75	
		DSP48s Slices	49	288	17	
	Speed	Slice Registers	18507	32640	56	3
		Slice LUTs	10778	32640	33	
		DSP48s Slices	118	288	40	
66.667	Area	Slice Registers	21083	32640	64	1
		Slice LUTs	24687	32640	75	
		DSP48s Slices	49	288	17	
	Speed	Slice Registers	18861	32640	57	1
		Slice LUTs	12565	32640	38	
		DSP48s Slices	116	288	40	

Table 6.5: Hardware Used in FPGA for a Linear FIR Multiplexed Replicator Unit, 3 Neurons Per Ganglia

Clock (MHz)	Optimization	Hardware	Number Used	Total Available	Percentage Used (%)	Number of Timing Errors
100	Area	Slice Registers	10392	32640	31	3
		Slice LUTs	14620	32640	44	
		DSP48s Slices	37	288	12	
	Speed	Slice Registers	7512	32640	23	1
		Slice LUTs	4120	32640	12	
		DSP48s Slices	78	288	27	
66.667	Area	Slice Registers	9809	32640	30	0
		Slice LUTs	12831	32640	39	
		DSP48s Slices	39	288	13	
	Speed	Slice Registers	7512	32640	23	0
		Slice LUTs	4120	32640	12	
		DSP48s Slices	78	288	27	

Table 6.6: Hardware Used for Virtex 4 and Virtex 6 FPGAs

Device	Optimization	Hardware	Number Used	Total Available	Percentage Used (%)
Virtex 6 ML605	Area	Slice Registers	9435	301440	3
		Slice LUTs	15131	150720	10
		DSP48E1s Slices	37	768	4
	Speed	Slice Registers	7607	301440	2
		Slice LUTs	6603	150720	4
		DSP48E1s Slices	76	768	9
Virtex 4 ML402	Area	Slice Flip Flops	10416	30720	33
		4 Input LUTs	18173	30720	59
		DSP48Es Slices	37	192	19
	Speed	Slice Flip Flops	8558	30720	27
		4 Input LUTs	7953	30720	25
		DSP48Es Slices	76	192	39

CHAPTER 7

SUMMARY, CONCLUSIONS, AND FUTURE WORK

7.1 Overview

The goal of this thesis was to implement the ANC system in software and hardware. We first implemented the ANC system in software and simulated control of the linear stochastic GLTS model as well as for a separate nonlinear stochastic two-axis gimbal model. Next, we examined the resiliency of the ANC system through simulation, with the linear GLTS model exposed to various attacks. After this, we simulated a hardware model of the ANC system for use on an FPGA. Finally, we used our hardware model of the ANC system and performed real time system replication of the GLTS test bed.

In this chapter, we restate all of the above results and give our conclusions. We end with possible solutions to the problem of full implementation of the ANC system in hardware for real time control.

7.1.1 Control of GLTS: PID and ANC

Our simulations show that both the PID and ANC controllers are able to properly control the GLTS to point to the reference signal. When the GLTS is subjected to process noise $W_p = 10^{-1}$ and measurement noise $W_m = 10^{-6}$, the PID controller fails. This is to be expected since a PID controller does not consider a stochastic system. On the other hand, the ANC controller is still able to properly maintain control. We note that the difference in the control capabilities of the ANC system is small when considering the GLTS with and without noise. The statistics of the PID and ANC simulations are restated in Table 7.1.

Our control performance metrics are pointing / reference error, root mean square pointing /reference error (RMSPE/RMSRE), and standard deviation of the pointing / reference error

(SDPE/SDRE). The pointing error is the difference between the reference point (15.15 cm) and the actual position co-ordinates and SDPE/SDRE indicates the variation of the error from the mean error value. We calculated the RMSPE and SDPE using the following equations, respectively,

$$\mu_\varepsilon = \sqrt{\frac{\sum_{t=t_0}^{t_f} \varepsilon^2(t)}{N_\varepsilon}},$$

$$\sigma_\varepsilon = \sqrt{\frac{\sum_{t=t_0}^{t_f} (\varepsilon(t) - \mu_\varepsilon)^2}{N_\varepsilon - 1}}.$$

Here ε is the error between the set-point position co-ordinate and the current position co-ordinate, N_ε is the number of samples.

Table 7.1: PID and ANC Controller Statistics

Controller	Maximum Value (cm)	RMSE	SDPE
PID	36.32	27.0	20.6
ANC	19.3	1.59	1.58

Additionally, the ANC system was able to properly identify and control a separate nonlinear stochastic gimbal model. The results of this simulation are restated in Table 7.2.

Table 7.2: ANC Controller Statistics for Nonlinear Gimbal

Maximum Value (cm)	RMSE	SDRE
3.19	0.19	0.28

In conclusion, we see that for our linear GLTS model, the ANC system outperforms the PID controller, in terms of RMSPE and SDPE, for both a stochastic (added measurement and process noise) and non-stochastic system. We also conclude that the ANC system is able to properly control the nonlinear two-axis gimbal model, with added measurement and process noise, in terms of RMSPE and SDPE. We emphasize that in both the linear and nonlinear simulations, the ANC system had no prior modeling information of the system to be controlled.

7.1.2 Resiliency of ANC System

Table 7.3 shows the results of our resiliency simulations.

Table 7.3: Resilient Metrics for PID and ANC Controllers

Metric	Added Latency		False Data Injection		Sensor Data Alteration		Plant Parameter Changes	
	PID	ANC	PID	ANC	PID	ANC	PID	ANC
T_i^r	16.84 s	0.65 s	∞	1.5 s	20 s	1 s	70 s	∞
P_i^d	-2.92 cm	-0.92 cm	∞	-2.85 cm	-10.45 cm	-9.95 cm	13.55 cm	7.58 cm
T_i^p	0 s	0 s	0 s	0 s	0 s	0 s	0 s	0 s
T_i^d	0.86 s	0.65 s	∞	0.7 s	0 s	0 s	0 s	0 s

From these results, we conclude that in terms of protection time, the performance of both the PID and ANC system are equal. From the remaining three resilient metrics (performance degradation, recovery time, and degrading time), we conclude that the ANC system is more resilient to each of the attacks than the PID controller, with the exception of the Plant Parameter Change attack. In this attack, the ANC system is never able to fully recover, oscillating around the 15.15 cm mark.

7.1.3 ANC using FPGA

We began our software to hardware conversion by first considering floating point to fixed point conversion, division, and nonlinear neural function in detail. Next, we performed both simulations and "hardware in the loop" simulations of our division design, nonlinear neural function, Replicator Unit designs (linear, nonlinear, and multiplexed multiplier designs), and finally, the full ANC system for control of the linear GLTS model. Simulations showed that at each step, our subsystems performed as desired. In the control simulation, we saw that the hardware version of the ANC system was able to properly control output of the linear GLTS model to that of our ideal reference system within 60 seconds.

From these results, we can conclude that our software to hardware conversion of the ANC system was successful, since each of our hardware subsystems behaves as expected. We also conclude that, while we do see an added delay when compared to our software model, the hardware model of the ANC system is able to properly control the linear GLTS model.

7.1.4 Hardware Implementation of ANC

We restate the hardware resources used for a single linear Replicator Unit design with 10 neurons per ganglia in Table 7.5 and the results from our real time system replication experiments in Table 7.4. From Table 7.5 we see that a single linear Replicator Unit can significantly max out our available hardware resources. To circumvent this problem, we redesigned the Replicator Unit to use less multipliers in both the weight update law and when synapse output is multiplied by its weight. While this new design reduces the amount of hardware needed, we still see that certain types of implementations still max out our hardware. Only when we consider a Replicator Unit with 3 neurons per ganglia, and a reduced number of multipliers are we able to run the Replicator Unit in real time on our hardware test bed. When this experiment is compared to the real time dSPACE replicator experiment, we see that we initially have a higher overshoot, but when comparing RMSPE, both the FPGA and dSPACE experiments give similar results, as seen in Table 7.4.

Table 7.4: Performance of System Replication on FPGA and dSPACE

Hardware	Max Value	Step Value	Percent Overshoot	RMSE
dSPACE	17.72	15.91	0.114	0.035
FPGA	28.8	15.15	0.901	0.030

In conclusion, the results validate our hardware model of the ANC system in terms of real time system replication of the GLTS test bed. Further, we conclude that this hardware model running in real time on an FPGA performs comparably to our previous model of the ANC system running in real time on the dSPACE control board.

Table 7.5: Hardware Used in FPGA for a Linear FIR Replicator Unit, 10 Neurons per Ganglia, 100 MHz

Optimization	Hardware	Percentage Used
Area	Slice Registers	157%
	Slice LUTs	594%
	DSP48e Slices	110%
Speed	Slice Registers	51%
	Slice LUTs	39%
	DSP48s Slices	420 %

7.2 Future Work: Hardware Implementation for Real Time Control

From the above discussion, we see that a single linear Replicator Unit with 10 neurons per ganglia maxes out the hardware resources of the Virtex 5 FPGA. Additionally, we saw that the full ANC system also maxed out the hardware available on the dSPACE board, in that the dSPACE board was not able to run the full control algorithm in real time at the necessary sample rate. We note that while the Virtex 5 FPGA is not trivial processor, we are using an evaluation board, with multiple peripheral devices attached, which are not needed for our purposed. Considering the hardware test beds which the original ANC system was implemented on, the NASA / LaRC Mini-MAST test bed and the ASTREX test bed at Airforce Philips Laboratory, we see that an off-the-shelf processor might not be the ideal situation for real time hardware implementation. To this end, we offer two possible solutions.

Since we have already designed a hardware model for implementation on an FPGA, one option to consider is the WILDSTAR 6 PCIe card offered by Annapolis Micro Systems, Inc. This board consists of three interconnected Virtex 6 FPGAs as well as a PCI connection, which could easily interface with the dSPACE board and the amplifiers in our test bed. This board could give us a separate FPGA for each of the adapting Replicator Units within the ANC system.

A similar processor which contains multiple FPGAs on a single board is the DNK7 F5 8

board, offered by Dini Group. This board contains four Xilinx Kintex-7 FPGAs. If even further computational power is needed, the DNK7 F5 8 Cluster Algorithm Acceleration System, also offered by Dini Group, contains eight of the above boards for a total forty five Kintex-7 FPGAs.

REFERENCES

Anthony, M. (1999). *Neural Network Learning: Theoretical Foundations*. Cambridge University Press.

Biswas, S., Ferrese, F., Dong, Q., and Bai, L. (2012). Resilient consensus control for linear systems in a noisy environment. In *American Control Conference (ACC), 2012*, pages 5862–5867.

Blanke, M., Frei, C., Kraus, F., Paton, R., and Staroswiecki, M. (2000). What is fault tolerant control? In *IFAC Symposium on Fault Detection Supervision and Safety for Technical Processes*, pages 40–51, Budapest, Hungary.

Boring, R. (2009). Reconciling resilience with reliability: The complementary nature of resilience engineering and human reliability analysis. *Human Factors and Ergonomics Society Annual Meeting Proceedings*, 53:1589–1593.

Butchart, R. and Shackcloth, B. (1966). Synthesis of model reference adaptive control systems by lyapunov's second method. In *Proc. 1965 IFAC Symp. Adaptive Control*, pages 145–152, Teddington, England.

Colavito, L. and Silage, D. (2009). Composite look-up table gaussian pseudo-random number generator. In *Reconfigurable Computing and FPGAs, 2009. ReConFig '09. International Conference on*, pages 314–319.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.

Davis, L. and Hyland, D. (1997). Adaptive neural control for the astrex testbed. In *American Control Conference, 1997. Proceedings of the 1997*, volume 3, pages 1794–1798 vol.3.

Dietz, R., Center, G. C. M. S. F., Center, L. B. J. S., Aeronautics, U. S. N., Scientific, S. A., and Branch, T. I. (1981). *Satellite power system: concept development and evaluation program*. Number v. 3 in NASA reference publication. National Aeronautics and Space Administration, Scientific and Technical Information Branch.

Ekstrand, B. (2001). Equations of motion for a two-axes gimbal system. *IEEE Transactions on Aerospace and Electronic Systems*, 37(3):1083 –1091.

Freeman, J. and Skapura, D. (1991). *Neural networks: algorithms, applications, and programming techniques*. Addison-Wesley.

Giorgi, S., Saleheen, F., Ferrese, F., and Won, C.-H. (2012). Adaptive neural replication and resilient control despite malicious attacks. In *Resilient Control Systems (ISRCS), 2012 5th International Symposium on*, pages 112–117.

Glaser, P. (1968). Power from the sun: Its future. *Science*, 162:857–861.

Hagan, M. and Demuth, H. (1999). Neural networks for control. In *American Control Conference, 1999. Proceedings of the 1999*, volume 3, pages 1642–1656 vol.3.

Hang, C. and Parks, P. (1973). Comparative studies of model reference adaptive control systems. *Automatic Control, IEEE Transactions on*, 18(5):419–428.

Holt, J. and Baker, T. (1991). Back propagation simulations using limited precision calculations. In *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*, volume ii, pages 121–126 vol.2.

Hyland, D. (1991). Neural network architecture for online system identification and adaptively optimized control. In *Decision and Control, 1991., Proceedings of the 30th IEEE Conference on*, pages 2552–2557 vol.3.

Hyland, D. (1995). Adaptive neural control for flexible aerospace systems: progress and prospects. In *Intelligent Control, 1995., Proceedings of the 1995 IEEE International Symposium on*, pages 3–8.

Hyland, D. (1997). Connectionist algorithms for identification and control - system structure and convergence analysis. In *Aerospace Sciences Meeting and Exhibit, 35th*, volume 3.

Hyland, D. (1998). Multiprocessor system and method for indentification and adaptive control of dynamic systems.

Jian, B., LiangJie, Z., and Yi, Y. (2012). Analysis on complexity of neural networks using integer weights. *Applied Mathematics and Information Sciences*, 6(2):317–323.

Jin, X., Ray, A., and Edwards, R. (2010). Integrated robust and resilient control of nuclear power plants for operational safety and high performance. *Nuclear Science, IEEE Transactions on*, 57(2):807–817.

Kennedy, P. and Kennedy, R. (2003). Direct versus indirect line of sight (los) stabilization. *Control Systems Technology, IEEE Transactions on*, 11(1):3 – 15.

Krishnan, R. (2001). *Electric motor drives: modeling, analysis, and control*. Prentice Hall.

Landau, I. D., L. R. M. M. . K. A. (2011). Introduction to adaptive control. In *Adaptive Control: Algorithms, Analysis and Applications*, Communications and Control Engineering, pages 1–33. Springer London.

Lange, R. (2005). Design of a generic neural network fpga-implementation.

Lee, T., Ge, S., and Wong, C. (1998). Adaptive neural network feedback control of a passive line-of-sight stabilization system. *Mechatronics*, 8(8):887–903.

Lin, C. and Hsiao, Y. (2001). Adaptive feedforward control for disturbance torque rejection in seeker stabilizing loop. *Control Systems Technology, IEEE Transactions on*, 9(1):108–121.

MacKunis, W., Dupree, K., Bhasin, S., and Dixon, W. (2008). Adaptive neural network satellite attitude control in the presence of inertia and cmg actuator uncertainties. In *American Control Conference, 2008*, pages 2975–2980.

Mankins, J. C. (1997). A fresh look at space solar power: New architectures, concepts and technologies. *Acta Astronautica*, 41(410):347 – 359.

Marchesi, M., Orlandi, G., Piazza, F., and Uncini, A. (1993). Fast neural networks without multipliers. *Neural Networks, IEEE Transactions on*, 4(1):53–62.

Narendra, K. (1996). Neural networks for control theory and practice. *Proceedings of the IEEE*, 84(10):1385–1406.

Omondi, A. and Rajapakse, J. (2006). *FPGA Implementations of Neural Networks*. Springer.

Osburn, P., Whitaker, H., and Keezer, A. (1961). New developments in the design of model reference adaptive control systems. In *Inst. Aeronautical Sciences*.

Parks, P. (1966). Lyapunov redesign of model reference adaptive control systems. In *IEEE Trans. Automat. Contr.*, pages 362–367.

Plagianakos, V. and Vrahatis, M. (1999). Neural network training with constrained integer weights. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3, pages –2013 Vol. 3.

Rieger, C. (2010). Notional examples and benchmark aspects of a resilient control system. In *Resilient Control Systems (ISRCS), 2010 3rd International Symposium on*, pages 64–71.

Rieger, C., Gertman, D., and McQueen, M. (2009). Resilient control systems: Next generation design research. In *Human System Interactions, 2009. HSI '09. 2nd Conference on*, pages 632–636.

Rieger, C. and Villez, K. (2012). Resilient control system execution agent (recosea). In *Resilient Control Systems (ISRCS), 2012 5th International Symposium on*, pages 143–148.

Ruddy, J. (2012). Computational acceleration for next generation chemical standoff sensors using fpgas. Master's thesis, Temple University.

Salaheen, F. (2013). Modeling and control of gimbaled laser target system. Master's thesis, Temple University.

Sridhar, S., Hahn, A., and Govindarasu, M. (2012). Cyber attack-resilient control for smart grid. In *Innovative Smart Grid Technologies (ISGT), 2012 IEEE PES*, pages 1–3.

Stengel, R. and Ryan, L. (1991). Stochastic robustness of linear time-invariant control systems. *Automatic Control, IEEE Transactions on*, 36(1):82–87.

Wei, D. and Ji, K. (2010). Resilient industrial control system (rics): Concepts, formulation, metrics, and insights. In *Resilient Control Systems (ISRCS), 2010 3rd International Symposium on*, pages 15–22.

Werbos, P. (1974). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University.

Zhu, J. and Sutton, P. (2003). Fpga implementations of neural networks - a survey of a decade of progress. In *Proceedings of the 13th International Conference on Field Programmable Logic and Applications (FPL 2003*, pages 1062–1066. Springer-Verlag.