ABSTRACT

LEAVITT, ZACHARY DOUGLAS. Neural Network Control of Object in Standing Wave Acoustic Field using Phase Modulation. (Under the direction of Dr. Paul Ro.)

Non-contact manipulation is an important area of development because of its industrial applications in microassembly, its ability to manipulate corrosive or reactive materials, and its use in high temperature liquid processing. There are several methods of non-contact manipulation and acoustic levitation is one of these promising methods. Particles are levitated in an acoustic field by creating a standing wave between two bolt-clamped Langevin transducers. The particles levitate at the pressure nodes in the field which are a stable point of levitation. The levitated particles are manipulated by changing the phase of the motion between the transducers. By changing the phase it is possible to move a pressure node and object from one side of the standing wave to the other.

Control of both the position and velocity of the levitated particle is the goal of this thesis. Three main types of controllers are used to control the 1-d motion of the levitated particle. A PID controller was tested first but performed poorly due to the nonlinearity of the acoustic system. The second type of controller is an internal model controller that uses a neural network as the internal model and the third type of controller is a neural network controller that utilizes error feedback. Based on the root mean square error the internal model controller performed five times better than the PID controller and the neural network controller performed three times better than the PID controller.

Neural Network Control of Object in Standing Wave Acoustic Field using Phase Modulation

by
Zachary Douglas Leavitt

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Mechanical Engineering

Raleigh, North Carolina

2013

APPROVED BY:

_____	_____
Dr. Gregory Buckner	Dr. Chau Tran

_____

Dr Paul Ro
Committee Chair

BIOGRAPHY

Zachary Leavitt was born in Ohio in 1989 but grew up in Western North Carolina. He received a B.S. in Mechanical Engineering from North Carolina State University in 2011and continued on at NC State to work on his M.S. in Mechanical Engineering. Zach's primary interest in engineering is at the junction of controls, computers, and electronics. Specifically his interest is in the development of automated processes and machines. Outside of engineering Zach enjoys being an enthusiastic teacher at Duke TIP and going on adventures. Before starting his engineering career he plans to serve in an AmeriCorps program working in communities in the Northeast United States.

ACKNOWLEDGMENTS

First I would like to thank my advisor Dr. Paul Ro for the support and guidance he has given during the course of this research and during my graduate career. When I was unsure of the next steps to take Dr. Ro was able to help me make decisions.

I would like to thank Joong-Kyoo for all his help. Joong was always willing to sit and explain a principle or help me figure out a problem. Joong seemed to have experience with every piece of software and hardware and was willing to share what he knew.

I would also like to thank Dr Gregory Buckner and Dr. Chau Tran for all the engaging classes they taught, for feedback on this thesis and suggestions for future work.

The word thanks does not begin to cover the gratefulness I have towards my parents and brother but I say thank you all the same.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Motivation of Research

Non-contact manipulation is an important area of development because of its industrial applications in microassembly, its ability to manipulate corrosive or reactive materials, and its use in high temperature liquid processing. Using non-contact handling methods for containerless liquid processing eliminates contamination of the liquid due to a container therefore facilitating undercooling [1]. Microassembly of components becomes increasingly difficult using traditional grippers as the size of components shrink. In the assembly of MEMS, opto-elctronic microsystems and other microcomponents electrostatic, van der Waals, and surface tension forces become significant causing parts to adhere to traditional grippers not allowing proper part placement. There are several approaches to mitigating this problem of adhesion due to surface forces—one approach is non-contact manipulation.

Non-contact manipulation methods eliminate surface forces that make traditional grippers ineffective. An additional advantage of non-contact manipulators is that they are capable of handling fragile and freshly painted parts, or components with delicate surface finishes. This benefit comes from the manipulation pressure being distributed over large portions of the part instead of at the gripping area of a mechanical gripper. Non-contact

handling in the food industry also has the advantage of avoiding end-effector contamination [2].

1.2 System Description and Project Focus

The focus of this project is to control the position and velocity of a small object in a standing wave acoustic levitation system. Travelling waves are produced by two bolt-clamped Langevin transducers facing each other. The two transducers are driven at the same frequency causing a standing pressure wave to be formed between them as shown in Figure 1

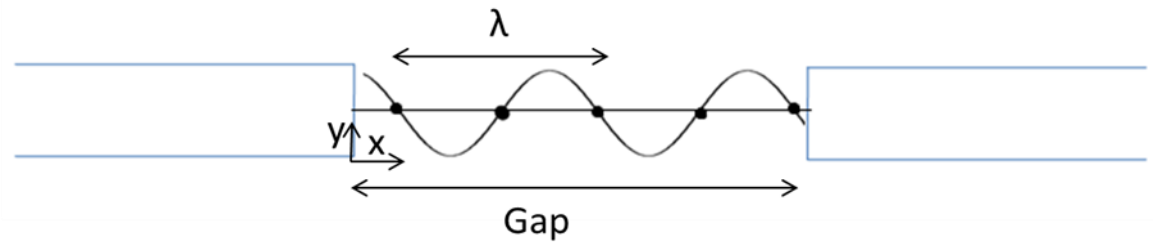


Figure 1 Standing Wave between transducers

The nodes of the standing wave shown in Figure 1 are large black dots and the reference for the node and levitated ball position is from the lower corner of the left actuator. In earth gravity and in air the forces generated by radiation pressure are generally not enough to levitate an object but because the gap between the transducers is close to the Rayleigh distance, near-field acoustic effects increase the pressure on the object. The Rayleigh distance is defined as

$$\mathbf{R} = \frac{\pi \mathbf{r}^2}{\lambda} \quad (1)$$

where r is the transducer radius[3].The nodes of the standing wave and the levitated pellet are moved by changing the phase relation between the two actuators. Several control schemes are compared and evaluated on their performance at various speeds and with the introduction of plant offsets. A particular emphasis is placed on the use of neural networks for plant identification and control. To see a graphical representation of all the node positions a finite element method is used to create a model of the system. The finite element model is compared to the experimental results to determine modeling accuracy.

The organization of the thesis will be as follow:

- Chapter 2 overviews current methods of non-contact handling and their limitations, describes the principles of acoustic levitation, neural networks, and the use of neural networks for control.
- Chapter 3 presents an ANSYS model, discusses data collection and preparation for neural network modeling, and shows how the control schemes are implemented in the experimental system.
- In chapter 4 the experimental setup is described, controllers are evaluated at different speeds and with plant offset, and the experimental results are compared to the modeled results.

Chapter 2

Literature Review and Principles

2.1 Current Methods of Non-Contact Handling

There are several methods of non-contact handling each with its advantages and disadvantages. Magnetics, Electrostatics, Bernoulli's effect, and acoustic forces are several of the physical principles exploited for levitating objects. An example application and a description of the basic principle of each of these methods are outlined below. The focus of this thesis will be on standing wave acoustic levitation but a literature survey of several different methods of levitation is given by Vandaele, Lambert, and Delchambre in *Non-contact handling in microassembly: Acoustical levitation*[2].

Electromagnetic suspension systems utilize the forces between moving charges and magnetic fields characterized by Lorentz's Law. Lorentz's law gives the force on a charged particle as

$$\mathbf{F} = \mathbf{q}(\mathbf{E} + \mathbf{v} \times \mathbf{B}) \tag{2}$$

Where F is the force exerted on the charged particle q by the electric field E and the cross product of the particle's velocity and the magnetic field B. For a current carrying wire equation (2)can be simplified to

$$\mathbf{F} = \mathbf{il} \times \mathbf{B} \tag{3}$$

This formulation is used to calculate the amount of force generated on a wire coil by the suspended object or carrier plate. This force is controlled by changing the amount of current in the coil.

Chen, Tsai, and Fu [4] created a 2-d electromagnetic suspension actuator that used three permanent magnets and three coils to create a levitated platform. Their design is shown in Figure 2.

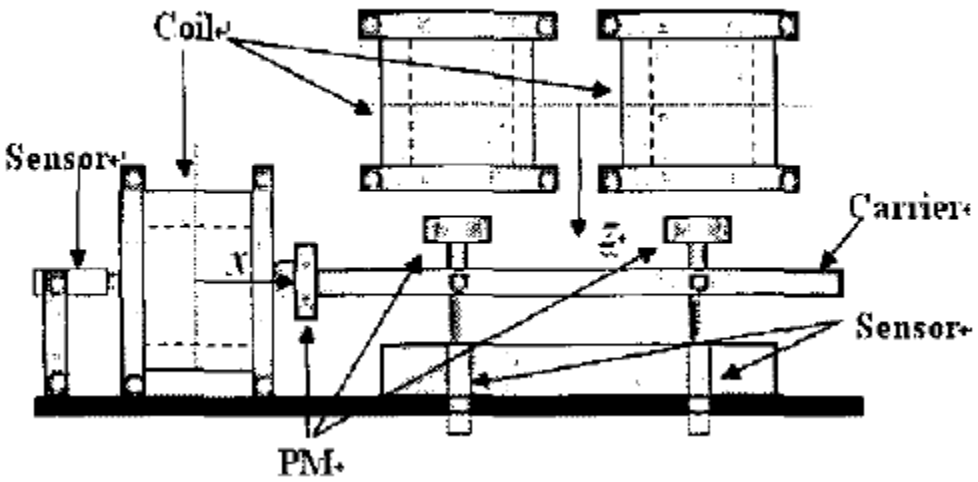


Figure 2 2-d Electromagnetic suspension actuator[4]

Although a reasonable analytical model can be created for this setup, calculations are too extensive for real time control. Instead the force is simply fitted to an experimental relationship between position and current. They demonstrate effective control of their system by using a sliding mode adaptive controller. The sliding mode setup allows for control of the nonlinear system over the range of motion instead of a linearized controller that is limited to

specific regions. The adaptive aspect allows for control to be achieved even when the system is not perfectly known.

Electromagnetic suspension systems work well in applications where a high degree of precision manipulation is necessary. A levitated system is often easier to move precisely because of the low friction that allows for the stick-slip problem to be avoided. Electromagnetic systems also have a high control bandwidth that allows for better system performance. A disadvantage of using a system based on magnetic forces is that the system is inherently unstable and requires active control. Another disadvantage is that a carrier platform must be used unless the object itself is ferromagnetic or a very large magnetic field is generated[4]. Magnetic levitation also is not effective in high temperature applications because demagnetization of magnets at high temperatures.

Bernoulli's lift principle can be applied to create a non-contact manipulation system. Erzincanli, Sharp and Erhal[5] looked at how this type of noncontact system could be applied as a robotic end-effector. The end-effector proposed would grip the lifted object from above using a high velocity air flow. A diagram of the proposed system is shown in Figure 3.

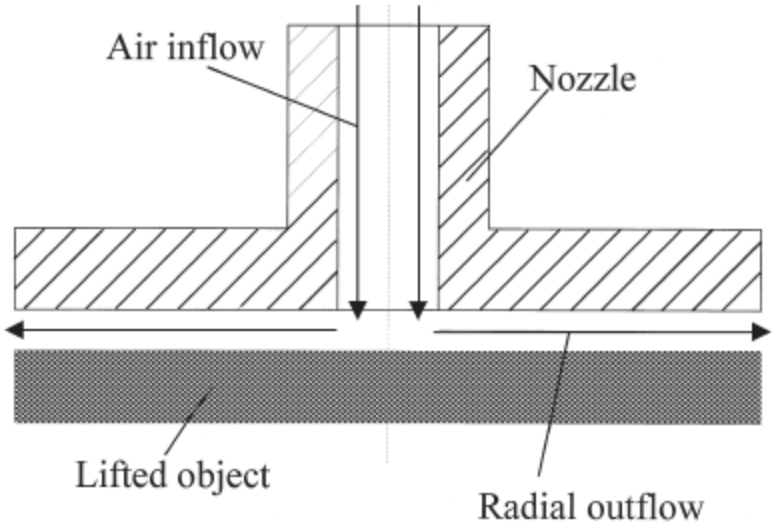


Figure 3 Noncontact handling using Bernoulli's principle [5]

In this system the high velocity flow over the top of the object creates a low pressure zone and generates lift. The object is lifted until the repelling force generated by the outward momentum of the air hitting the object is equal to the Bernoulli lift force. To a degree this system is self-regulating causing a stable holding position. The nozzle proposed in Figure 3 can be constructed of a wide variety of easily manufactured materials. Some common materials might include stainless steel, aluminum, or rigid plastics depending on the hygiene requirement of the application. The end-effector is also light which mean a smaller payload if it is used as a robotic end-effector. Cost of use of a levitation system of this design is also small because compressed air is often readily available in industrial facilities. A limitation of this type of system is that a relatively flat surface is required and the object can't be too compliant because indentations and bending decrease the lifting force[5].

A third method of non-contact manipulation is through electrostatic forces generated between two charges. The force generated between two particles is given by Coulomb's law which is

$$\mathbf{F} = \frac{(\mathbf{q_1 q_2 \hat{r}_{21}})}{\mathbf{4\pi\epsilon_o r_{21}^2}} \tag{4}$$

Where ϵ_o is the permeability of freespace, q is the charge of the two particles and r is the position vector between the particles. Coulomb's law is used to formulate the definition of an electric field as the force generated on a test charge per unit charge at a specific point or

$$\mathbf{E} = \frac{\mathbf{F}}{\mathbf{q}} \tag{5}$$

Using the concept of electric field applied to a distribution of charges instead of a point charge by integrating Coulomb's law over a volume allows the force between two charged objects to be calculated. Equations(4)and (5)show that the direction of the force is always toward the region of higher electric field. Electrostatic non-contact handling uses various methods of manipulating the electric field to control the forces applied to an object.

Biganzoli and Fantoni [6] use electrostatic forces for 2-d manipulation of objects placed on a surface. While they do not levitate the objects they do demonstrate another method of non-contact handling. The setup they use consists of placing objects on a conducting vibration table and then moving an electrode above the surface of the table to change the position of the objects. The vibration table is used to reduce the friction between the objects and the table surface allowing for them to move freely. To keep an electrical discharge from occurring between the table and electrode the table is covered by a glass plate. A diagram of how objects are positioned in their system is shown in Figure 4.

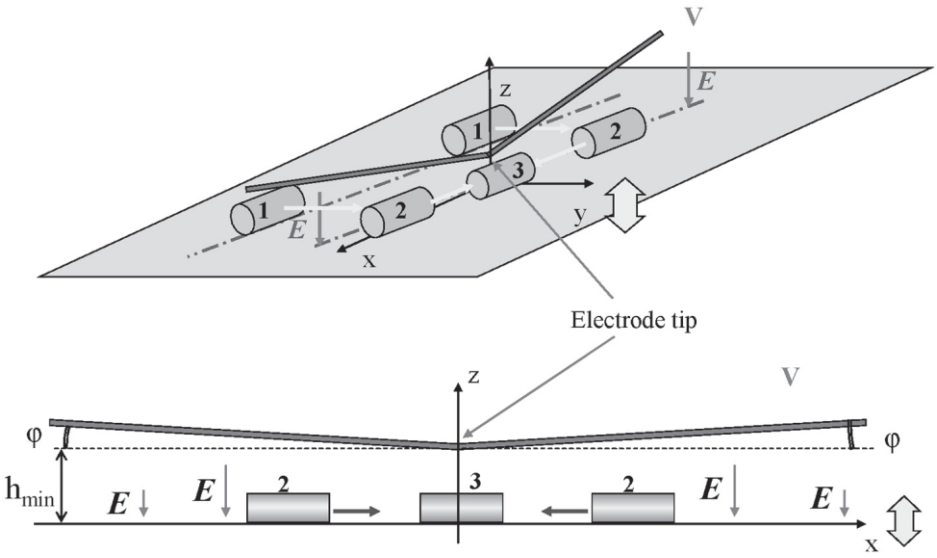


Figure 4 Electrostatic non-contact manipulation system[6]

The electrode is positively charged while the table is grounded causing a buildup of charge between the two just like a capacitor. This buildup of charge causes forces on the cylindrical objects in the direction of the greatest electrical field which is just under the electrode. The objects first line up with the electrode by moving from 1 to 2 and then move to be centered under the tip by of the electrode at 3 because it is closer to the table.

In the setup that was used there were several problems. The first is that the induced charges caused an increased adhesion force between the table and the object. Another potential problem with electrostatic handling is that dust is also attracted to the static charge, which may cause problem for sensitive objects. A third problem is that small irregularities in the object shape or surface cause incorrect object motion[6]. Electrostatic methods can be used on both conductive and dielectric materials which is an advantage over electromagnetic

manipulation systems. Electrostatic methods can be applied to create a levitated system but because they use an attractive force they require active control to maintain stability[7].

2.2 Standing Wave Acoustic Levitation

Standing wave acoustic levitation (SWAL) systems often uses an acoustic transducer and a reflector to create a standing wave between the reflector and transducer. Another method of creating a standing wave is to use two transducers driven at the same frequency.

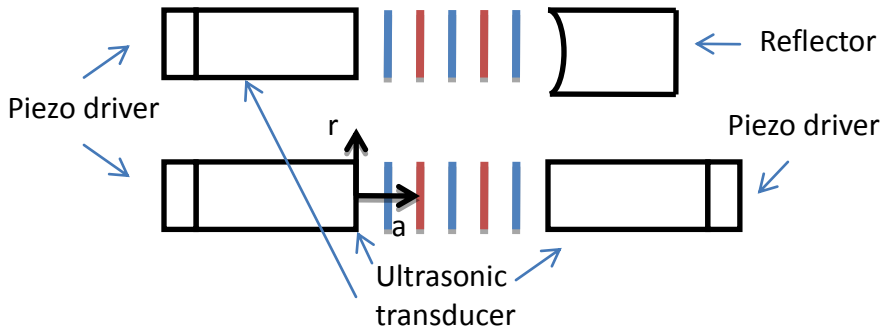


Figure 5 Acoustic transducer with reflector (top) Two acoustic transducers (bottom)

Figure 5 shows a diagram of a SWAL system that utilizes a reflector and one that utilizes two drivers. The lines between the transducers and reflector show the high and low pressure points of the standing wave. In this diagram the waves are shown as perfect plane waves but this is not the case for real systems. The area between each line is a pressure node where an object can be levitated. The numbers of nodes that form depend on the speed of sound and driving frequency. In Figure 5, **a** is along the axial direction and **r** is the radial direction.

SWAL systems have several advantages over other methods of non-contact handling. The ability to levitate an object is independent of the whether the material is conductive, or magnetic therefore a large range of material can be levitated. The material limitation imposed by standing wave systems is that the object density must be below a critical value determined by the system parameters. Pressure nodes occur along the axial direction of a standing wave and it is these nodes that create a stable point to levitate objects. These nodes do not need to be controlled to remain stable unlike many magnetic and electrostatic systems. The third advantage of a SWAL system is that the object shape is not limited to flat or spherical objects, any shape object can be levitated as long as it is smaller than half the wavelength of the standing wave. A characteristic of a SWAL system is fluid flow around the levitated object which can cause heat and mass transfer. This heat and mass transfer can be beneficial in experiments where part cooling is desired but may not be desired in high temperature experiments. Fluid flow around the part also means that while the part is stable it does rotate.

The force that levitates an object is due in part to the acoustic radiation forces on the object. Early work on acoustic radiation pressure was done by King who developed the equations describing the pressure on small, incompressible spherical objects. Small defined by King is the radius of the object being much smaller than the wavelength of the standing wave. The wavelength of the standing wave is given by

$$\boldsymbol{\lambda_s = C_s/f} \tag{6}$$

where C_s is the speed of sound in the medium and f is the driving frequency.

King's analysis was for both traveling and standing waves but his result shows that the pressure of standing waves is orders of magnitude greater than traveling waves. The

expression developed by King for the average pressure on a small sphere from traveling waves is

$$\mathbf{P_{avg} = 2\pi\rho_o|A|^2\left(\frac{2\pi r}{\lambda}\right)^6 * \frac{1+\frac{2}{9}\left(1-\frac{\rho_o}{\rho_1}\right)^2}{\left(2+\frac{\rho_o}{\rho_1}\right)^2}} \quad (7)$$

The pressure from traveling waves is a function of the density of the surrounding median,ρ_o, the density of the object,ρ_1, the wavelength λ, and A is assumed to be the wave amplitude. The pressure on a rigid sphere from a standing wave is given by the function

$$\mathbf{P_{avg} = \pi\rho_o|A|^2 \sin\left(\frac{2\pi h}{\lambda_s}\right)\left(\frac{2\pi r}{\lambda_s}\right)^3 * \frac{1+\frac{2}{3}\left(1-\frac{\rho_o}{\rho_1}\right)^2}{\left(2+\frac{\rho_o}{\rho_1}\right)^2}} \quad (8)$$

The pressure from a standing wave has a similar form as the traveling wave but there are several important differences to note. The first difference is that the standing wave pressure as expected is dependent on the position in the wave. This is given by the $\mathbf{sin}(\frac{2\pi h}{\lambda_s})$ term, where **h** is the distance from a pressure node. The reason standing waves generate more force than traveling waves is because of the $\frac{2\pi r}{\lambda_s}$ term. The circumference of the object must be much less than the wave length in order for levitation to occur or mathematically

$$\frac{2\pi r}{\lambda_s} << 1 \quad (9)$$

In the traveling wave this term is raised to the 6th and in standing waves it is only raised to the 3rd meaning the magnitude of pressure from the standing wave is greater [8]. The analysis done by King makes several assumptions including incompressible small spherical objects, perfect plane waves, and a non-viscous medium. In 1955 Yoshikawa and Kawashima

extended King's work to compressible spheres[9]. For objects with $\frac{r}{\lambda} > .08$ King's model looses accuracy and must be modified as suggested in [10] with this modification the model is accurate as long as $\frac{r}{\lambda} < .12$.

The analysis done by King and others describes the forces along the axial direction between the actuators but there is another force that acts in the radial direction. The force in the radial direction is towards the actuator centerline at the pressure nodes. There are several phenomena that lead to the development of this force. In the idealized case of plane waves there would be no distribution along the radial direction but realistically perfect plane waves are not created causing a non-uniform pressure distribution. Another source of this radial force is from acoustic streaming caused by the non-uniform pressure distribution along the radial direction of a node [11]. Tuziuti et al. relate this idea of a non-uniform pressure distribution along the radial direction to there being a non-uniform velocity distribution. Using this velocity distribution and an equation that relates radiation force to potential and kinetic energy the force is quantified in the radial direction [12]. The radiation force equation used was first developed by Nyborg and is

$$\boldsymbol{F} = \frac{4\pi r^3}{3}[\boldsymbol{B}\nabla\langle \boldsymbol{K_a}\rangle - \nabla\langle \boldsymbol{P_a}\rangle] \tag{10}$$

where$\langle K_a\rangle$ and $\langle P_a\rangle$ are the time averaged kinetic and potential energy of sound respectively. The **B** in equation (10) **1010**is defined by the density of the medium and object and is

$$\boldsymbol{B} = \frac{3(\rho_1 - \rho_o)}{2\rho_1 + \rho_o} \tag{11}$$

where $\rho_o$is the density of the object and ρ_1 is the density of the medium[13]. Tuziuti et al. [12] proposed and validated experimentally that this velocity distribution in the radial

direction is a Gaussian distribution, g(y). If the distribution of velocity is Gaussian then the $\langle K_a \rangle$ and $\langle P_a \rangle$ terms become

$$\langle \boldsymbol{K_a} \rangle = \boldsymbol{g(y)^2} \left(\frac{\boldsymbol{A^2}}{\boldsymbol{\rho_o c^2}} \right) \tag{12}$$

$$\langle \boldsymbol{P_a} \rangle = \boldsymbol{0} \tag{13}$$

$$\boldsymbol{g(y)} = \mathbf{exp}(-\frac{\boldsymbol{y^2}}{\boldsymbol{2\sigma^2}}) \tag{14}$$

Equations 10-(14)are combined to form the equation for the radiation force in the radial direction which is

$$\boldsymbol{F_y} = -\left(\frac{\boldsymbol{2By}}{\boldsymbol{\sigma^2}} \right) \left(\frac{\boldsymbol{4\pi r^3}}{\boldsymbol{3}} \right) \left(\frac{\boldsymbol{A^2}}{\boldsymbol{\rho_o c^2}} \right) \mathbf{exp}(-\frac{\boldsymbol{y^2}}{\boldsymbol{\sigma^2}}) \tag{15}$$

A in equation(15) is the sound pressure amplitude and σ is the standard deviation of the Gaussian velocity distribution[12]. A plot of this force over a range of $\frac{y^2}{\sigma^2}$ is shown in Figure 6.

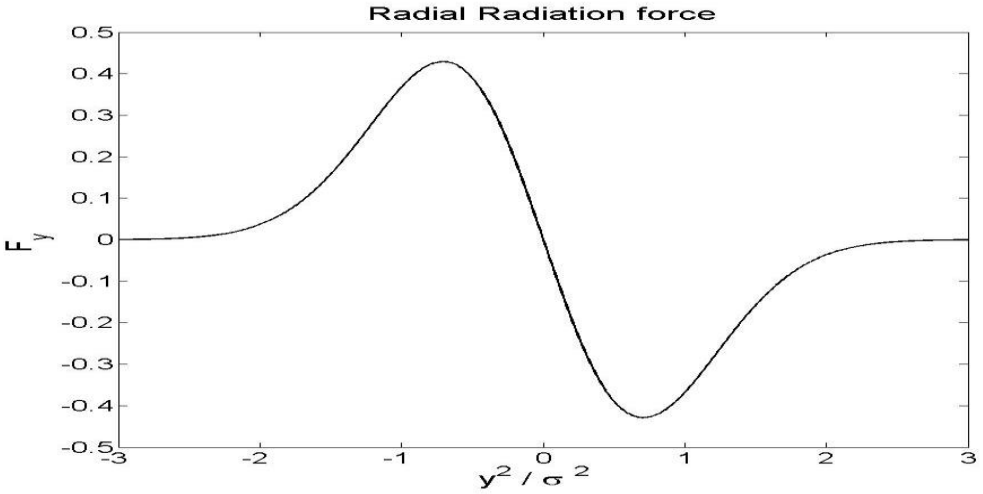


Figure 6 Radiation force distribution in the radial direction

The force in the radial direction causes an object to be pushed toward the centerline of the cylindrical actuators. There are two equilibrium points for the object to levitate at but the centerline is the most stable point. This radial centering force combined with the stable axial position makes standing wave acoustic levitation an attractive levitation technique.

2.3 Neural Networks

Neural networks (NN) or Artificial neural networks (ANN) as they are often called are a tool that is based on the biology of the nervous system. A biological neuron is composed of four main parts: dendrite, synapse, soma, and axon. A diagram of a biological neuron is shown in Figure 7.

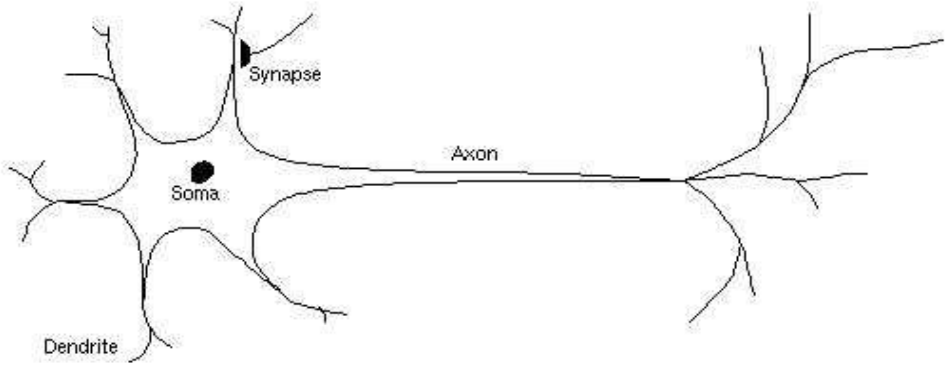


Figure 7 Biological Neuron

The purpose of a neuron is to receive, process and transmit signals. A neuron receives signals from connected neurons through the dendrites. The dendrites then transfer all the collected signals to the soma which processes incoming signals and transmits a signal to the axon. The

axon is the device by which the neuron passes its signal to other neurons. The connection of many of these neurons creates a powerful network that is capable of processing complex information and acting on it[14].

Artificial Neurons have a similar topology as their biological namesake. A model of an artificial neuron is shown in Figure 8.

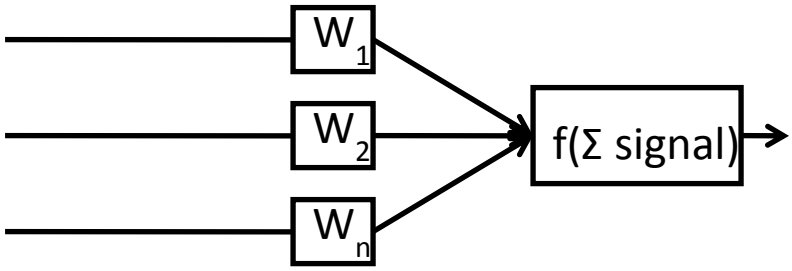


Figure 8 Artificial Neuron

The artificial network collects signals and multiplies each signal by a distinct value called a weight therefore assigning certain inputs more weight than others. The signals are then summed and processed just as in the soma of the biological neuron. The output of the neuron shown in Figure 8 does not branch but the output of one neuron could also be the input to multiple other neurons. A single neuron is limited in its ability to process signals and it is only through the connection of several neurons forming a network that a powerful tool is developed [15].

Neural networks can be applied to a broad range of applications including robotics, image processing and market predictions. These three applications relate to three types of

problems that NNs are good at solving. Control of robotic systems is difficult using conventional controllers because of the nonlinearity and changing system parameters. The inverse kinematics of a robotics system is a particular difficulty in robotic controls. The inverse kinematics problem in robotics is given a desired end-effector position find the joint variable that lead to this end-effector position. The difficulty is that there may be multiple solutions, a single solution or no solution at all to reach the desired end-effector position. Traditionally there have been several methods of solving the inverse kinematics problem including algebraically, geometrically, and using an iterative method. Each of these methods has drawbacks and limitations, and a better method of solving for the joint variables is desired. One solution is to formulate the inverse kinematics problem as an optimization problem which is well suited to being solved in real time by a neural network [16]. This example illustrates one way in which NNs can be applied to controls problems.

Image processing is a broad area of study requiring the manipulation of large amounts of information to extract information, optimize information transfer, or modify information. Neural Networks can aide in all of these processes but one particular application is in feature extraction from an image. The goal of feature extraction is to reduce an image to a set of particularly useful parts. Feature extraction leads to a less complex image that has a lower computational cost of further processing but still contains the desired information. This process is often used in conjunction with object recognition, image matching, and segmentation [17]. Using NNs to accomplish feature extraction tasks demonstrates their ability to recognize patterns in complex systems.

Forecasting means linking the interdependence of variables in order to predict other variables. Financial markets are influenced by tangible assets such as buildings and land but they are also influenced by the harder to model in-tangible assets such as patents, copyrights, and brand recognition. Finding how these variables are linked allows for one to build a model and predict future outcomes. One method that is being used to find the relationships among these many variables utilizes the powerful function approximation capabilities of NNs to develop models and make predictions. NN models have the advantage of not being fixed around a certain formula, which gives them the ability to adapt to changes in the market [18].

The first mathematical model of a neuron was developed by Warren McCullouch and Walter Pitts in their 1943 paper. The McCulloch-Pitts model is governed by a set of five rules:

1. The activity of a neuron is an all or nothing process
2. The activity required for a neuron to fire is independent of previous activity
3. The only delay within the system in the transfer of the signal
4. A neuron's firing threshold will increase if an inhibitor signal is active
5. The structure of the network does not change with time

The mathematical realization of these rules is

$$\mathbf{u} = \sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{w_i x_i} - \mathbf{l} \tag{16}$$

where w is a fixed vector of weights, x is the set of input signals, and l is the required activation energy. The output of the neuron is given by

$$\mathbf{y} = \mathbf{f}(\mathbf{u}) \tag{17}$$

where the function is the step function shown in Figure 9 [19].

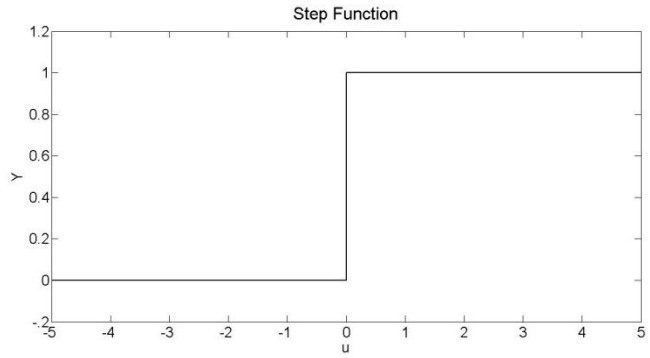


Figure 9 Step Function

The function shown in Figure 9is often called the activation function of the neuron. This activation function is what decides if the input signals are going to be passed on or in other words if the neuron is going to fire. There are many possible activation functions, and some of the more common functions include the sigmoid function given by

$$\mathbf{f(u)} = \frac{\mathbf{1}}{\mathbf{1+e^{-u}}} \tag{18}$$

and shown in Figure 10. The sigmoid function is popular because it is differentiable which is useful when trying to determine the weights.

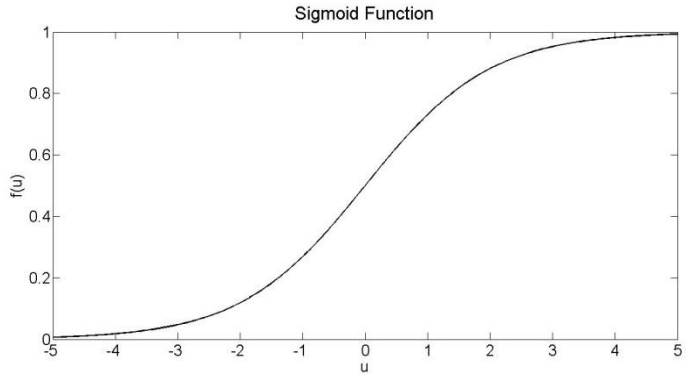


Figure 10 Sigmoid Function

Another type of activation function a piecewise linear function such as

$$f(u) = f(x) = \begin{cases} -1, x < -1 \\ x, \ -1 \le x \le 1 \\ 1, \qquad x > 1 \end{cases} \tag{19}$$

From the McClloch-Pitts model came the Perceptron model developed by Frank Rosenblatt in 1958. The Perceptron has a similar structure as the neuron shown in Figure 8 and similar equation as the McCllouch-Pitts model. The biggest difference is that in the Perceptron the weights are tuned parameters. The Perceptron is tuned by using a sample set of data that includes inputs and desired outputs. The difference between the desired output and the actual output is used to change the value of the weights. The weight update rule used in Rosenblatt's Perceptron is

$$\mathbf{w_i(k+1) = w_i(k) + \mu(d - y)x_i} \tag{20}$$

Where $w_i(k)$ is the current weight of the ith input, μ is the learning rate and d is the desired output [20].

The structure of the McClloch-Pitts model and the learning ability of the Perceptron form the building blocks of many types of neural networks. These networks of neurons can be connected in a vast array of ways to form many different network architectures. When neurons are connected in a single layer with no feedback from the neuron outputs to the inputs the network is called a single layer feedforward network. An example of this kind of network is shown in Figure 11.

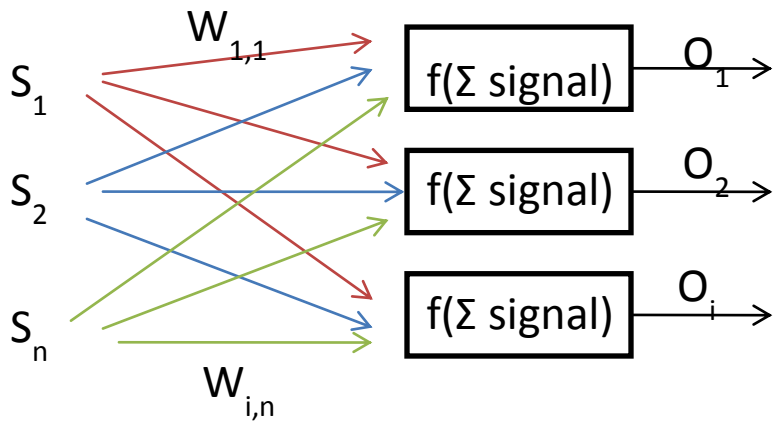


Figure 11 Single layer feedforward neural network

For the single layer network S_n is the nth input and $W_{i,n}$ is the weight of the nth input going to the ith neuron. The number of outputs O is the same as the number of neurons in the network. This type of diagram becomes cumbersome when describing more complex networks therefore an abbreviated diagram that utilizes vectors and matrices is commonly used.

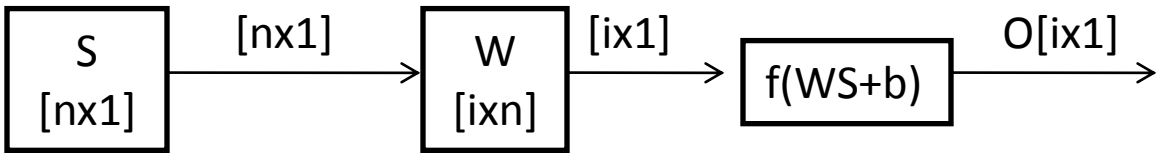


Figure 12 Abbreviated single layer diagram

For the abbreviated diagram the input becomes a column vector that is multiplied by a weight matrix. The **b** column vector is called the bias term which is a vector of constants that shifts the activation function for each neuron.

Adding additional layers is one way of increasing network capabilities. When three or more layers are used and there is no feedback from previous layers the network is called a multi-Layer feedforward network or multi-layer perceptrons (MLPs). These multilayer perceptrons have become the most widely used architecture particularly in the systems and controls field[21].Figure 13shows a MLP with a single hidden layer.

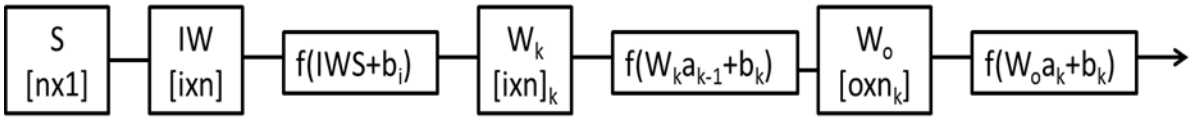


Figure 13 Multilayer Perceptron

The multilayer perceptron is usually broken into three parts. The first part is the input layer which consists of a weight matrix, IW, and an activation function. The last part of the network is called the output layer and it consists of the weight matrix, W_o, and the output activation function. The output layer contains as many neurons as desired outputs, also the output activation function is often just a linear scaling factor. Between the input and output layer is the hidden layer(s) which can be multiple layers and consists of a weight matrix and activation function for each layer. While more layers are sometimes used a single hidden layer is the most common configuration because adding layers dramatically increases the

training time and a MLP with a single hidden layer can approximate any function arbitrarily well with enough perceptrons [22].

A single perceptron can effectively be trained by minimizing the error between the input and output using equation(18). Training multi-layer networks with several neurons and multiple layers becomes a much more difficult task than the training of a single perceptron. To accomplish this task a back-propagation algorithm can be utilized. The back-propagation algorithm looks at the last layer of the network first and for each output finds the error between the output and desired output and calculates how changing the weights of the input impacts the output. The algorithm then changes the weights in the opposite direction as the calculated gradient, thus causing the overall system error to decrease. For the hidden layers a similar process is used that says the error of a neuron is the sum of the error of the neurons connected to its outputs times their weights. This hidden layer error calculation is used to come up with an error gradient and the weights are changed in the opposite of the maximum gradient. By continuing this process over the entire set of training data the MLP network begins to model the unknown input output function.

A variant of back-propagation that speeds up convergence is the Marqurdt-Levenberg algorithm. This algorithm combines back-propagation with a quasi-Newton approach. Although a quasi-Newton approach is a more effective method, using just that approach leads to computational problems for large networks. The Marqurdt-Levenberg algorithm acts like the computationally efficient back-propagation method when the error is large but as the error decreases the algorithm acts like the more effective but computationally expensive quasi-Newton method. What this means is that the algorithm is able to quickly zoom in on

minima and then slowly converge to the precise minimum. A thorough examination of the back-propagation method and the quasi-Newton method can be found in *Training Feedforward Networks with the Marquardt Algorithm* written by Hagan and Menhaj[23].

2.4 Neural Networks for Control

Neural networks can be applied to the field of controls in a variety of ways. Networks can be used for modeling unknown nonlinear systems, feedback controllers, inverse plants, predictive controllers or as adaptive controllers. This thesis will focus on the use of neural networks for plant identification, using the inverse plant in conjunction with a PID control scheme, an internal model control scheme, and a neural network feedback controller. All of these controllers can be trained offline and do not utilize online adaptive techniques.

The simplest type of controller that can be implemented using a neural network approach is an inverse plant controller. A diagram of a system using an inverse plant as a controller is shown in Figure 14.

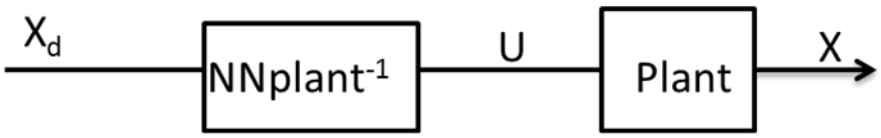


Figure 14 Inverse Plant Diagram

This controller relies on being able to model the inverse dynamics of the plant using input output pairs collected offline. The input vector X_d is the set-point of the plant and for many systems will contain previous set-points of the system. While this is a simple controller to implement one flaw is that there is no feedback and changes in the plant over time and disturbances will reduce the effectiveness of the controller. To increase disturbance rejection and tracking abilities the controller needs to be formulated in such a way that feedback can be introduced. An inverse plant controller with feedback is shown in Figure 15

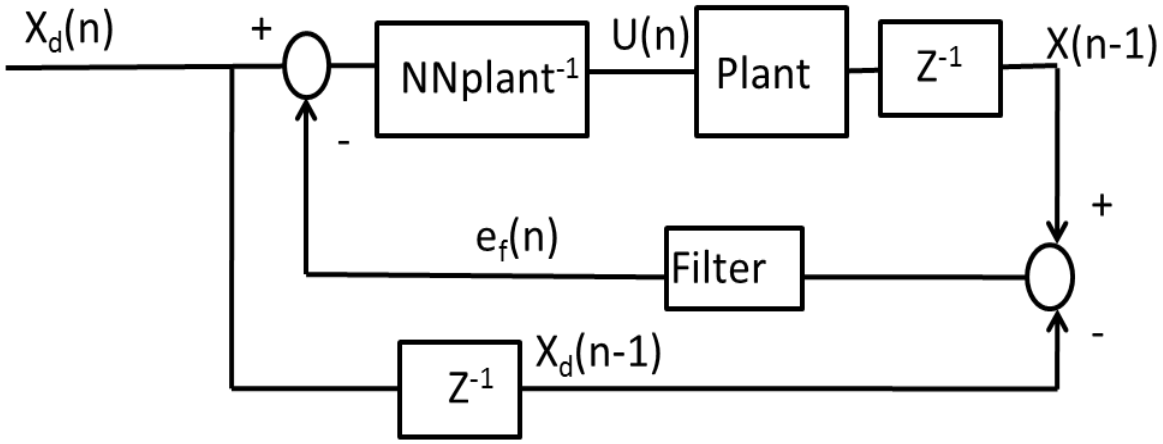


Figure 15 Inverse Plant with Feedback

The control input to the plant at time n can be formulated as

$$\boldsymbol{U(n) = Plant_{nn}^{-1}(X_d(n) - e_f(n))} \tag{21}$$

where the $Plant_{nn}^{-1}(.)$ function is trained from input output data of the plant and the filtered error is given by

$$\boldsymbol{e_f(n) = F(X(n-1) - X_d(n-1))} \tag{22}$$

where the filter F(.) applied to the error is a first order filter of the discrete form

$$\boldsymbol{e_f(n) = \alpha * (X_d(n-1) - X(n-1)) + (1-\alpha) * e_f(n-1)} \quad (23)$$

α is the tuning parameter that changes the amount of filtering applied where $\alpha = 1$ is not filtering and $\alpha = 0$ is a no-pass filter. The Z^{-1} blocks are discrete unit time delays. This control scheme allows for compensation to be made for the differences between the plant and inverse plant or disturbances.

In the previous control scheme the error is only dependent on the set point and output of the plant but another formulation would make the error dependent on previous error values, the plant output and the set-point.

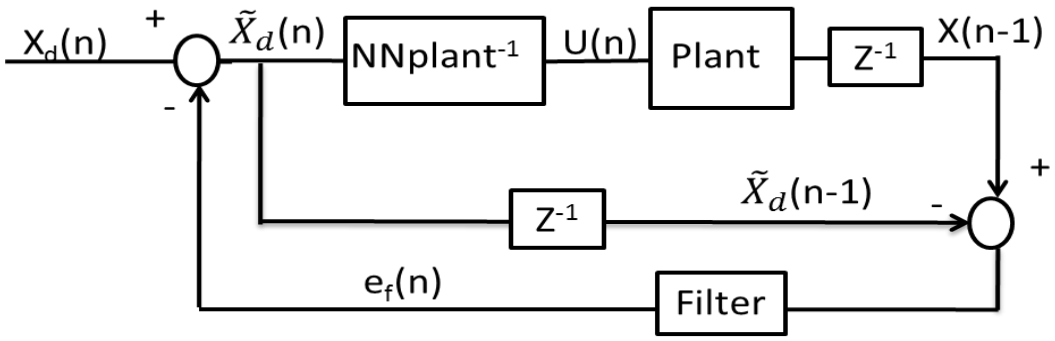


Figure 16 Internal Model Controller

Figure 16 shows an internal model controller (IMC) that compares the output of the plant to a set-point modified by the previous error. The plant input is governed by equation (21) where the error is given by

$$\boldsymbol{e_f(n) = F(X(n-1) - \tilde{X}_d(n-1))} \quad (24)$$

where $\tilde{X}_d$ (n) is the modified set point that accounts for modeling error and disturbances and is given by

$$\widetilde{\boldsymbol{X}}_{\boldsymbol{d}}(\boldsymbol{n}) = \boldsymbol{X}_{\boldsymbol{d}}(\boldsymbol{n}) - \boldsymbol{e}_{\boldsymbol{f}}(\boldsymbol{n}) \tag{25}$$

Three properties that make IMC a good choice of a control scheme are

1. If the controller and plant are stable and the internal model perfectly represents the plant, then the closed loop system will be stable.
2. If an inverse plant model exists and is used as the controller, and the system is closed loop stable, then with no disturbances perfect control is achieved.
3. If the controller gains are inverse of the internal model gains then offset-free control is achieved for constant set points [24].

While a perfect model can never be achieved accurate models can be developed using neural networks to model the inverse properties of a system. One advantage of the IMC controller over the feedback controller shown in Figure 15 is zero steady state offset is achieved.

One way of combining the power of neural network mapping with a more traditional controls approach is by using an inverse plant model in conjunction with a PID controller applied to the error. This configuration is similar to the scheme shown in Figure 15 except the error goes through a PID controller.

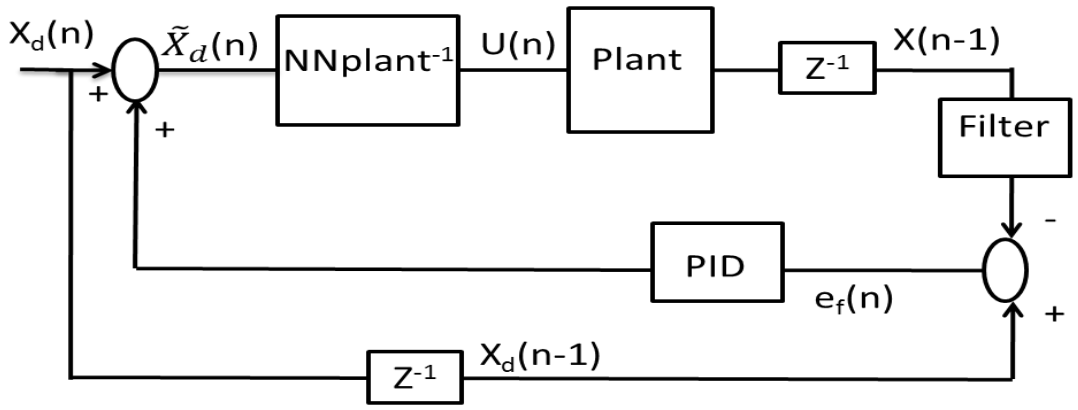


Figure 17 Inverse plant with PID control

The input into the inverse plant is given by

$$\widetilde{\boldsymbol{X}}_{\boldsymbol{d}}(\mathbf{n}) = \mathbf{X_d}(\mathbf{n}) + [\mathbf{k_p e_f}(\mathbf{n}) + \mathbf{k_i}\sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{e_f}(\mathbf{i})\mathbf{t_s} + \mathbf{k_d}\frac{\mathbf{e_f(n)-e_f(n-1)}}{\mathbf{t_s}}] \quad (26)$$

where the error is given by

$$\boldsymbol{e_f(n)} = \boldsymbol{X_d(n-1)} - \boldsymbol{F(X(n-1))} \quad (27)$$

An advantage of this controller over a simple feedback system is that because it incorporates integral control steady-state offset can be eliminated. Also, this controller has the capability to respond faster to error than a simple feedback controller. A disadvantage of trying to use a linear control technique on a nonlinear system is that finding gains that work over the entire system space is impossible for many systems.

A PID controller applied to error feedback may not be able to respond well over the input space but a NN feedback controller can be developed that would respond better over the entire region given the proper inputs. The schematic of this NN controller is shown in Figure 18. The difference between this controller and the inverse plant controller with

feedback is this includes a neural network that uses the set-point and error to find the input into the inverse plan. The nonlinear properties of neural networks allow the control action to depend on the region and not just the error.

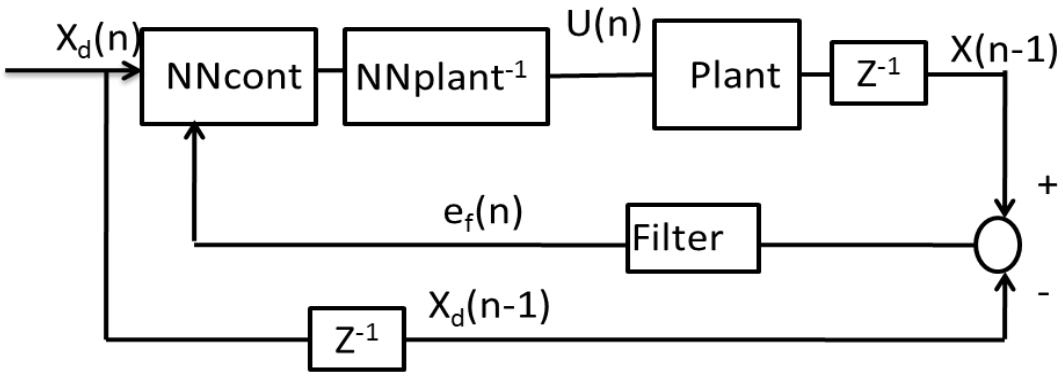


Figure 18 Feedback Neural Network controller

This multi input controller is trained from the input and outputs of the inverse plant controller. Figure 18 shows the NNcont and $NNplant^{-1}$ as separate blocks but in reality it is possible to combine them into a single larger neural network.

To manipulate any levitated object some form of control is necessary. With magnetic and electrostatic levitation active control is necessary just to create a stable levitation position. The principles of acoustics show that standing wave acoustic levitation creates a stable levitation position at the pressure nodes. The objects that can be levitated are not limited to a certain shape or material type but are limited by size and density. Although there is a stable levitation position manipulating this position is a nonlinear process that needs to be modeled and controlled. With their nonlinear capabilities neural networks are a good way

of modeling the levitation system from experimental data. The models created can be applied to form a control system capable of handling the acoustic nonlinearity. Neural networks are a possible way to model the system from experimental data but a model based on acoustic principles is also useful in understanding this system. To model and control this complicated system both a neural network and finite element model are created. The development and implementation of these two models is discussed in chapter 3.

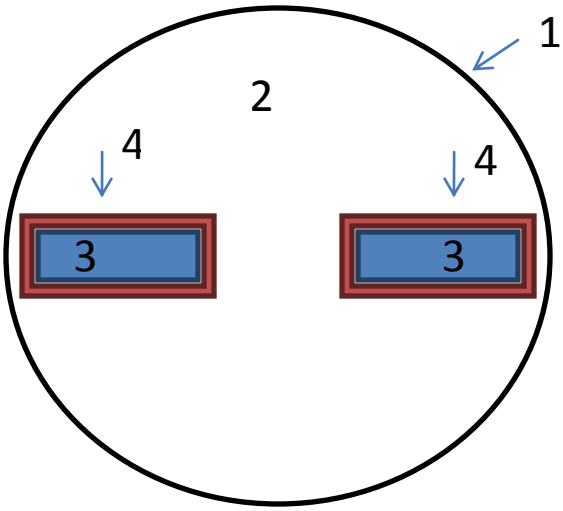
Chapter 3

Finite Element and Neural Network Modeling

3.1 ANSYS Model

Analytically modeling an acoustic system for a non-idealized case is not a trivial task. Since the pistons used in this experiment are not baffled they do not create perfect plane waves as in the idealized case. Also it is desired to know the pressure distribution along the radial direction not just at the centerline. A finite element model is created to view the pressure distribution and node locations between the actuators at different phases. The first step to creating a finite element model is to define the geometry of the system. After the geometry is defined the material properties, element types, system loadings, and analysis mode are selected. The final step is to create a mesh for the system and solve for the desired outputs.

The model for the acoustic system will be a 2-d model of the cross-section of the cylinders. The geometry for this system will simply consist of a circular air region, and two aluminum cylinders. Each region is defined by a certain element type and physical properties. There are three acoustic element types and one structural type element used in this model. Figure 19 shows the geometry and region to which each element is applied.



1	FLUID129 (1-d acoustic element)
2	FLUID29 (2-d acoustic element)
3	PLANE182 (2-d structural element)
4	FLUID29 (2-d acoustic interface elements)

Figure 19 ANSYS model and element types

The gap between the cylinders is 3 cm and the diameter of them is 2cm. The circular region is the air region which has a radius of 9cm. The first element that is used is the 1-d acoustic element that surrounds the air region—this element is an absorbing boundary condition. An absorbing boundary condition is used to simulate a large open space that does not reflect sound waves back toward the actuators [25]. The second element that is used defines the air region and is the 2-d acoustic element called FLUID29. The 2-d acoustic element is based on the wave equation

$$\frac{1}{c^2} * \frac{\partial^2 P}{\partial t^2} - \nabla^2 P = 0 \tag{28}$$

where C is the speed of sound in the fluid, P is the acoustic pressure and is a function of position and time. A discrete form of this equation can be written as

$$\left[\boldsymbol{M_e^{\mathrm{p}}}\right]\{\ddot{\boldsymbol{P}}_{\boldsymbol{e}}\} + [\boldsymbol{K_e^P}]\{\boldsymbol{P}_{\boldsymbol{e}}\} = \mathbf{0} \tag{29}$$

The matrix M_e^{p} is the fluid mass matrix and is calculated using the pressure shape function {N} and the equation

$$[\boldsymbol{M_e^{\mathrm{p}}}] = \frac{\mathbf{1}}{\boldsymbol{C^2}} \int_{\boldsymbol{vol}} \{\boldsymbol{N}\}\{\boldsymbol{N}\}^{\boldsymbol{T}} \boldsymbol{d}(\boldsymbol{vol}) \tag{30}$$

The element stiffness matrix is calculated by the equation

$$[\boldsymbol{K_e^P}] = \int_{\boldsymbol{vol}} [\boldsymbol{B}]^{\boldsymbol{T}} [\boldsymbol{B}] \boldsymbol{dvol} \tag{31}$$

Where the matrix [B] is the dot product of the gradient and shape function and is given by

$$[\boldsymbol{B}] = \boldsymbol{\nabla} \cdot \{\boldsymbol{N}\}^{\boldsymbol{T}} \tag{32}$$

Equation (29) describes the 2-d acoustic elements that are only in contact with other acoustic elements but region 4 is in contact with structural elements as well. A modification is made to the standard acoustic element that couples it to the actuators allowing for fluid structure interaction. The coupling term at the fluid structure interface is that the pressure gradient of the fluid is equal to the density of the fluid dotted with the acceleration of the structure or

$$\{\boldsymbol{n}\} \cdot \{\boldsymbol{\nabla P}\} = -\boldsymbol{\rho_o} \cdot \frac{\boldsymbol{\partial^2}}{\boldsymbol{\partial t^2}} \{\boldsymbol{u}\} \tag{33}$$

where {n} and {u} are the normal vectors of the fluid and structural surface respectively. In discrete form this coupling term becomes

$$\boldsymbol{F_c} = \boldsymbol{\rho_o}[\boldsymbol{R_e}]\{\ddot{\boldsymbol{u}}\} = \boldsymbol{\rho_o} \int_{\boldsymbol{s}} \{\boldsymbol{N}\}\{\boldsymbol{n}\}^{\boldsymbol{T}}\{\boldsymbol{N'}\}^{\boldsymbol{T}} \boldsymbol{d}(\boldsymbol{s})\{\ddot{\boldsymbol{u}}\} \tag{34}$$

This term is added to the left side of the discrete wave equation to create the acoustic elements with fluid interaction which are governed by

$$\left[\boldsymbol{M_e^p}\right]\{\ddot{\boldsymbol{P}}_e\} + [\boldsymbol{K_e^P}]\{\boldsymbol{P_e}\} + \boldsymbol{\rho_o}[\boldsymbol{R_e}]\{\ddot{\boldsymbol{u}}\} = \boldsymbol{0} \quad \textbf{(}35\textbf{)}$$

Region 3 uses a simple 2-d structural element defined by Young's Modulus and Poisson's ratio to model the acoustic actuators. The structural element also has a coupling term that links its governing equation to the pressure gradients at the fluid structure interaction. The goal of this thesis is the investigation of the standing acoustic wave not the structural vibration therefore a detailed analysis of the structural equation is omitted. For a more detailed explanation of the acoustic and structural elements see the theory reference section of the ANSYS help documentation.

To completely define the model not only must the element types be created but the material properties and loadings must be set as well. Table 1 gives the properties that are used to define the model shown in Figure 19.

Table 1 Physical properties

Aluminum	Air	Loadings
E=60e9 Pa	Speed of sound=340 m/s	Frequency=26.7 kHz
Poisson's ratio =.35	Density=1.2 kg/m^3	Amplitude= 7um
	Mu=0	

The actuators are made out of aluminum and are described by the aluminum section of the property table. After the model is defined each section is meshed individually. Each section is meshed individually because the fluid structure interface and area between need a fine mesh while most of the air region does not need a fine mesh. The results of this meshing process are shown in Figure 20.

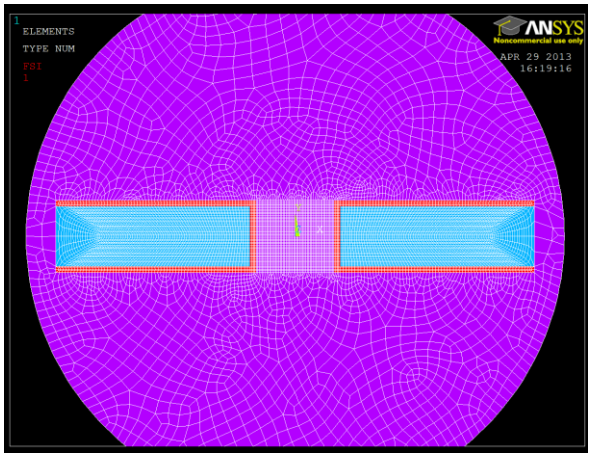


Figure 20 Meshed acoustic system model

Once the model is meshed a harmonic loading is placed on the inside faces of the actuators while the outside faces are fixed in space. The load is defined such that the inside faces move toward each other at the same speed when the phase and time are 0. The phase between the actuators is changed by modifying the load on the right actuator face. The harmonic load is written as a complex number where the real and imaginary terms define both the magnitude and phase of the load. For this analysis the magnitude is kept constant but the phase is increased from 0 to 330 degrees on the right face. The positions of the left and right actuators are given by the equations

$$\boldsymbol{U_L = 7 * 10^{-6}\sin(\omega t)}$$

$$\boldsymbol{U_R = -7 * 10^{-6}\sin(\omega t + \phi)} \quad (36)$$

where U_L is the left actuator, U_R is the right actuator and ϕ is the phase between them.

With the load applied the solution is calculated and analyzed. The pressure distribution of the system with the applied load given in Table 1 and $\phi = 0$ is shown in Figure 21.

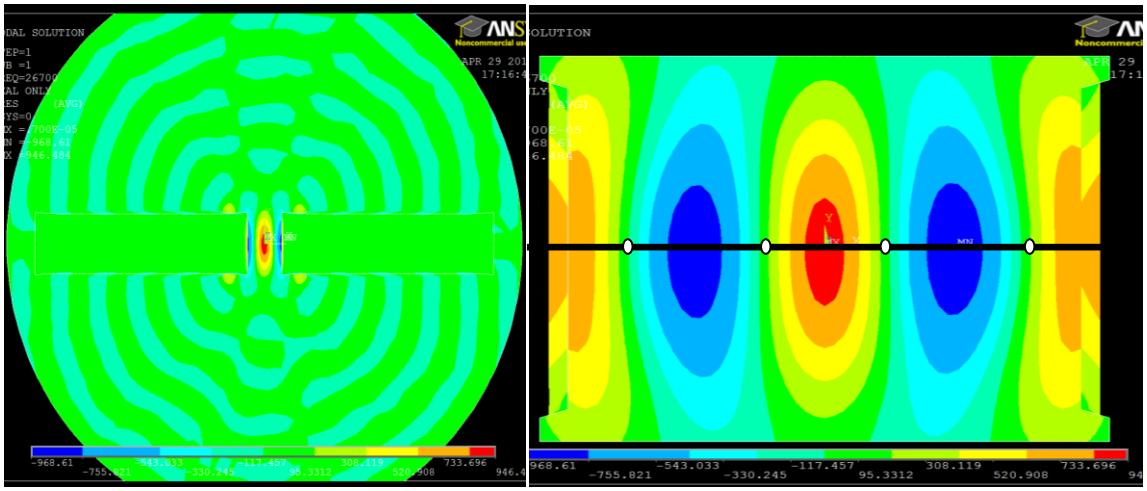


Figure 21 Pressure distribution at 0 degrees phase

The left image shows the distribution for the whole system but the area of interest is really the area between the actuators which is shown in the right image. This image is a contour plot of the pressure at some point in the cycle with each point not necessarily at its maximum position. The nodal regions in the pressure distribution are shown by the white dots along the centerline. A plot of the magnitude of the pressure is helpful in determining how the system changes with changes in phase. A plot of pressure magnitude can be created by looking at the nodal solutions along the centerline shown by the black line. Plotting the pressure magnitude for several different phases results in Figure 22.

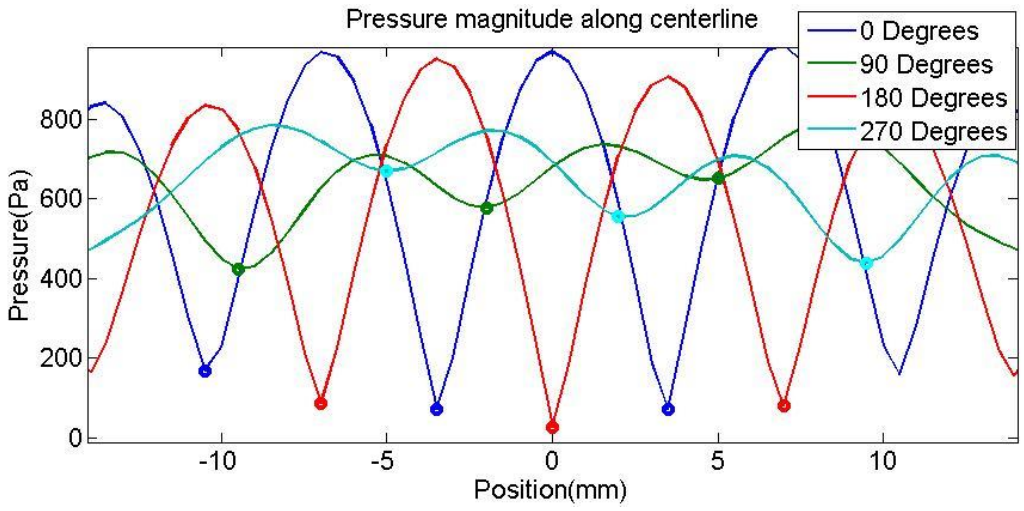


Figure 22 Pressure magnitude along centerline at several phases

In this plot the bold points each represent a node and depending on the phase there are either 3 or 4 nodes between the actuators. The number of nodes between the actuator is determined by the frequency and gap distance. As the phase is increased the position of the nodes moves to the right and Figure 22 shows that when a phase shift of 270 degrees is reached the node position is closer to the next 0 degrees node then the original node. What this means is that when the phase is shifted 360 degrees with a levitated object the object moves to the next node therefore the object can be moved from one side to the other by increasing the phase and moving node to node.

In the results section the ANSYS model will be compared to the experimental results. The ANSYS model gives values for the pressure magnitude between the actuators but it does not include the near field acoustic effect, which is a significant factor for the gap size being used. In the experimental system it is difficult to take pressure measurements in the oscillating field therefore the pressure magnitudes of the ANSYS model and experimental

model will not be compared. What will be compared are both the distance between the nodes in the model and the experiment and how the node position changes with phase.

3.2 Neural Network Modeling

Neural networks can be trained either 'online', 'offline' or a combination of both. Online training consists of an untrained network being implemented into a control system and weights and biases being updated based on actually using the plant. Offline training is using data either collected from the actual system or from another model to train a neural network before it is implemented. Often networks will be initially trained offline and then implemented with the ability to continue training or 'adapt' to the actual plant. This project will focus on using offline methods to create neural networks. The MATLAB Neural Network toolbox is used to construct and train the networks created in this project.

The first step to developing a NN offline is to collect the data that will be used to train the network. The network will only be as accurate as the data that is collected therefore it is important to collect data that fully represents the plant. Part of fully representing the plant is to have data that covers the complete range of inputs and outputs of the plant. It is important to cover the full range because while NNs are good at interpolating between points they do a poor job of extrapolating information. For the acoustic system being used the data that will initially be collected to represent the plant is the phase between the acoustic transducers and the position of the Styrofoam pellet. Additionally another model will be created that utilizes the phase, position and also the desired velocity of the pellet.

After the data is collected training speed and accuracy are optimized by preprocessing the data. The first preprocessing function is to delete inputs that are constant for the whole set of data because they do not contain any information about how the plant responds. The second preprocessing function is to map all the data between -1 and +1. The reason for this is that the sigmoid function shown in Figure 10 is often used in the hidden layer of a network and it quickly saturates when the inputs are large unless the input gains are very small. This saturation causes a small gradient in the weight updating process thereby increasing the required training time. The algorithm used by MATLAB to accomplish this mapping is given by

$$\mathbf{y = 2 * \frac{x - x_{min}}{x_{max} - x_{min}} - 1} \tag{37}$$

where x is the data value to be mapped and x_{min} and x_{max} are extremes of the data and y is the rescaled data point. The third step to processing data is to divide it into three sets: training, validation and testing. The training set is usually the largest and is what is used to calculate error gradient and update the weights. The error of the validation set is used during the training process to find when the network generalizes well. Initially with each weight update the error of both the test data and the validation set decreases. This continues to happen until the network becomes over-trained at which point the testing data set error will continue to decrease but the validation set error will begin to increase. This means that the network is no longer generalizing well and is instead fitting itself to the training data set and training should stop. The third data set called the testing set is used after training to compare error with the validation set. If the error between the validation and test set is not similar then

the data sets may not have been divided well. Typical data division percentages are 70% for training, 15% for validation and 15% for testing [26].

The data collected to determine the relationship between transducer phase and node position of the SWAL system is collected over the entire feasible control range. If the transducers are 30mm apart then the region of control is from 5mm to 25mm from the left face of the actuator. The procedure for initializing the system to be able to collect consistent data is given below:

1. Allow the system to warm up by tuning the transducer on and waiting a few minutes
2. Set the channels 1 and 2 phase to 0 degrees.
3. Align the phase between the two channels of the signal generator by pressing the 'align phase' button on the signal generator.
4. Levitate the Styrofoam ball between the actuators at the first node to the right of the center point (about 18mm).
5. Increase the phase of channel 1 on the signal generator until the ball is centered. If the system is warmed up then this will correspond to about 160 degrees.

After initialization data is collected by decreasing the phase on channel 2 of the signal generator until the ball is levitating 4.5 mm away from the left actuator face. From here position and phase data is collected as the phase is slowly increased by 3 degree until the ball is positioned about 26.5 mm away from the left actuator. To come up with an accurate measurement at each phase, several data points for position are taken and averaged. The average of several data points is used because the pressure on the ball is changing at the driving frequency causing the ball to oscillate over a small range even when the phase is

constant. A plot of the position versus phase data collected in this manner and used to create a NN and inverse plant is shown in Figure 23.

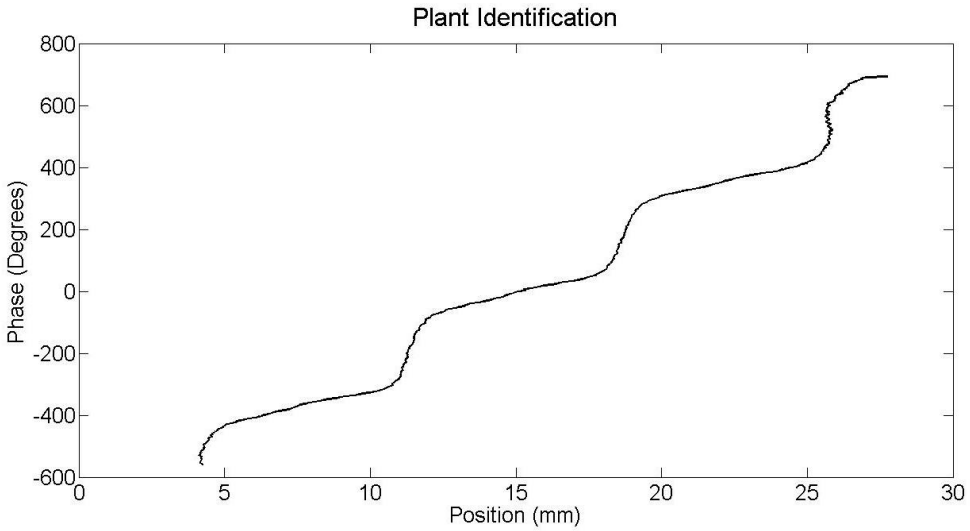


Figure 23 Plant Identification from Phase

Figure 23 clearly shows that the system is nonlinear and that there are regions where changing the phase by a few degrees results in a much greater position change then in other regions. The plant identification curve comes from averaging several position points at a specific phase and it is useful to look at what the standard deviation is for each of these averages.

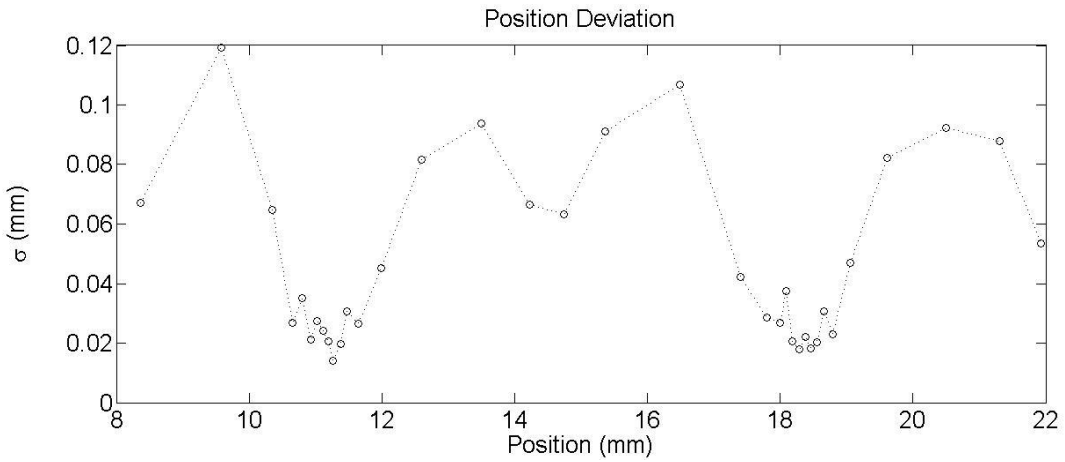


Figure 24 Standard deviation of position over the control region

The standard deviation of position shows the amount the object oscillates due to the changing forces on it. Figure 24 shows that there are certain regions where the object will oscillate more and that these regions are the same in which the object position is sensitive to phase change. The standard deviation plot is created by collecting several position points at 10^o phase increments and finding the average and standard deviation of those points. The oscillations occur much faster than the data sampling rate and therefore they cannot be controlled. This means that the optimum controller for this system will have error that approaches the standard deviation of the system.

The oscillations cannot be controlled but they can cause control problems when the position is sensed. Figure 24shows that the maximum standard deviation is .12 mm and a deviation of this large can cause control action that is unnecessary and destabilizing to the system. The filter described by equation (23) is added to avoid the scenario of the controller responding to the high frequency oscillations. Designing a low-pass filter is all about the tradeoff of between high frequency attenuation and lag. To select an appropriate filtering

coefficient, α, different values are tested with the same controller to see the impact of filtering. The end goal is to minimize error therefore the filtering coefficient that minimizes error over the operating range is selected.

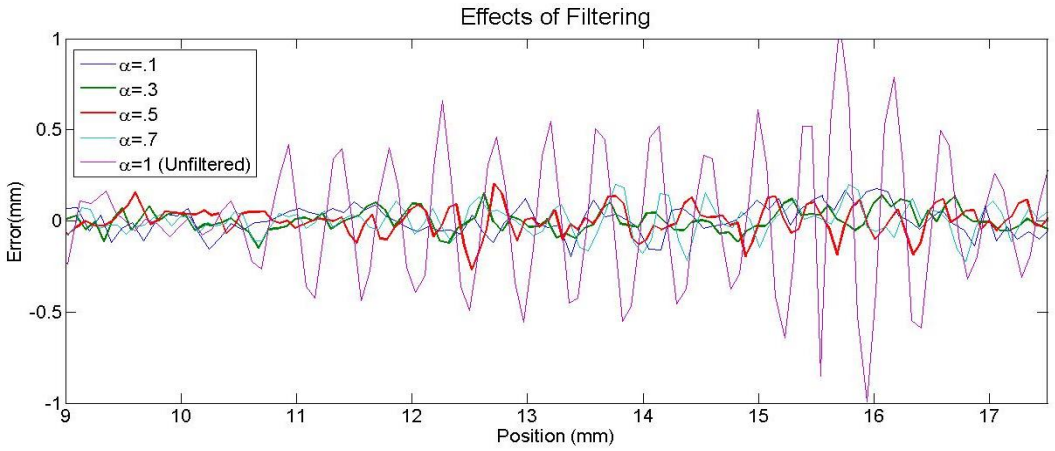


Figure 25 Effects of filtering on the controller error

Figure 25 clearly shows why some form of filtering is necessary for optimal performance. The fact that the error is small for all the controllers in the region from 9mm to 10.5mm shows that the cause of the error spikes is the oscillations of the object. In the region from 9mm to10.5mm the standard deviation is small in Figure 24—meaning there is a small amount of oscillations and little need of a filter in this region. The unfiltered controller in Figure 25 has particular problems in the regions where the standard deviation is large. In Figure 25 the object is moving with a speed of .5 mm/s which is slow and why there is not a notable lag between the filtered and unfiltered error, but as the speed is increased the lag becomes more observable. The bold red line has a coefficient of .5 and is the one selected to

use for this experiment because it performs well at different velocities and is a good balancing point between high frequency attenuation and lag.

3.3 Control Implementation

Control of both the position and velocity of the levitated object is the goal of this control system. The first goal is to be able to command a set position and have the object levitate at that position with no steady state error. The second goal is to be able to specify the velocity with which the object moves to the desired location. To achieve this second goal a path must be generated that specifies a desired position at a desired time. A point is generated one step ahead for a constant velocity using the equation

$$\boldsymbol{X_{d(n+1)}(t_n) = X_0 + V_g(t_n + \tau)} \tag{38}$$

Where X_o is the initial object position and τ is the time it take for one cycle of the control loop. It is also desired for the object to be able to follow a sinusoidal path based on a center point, frequency and amplitude. The equation that generates the desired path is given by

$$\boldsymbol{X_{d(n+1)}(t_n) = X_c + Asin(2\pi f(t + \tau))} \tag{39}$$

Where X_c is the center position about which the object will oscillate. A comparison between this desired position and real position is how the error is calculated and the controllers evaluated.

The first controller to be implemented is a PID controller with tuned gains that lead to optimum performance for this nonlinear system. The block diagram of this controller is shown in Figure 26.

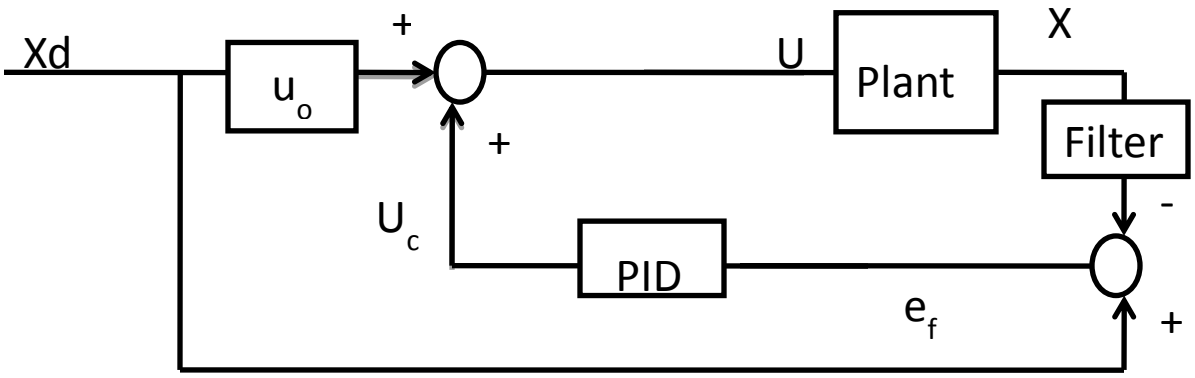


Figure 26 PID control diagram

The PID block is governed by

$$\mathbf{U_c} = [\mathbf{k_p e_f(n-1)} + \mathbf{k_i} \sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{e_f(i) t_s} + \mathbf{k_d} \frac{\mathbf{e_f(n-1) - e_f(n-2)}}{\mathbf{t_s}}] \quad (40)$$

Where t_s is the time between samples. U_c is the control action of the PID block from the error term. U_o is the expected control input for the desired position and is calculated assuming the plant is linear by using the slope between the two endpoint of Figure 23. For this system U_o is

$$\boldsymbol{U_o = 44.8X_d - 680} \quad (41)$$

The most effective gains found for this system are K_p=35, K_i=.5, and K_d=0. No derivative gain is used because it only amplifies the oscillations that the filter tries to attenuate.

The second type of controller that is implemented is an open-loop inverse plant controller shown in Figure 14. To create this inverse plant the data shown in Figure 23 is used to create input output pairs where the input is position and output is phase.

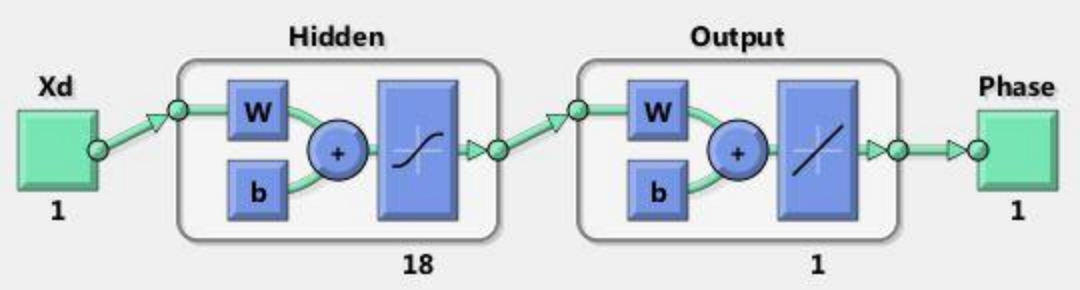


Figure 27 Inverse Plant Neural Network

A neural network with a single hidden layer and an output layer were able to successfully model the system dynamics. Out of the 801 data point of the inverse plant network 70% of the data points are randomly selected to be used for training, 15% for testing and 15% for validation. The network is trained for 935 epochs but the best epoch was 835 and has a mean square error value of 4 degrees. Training was stopped at epoch 935 because the validation data performance did not improve for 100 epochs. The progression of the plant training is shown in Figure 28.

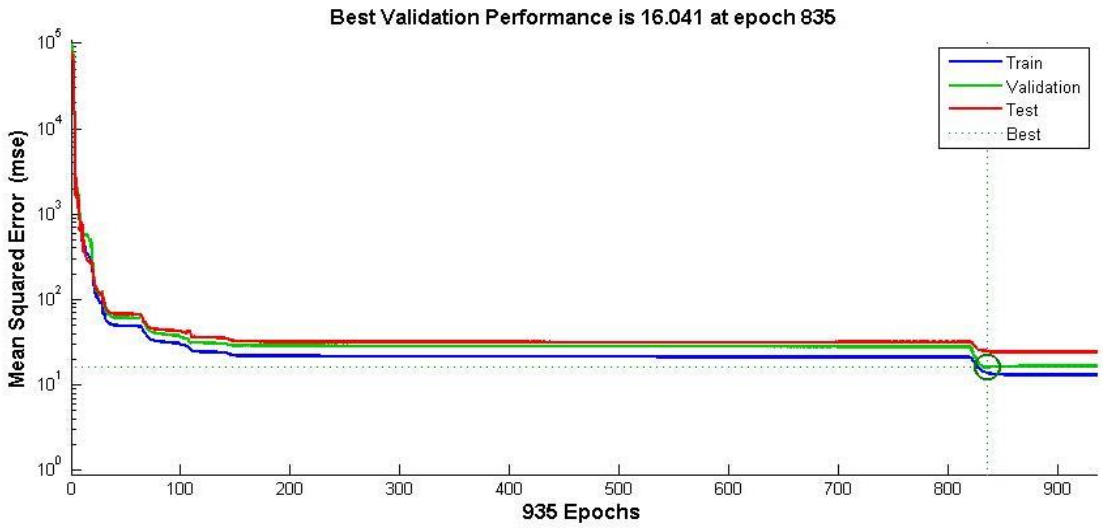


Figure 28 Inverse plant training performance

The hidden layer consists of 18 neurons with a sigmoid activation function while the output layer consists of a linear activation function. There is only a single output therefore the output layer has a single neuron. This inverse plant is all that is needed to create the controllers proposed in Section 2.4 except for the feedback neural network controller shown in Figure 18.

At slower velocities the inverse plant based on the data shown in Figure 23 gives accurate results but as the velocity is increased this model becomes less accurate. The model becomes less accurate because as the desired object velocity increases the plant dynamics change and these changes are not reflected in the NN model. The fact that the plant changes at different speeds is clearly shown by Figure 29. This figure shows position versus phase relationship for the object traveling at different velocities. These plots were obtained by recording the phase and position data of the controllers with the least error at each speed.

The top plot shows the object traveling from the 6mm position to the 25mm position—left to right, and the second plot shows the object traveling from right to left.

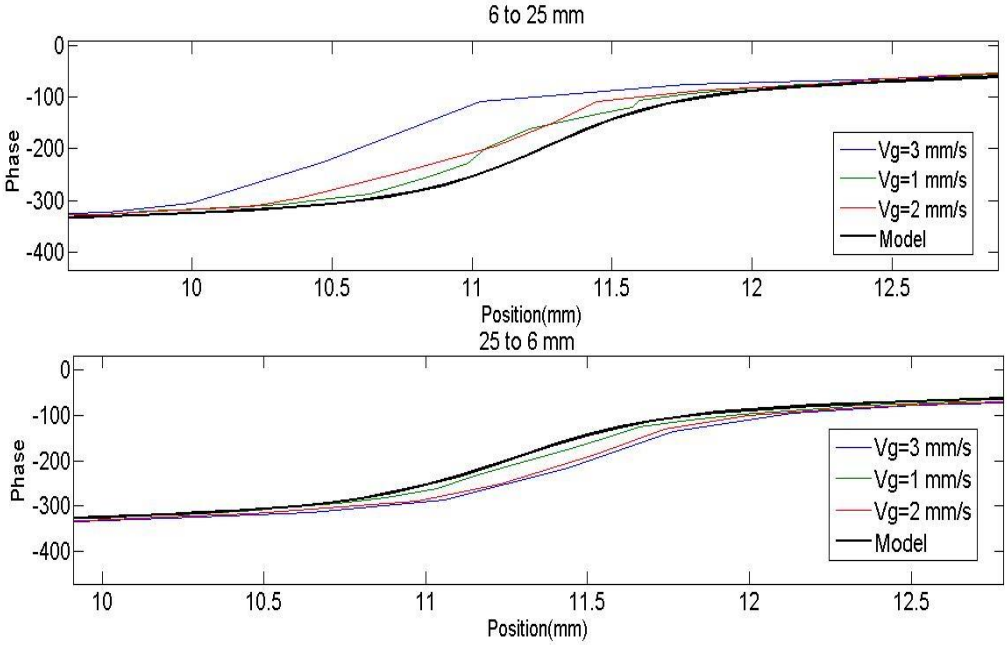


Figure 29 Plant Identification at various velocities

In traveling left to right the top plot shows that the model always under-predicts the necessary phase to achieve a certain position. In traveling right to left the bottom plot shows that the model always over-predicts the necessary phase. For Figure 29 data was collected for the same range as Figure 23 and the overall shape is similar, but Figure 29 is zoomed in to clearly show the impact velocity has. From this data another inverse plant model was created that has velocity and desired position as an input. The neural network that was used to create this new inverse plant is shown in Figure 30.

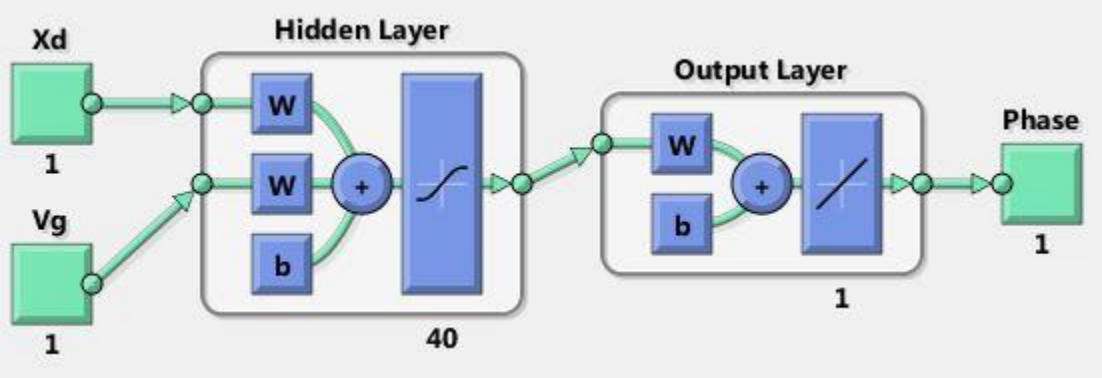


Figure 30 Inverse plant neural network with velocity input

This neural network has the same structure as the network in Figure 27 except that this has multiple inputs and more neurons in the hidden layer. The velocity dependent network is trained using the phase and position data collected when moving the object at various speeds. For this network 1517 data points are collected and again are separated randomly into training, validation, and testing sets—70% of the data is for training and 15% each for validation and testing The best performing weights have a root mean square error of 8.71 degrees. The root mean square error for this plant is larger than the velocity independent plant because the network is trying to model a more complicated relationship. The error could be decreased by increasing the number of neurons but the cost is a loss of generality. The network trained for 154 epochs but the best performance from the validation test set was at epoch 54.The progress of the plant training is shown in Figure 31.

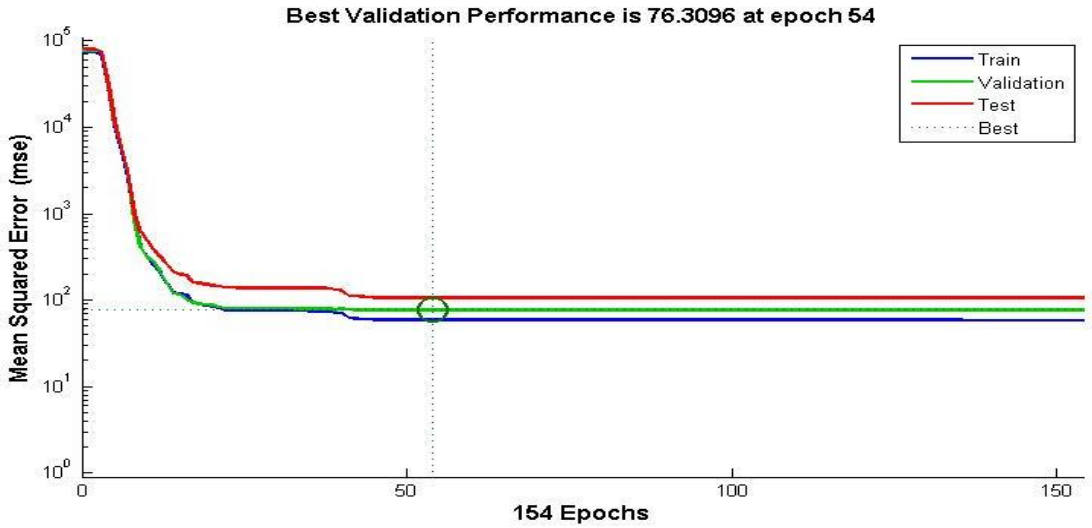


Figure 31 Velocity dependent inverse plant training performance

To show how this network captures the dynamics of the system a plot of the output over the control region (6-25mm) at different velocities is shown in Figure 32. This figure shows that as the velocity increases the position vs. phase curve shifts to the left meaning that more phase change is required at higher velocities.

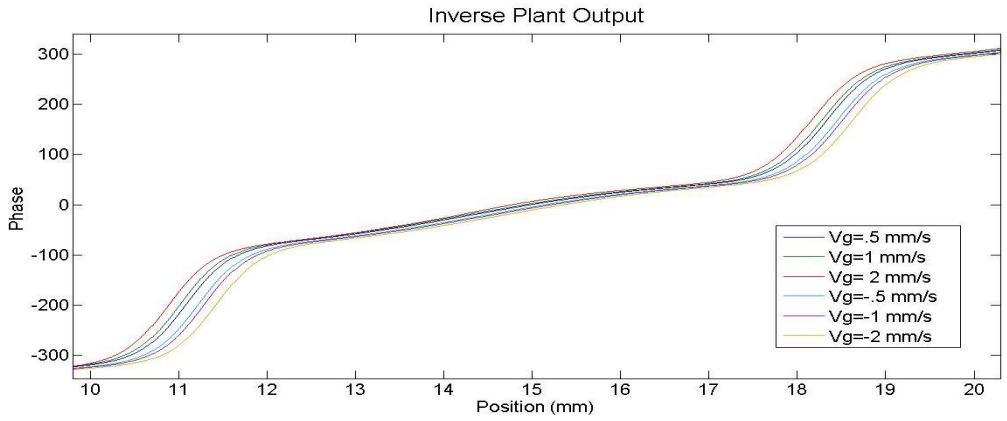


Figure 32 Inverse plant neural network output at different velocities

One possible reason that this shift occurs is that the object itself affects the acoustic field and if the object is commanded to change quickly it is expected that it will disturb the surrounding acoustic field more. This added disturbance can impact the acoustic near-field causing the plant behavior to change.

Another controller that is tested is the neural network controller with feedback which is shown in Figure 18. This controller has desired position, goal velocity, and error feedback as inputs. Training this network offline is not as straightforward as training the inverse plant. Training this network requires having a plant model and cascading this with the controller. A diagram of this cascaded training network is shown in Figure 33.

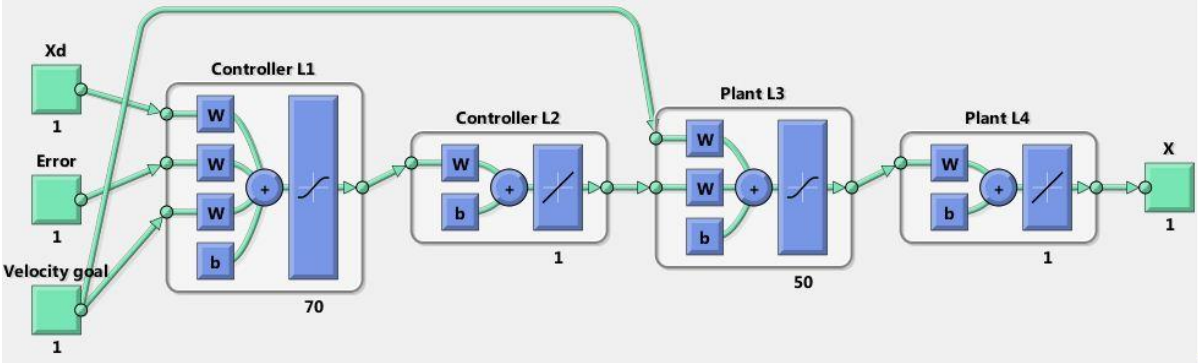


Figure 33 Neural network controller training network

The first two layers of the training network are what turn out to be the neural network controller that is implemented. The second two layers are a model of the plant which was found in the same way as the inverse plant shown in Figure 32. The weights of the plant

layers are fixed during training because it is the controller which is to be trained not the plant. The output of the training network is position, X, which is trained to be the same as the Xd value. The only way the Xd input and X output can match is if the controller weights change so that it becomes both an inverse plant and feedback controller. The weights attached to the Xd and velocity goal inputs will be the model of the inverse plants just like in Figure 30. The error input weights are what will become the feedback portion in the controller. The training data used is from the open-loop inverse plant controller and is shown in Figure 34.

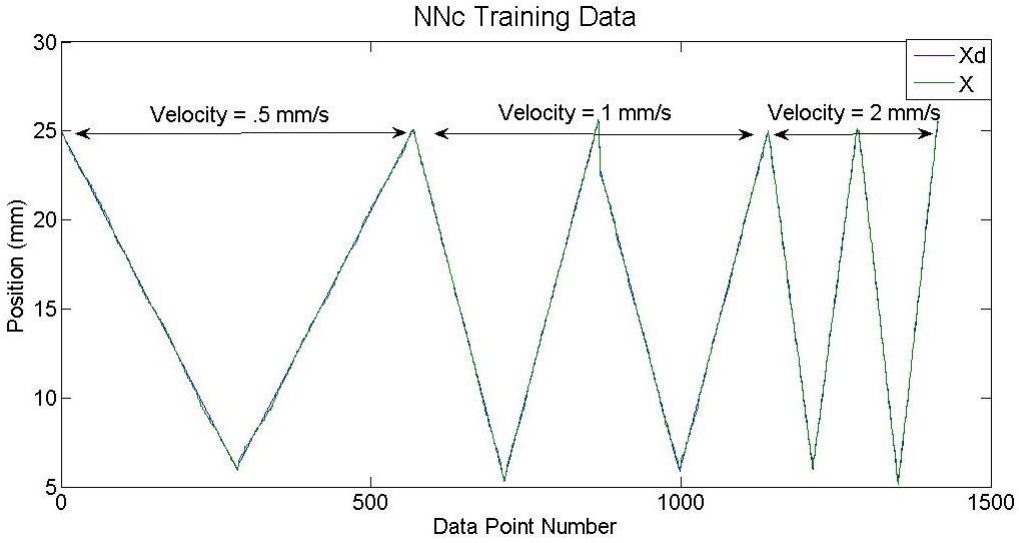


Figure 34 NNc Training Data

The training data contains 1415 points and covers several velocities going from 6mm to 25mm as well as going from 25mm to 6mm.The open-loop controller data is used because it does not contain error feedback but does give an error value which can be used to find the weights that would minimize the difference between position input and output.

The actual process of training the feedback neural network controller takes more time because there are four layers through which the error must back-propagate. The Marqurdt-Levenberg algorithm is used to optimize the performance of this network. The network is trained for 165 epochs and the best epoch was 65 and has a root mean square error of .077mm. The performance measure of this network is in mm instead of degrees like the other networks because the training network contains both a controller and a plant model which has an output of position. The performance progress of the training, validation and testing data sets are shown in Figure 35.

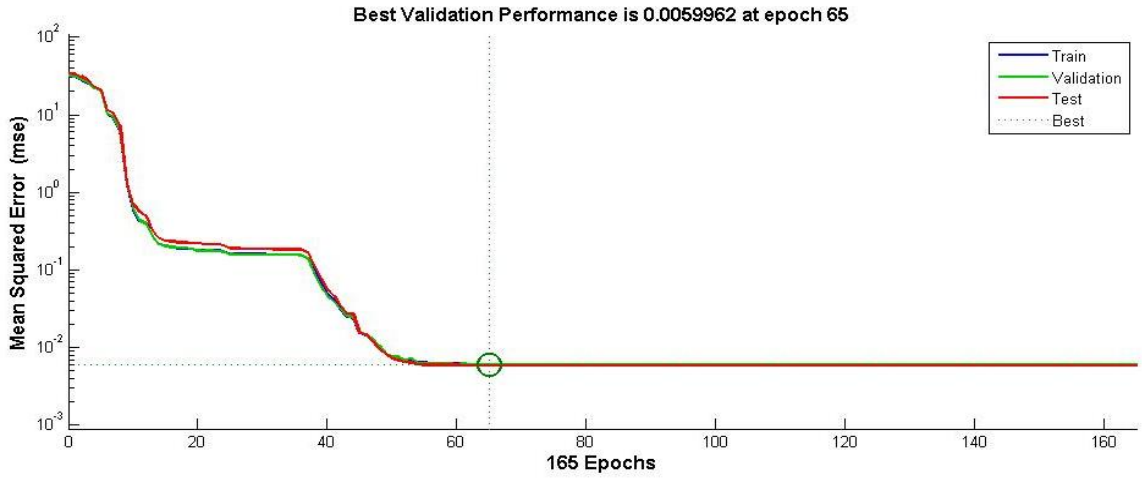


Figure 35 Neural network controller training performance

Once training is complete the weights of the first two layers are then transferred to a network that is identical in structure to the first two layers of the training network. This network will be the actual controller implemented and a diagram of this network is shown in Figure 36.

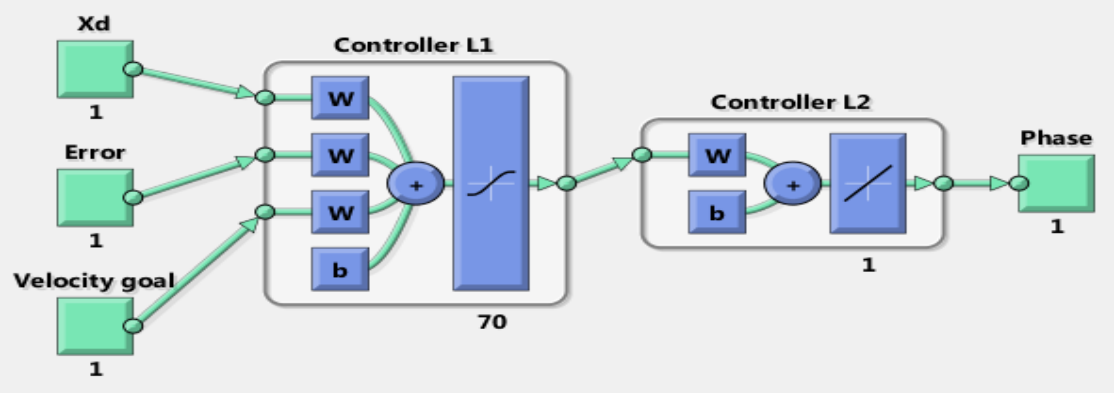


Figure 36 Neural Network controller

After training the network with the data shown in Figure 34 the controller is tested by comparing the error of the training data, X_d-X, to the error produced by the simulated plant when the trained controller is applied. The inputs to the controller are the first 500 training data point and the results of the simulation are shown in Figure 37

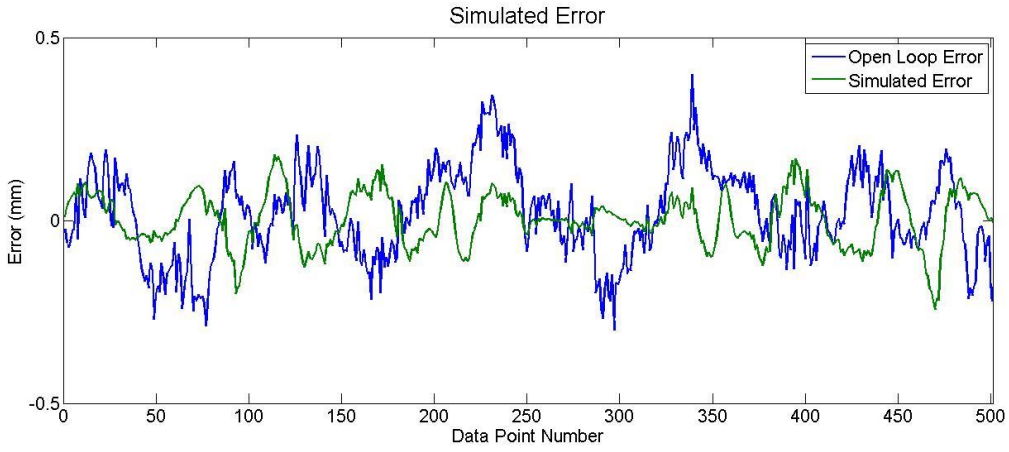


Figure 37 Neural network controller with feedback validation

The simulated results show the controller reduces the amount of error in the training data which indicates that it is correctly modeling the inverse plant and compensating for error. To see the effects of the error feedback on the controller, output plots of the network over the control region at a constant velocity and different errors are used.

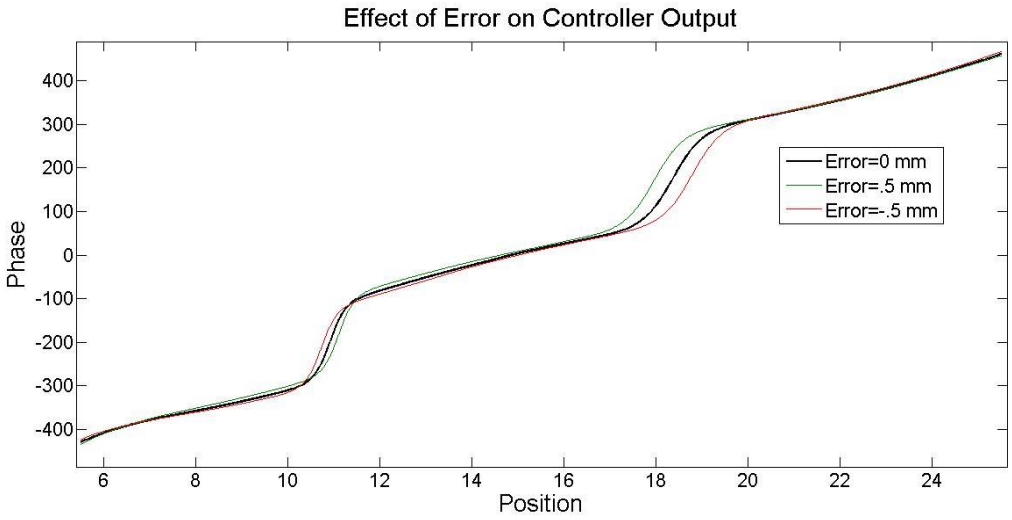


Figure 38 Neural network feedback controller output at different errors

Figure 38 shows the output of the neural network feedback controller when the input is a velocity goal of 2 mm/s over the 6 to 25mm control region with errors of 0mm, -.5mm, and .5mm. The plot shows that for the majority of the region when error is positive the controller outputs a higher phase value and when the error is negative the output is lower than the zero error line—this is correct control action that leads to less error. The important thing to note about the phase output is that the amount of control action depends on position even for a

constant error. In the areas where the system is sensitive to phase changes, such as from 12mm to 17mm, the amount of control action is small. From 18mm to 20mm the system is not as sensitive to phase change therefore a greater amount of phase change is required for a constant error. Figure 38 shows that between 10mm and 12mm positive error is at a lower phase angle then the error free output—this is the opposite of what is generally wanted. The network outputs in this region could mean several things the first is that the plant behaves differently over this region, second that there were not enough training data points in this region to create a good general model, or third that more training of the network is necessary.

The finite element model and the neural network model both show that it is possible to move a levitated object between the actuators gap by manipulating the phase between the actuators. To create an accurate neural network model data needed to be carefully collected and used to train the networks. The data collected showed that the oscillations in the acoustic system require a filter to be applied to the position sensor in order for effective control to be possible. When testing the controllers it was noticed that as the object speed increased the plant behaved differently therefore a set of models were created to incorporate these dynamics. After the controllers are created and implemented in the experimental setup their performance needs to be evaluated and compared. Chapter 4 discusses how the controllers will be evaluated at different speeds and plant offsets. Chapter 4 also compares the experimental modeling to the finite element model and draws some conclusions.

Chapter 4

Results and Conclusion

4.1 Experimental Setup

In this experiment various control schemes are used to manipulate a levitated object between two bolt clamped Langevin type transducers. To implement control schemes with feedback and to run the transducers five components are needed.

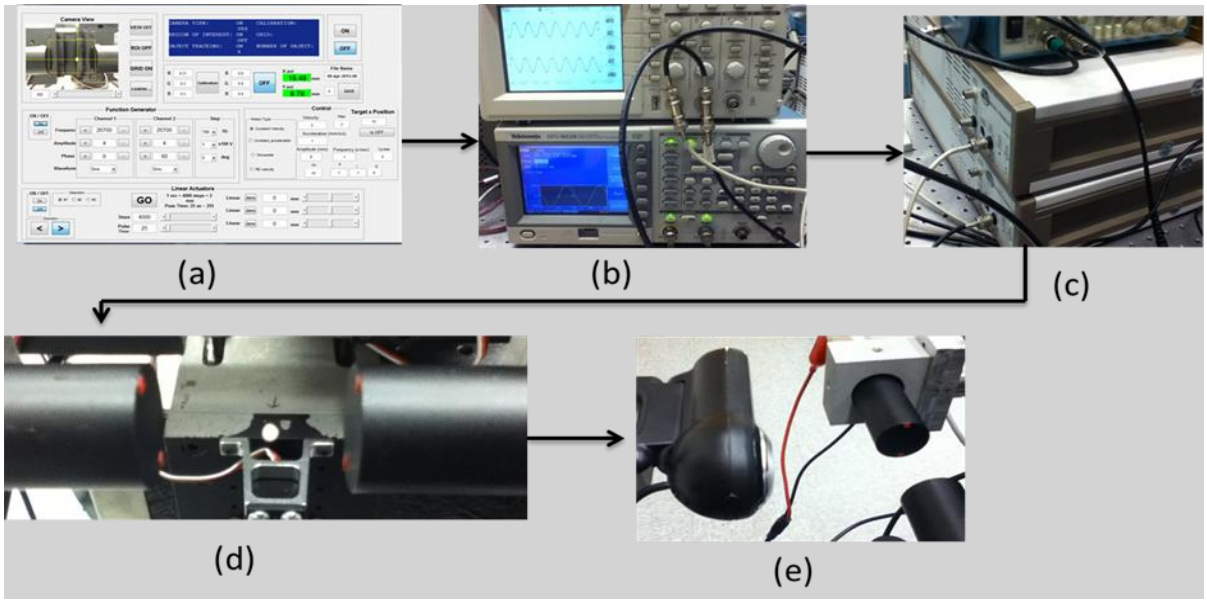


Figure 39 System Flow Chart

(a) MATLAB GUI (b) signal generator (c) signal amplifier (d) transducers (e) camera

The first part of the system shown is a computer that runs a MATLAB GUI that calculates the control action and communicates that information to the signal generator. The GUI was created by Joong-Kyoo Park [3] and control functionalities were added for these experiments. The signal generator is a Tektronix AFG 3022B signal generator which is able to produce two sine waves of different phases. The signals are then amplified 100 times by two Trek Piezo drivers. The amplified signal is then passed to the two transducers which produce the standing wave. The transducers are set up so that 0 degrees phase difference on the signal generator means that they are moving at the same speed but in opposite directions. A camera takes a picture of the gap between the actuators which is sent to the computer. The ball is white and the transducers are black with red control points that allow the computer to calculate the position of the ball with respect to the lower corner of the left actuator. With the updated position information the computer calculates the next control input.

The experimental setup is the same used by Park and Ro in *Non-contact Manipulation of Light Object Based on Parameter Modulations of Acoustic Pressure Nodes* [3].For this particular system Park and Ro found several specific parameters that optimize acoustic power. The driving frequency used is 26.7 KHz because that is the natural frequency for the transducers. Using the natural frequency leads to the greatest amplitude of displacement which is 7μm. To achieve levitation over a gap of 3cm the voltage of the signal generator is set to 4 volts peak-to-peak.

To interface with the system a GUI is created in MATLAB. Using this GUI simplifies changing function generator parameters, positioning the actuators, calibrating the

tracking, changing control parameters, and saving data. An image of the GUI is shown in Figure 40.

Figure 40 Graphical interface control panel

The top left corner of the GUI shows the image of a levitated ball being tracked. The position of the ball is found by using the four red control points to set up a coordinate system based on the lower left hand control point. The coordinate of the ball is shown in the green boxes and is updated every cycle. The object tracking and coordinate calibration are tuned using the R G B variables so that if the lighting changes it is still possible to track the object position. The function generator block allows for changing frequency, amplitude, phase, and wave form. The gap between the actuators is manipulated by the variables in the linear

actuator block. The controls block allows for different control algorithms to be tested, different PID gains, and filtering changes. The control block also allows changes in the goal velocity and set-point for each run. Data from each run is saved and stored in a MATLAB data file. The system collects phase, x and y position, time, error, desired position, and for the PID controller control action due the PID gains.

4.2 Controller Response at Various Speeds

In this project there are eight different control setups that are tested with various gains and filtering values. To clearly distinguish which control setup is being used a naming convention is given in Table 2.

Table 2 Control Scheme Names

Abbreviated Name	Inverse Plant Used	Block Diagram
PID	n/a	Figure 26
INV	Velocity Independent	Figure 14
iPID	Velocity Independent	Figure 17
IMC	Velocity Independent	Figure 16
INVv	Velocity Dependent	Figure 14
iPIDv	Velocity Dependent	Figure 17
IMCv	Velocity Dependent	Figure 16
NNc	n/a	Figure 18

Several of the controllers require the use of an inverse plant and there are two different types of inverse plants that are used. The first inverse plant is only dependent on position and is shown in Figure 23. The second inverse plant is dependent on the goal velocity and is shown in Figure 32. The only thing using this velocity dependent plant changes is each control scheme has multiple inputs—the first being position and the second being desired velocity.

Each controller is tested by moving the levitated ball from the 6mm to the 25mm position at various velocities. The velocities most often used are .5 mm/s, 1 mm/s, and 2 mm/s but some testing was done at 4 mm/s as well. The plant responds differently when moving left to right then when moving right to left therefore tests are done in both directions. To compare the results of one controller to another steady state error once the object has reached its set-point is looked at as well as the root mean square of the error. The RMS error value is calculated using

$$\boldsymbol{RMSerror} = \sqrt{\frac{\sum_{i=1}^{n}\left(x_{d_i} - x_i\right)^2}{n}} \tag{42}$$

where n is the number of data points collected during one run.

The most commonly used controller in the world is a simple PID controller and it will be used as a baseline for testing done in this experiment. The best gains found that work for the range of tested velocities are K_p=35, K_i=.5, and K_d=0.

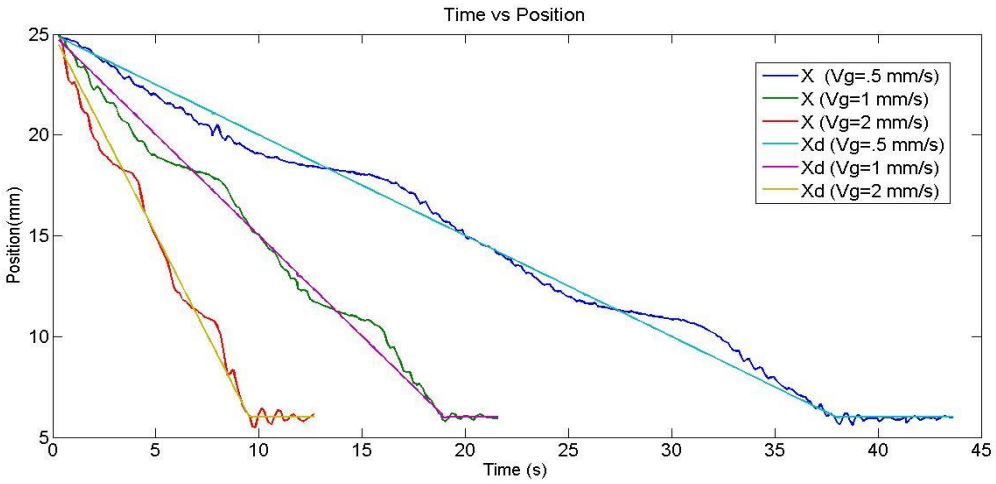


Figure 41 PID controller results at various speeds

There are six lines in Figure 41 which show how the PID controller performs at .5 mm/s, 1 mm/s, and 2 mm/s. The plot shows a straight line which is the desired trajectory and the actual trajectory. The results in Figure 41 show that while the object is able to accurately follow the desired path there is significant periodic error. This periodic error is expected because the system is nonlinear and therefore a gain that works well in one region would not work well in another. This controller does have integral control therefore it settles to the set-point value.

Next an open-loop inverse plant is tested. These tests will show how accurately the neural network modeled the nonlinear acoustic system. The first system tested is the plant that only uses position as an input.

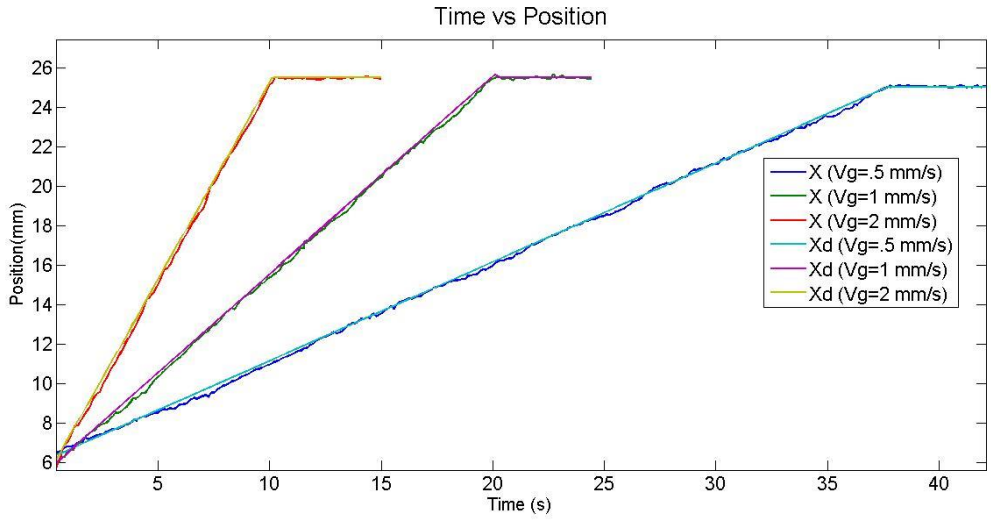


Figure 42 INV controller results at various speeds

Adding the inverse plant even though this is an open-loop controller resulted in a much better performance than the PID controller. That is because the inverse plant takes into consideration the nonlinearity of the acoustic system. Figure 42 shows a plot of time versus position but because the error is much smaller than the PID controller it is hard to exactly see what the error is. To assess the controller performance easier a plot of the position versus error will be used instead of a time versus position plot. The error is plotted versus position because regions where the controller performs poorly will show up clearly. The error versus position plot for the INV controller is shown in Figure 43.

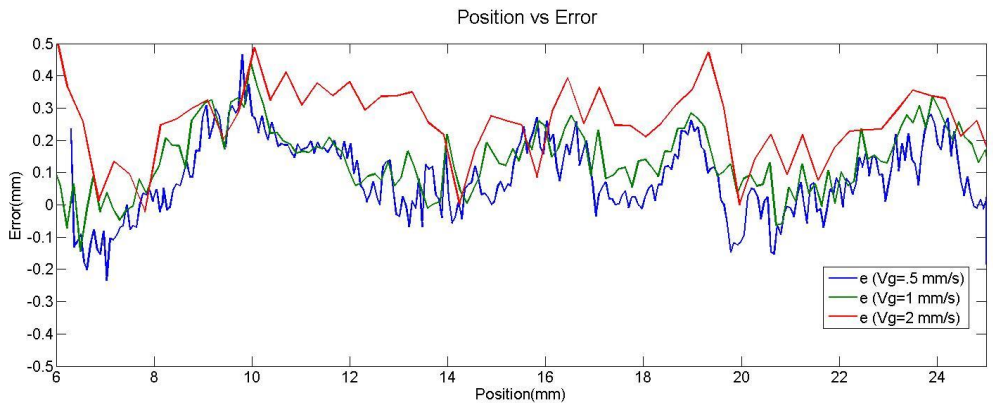


Figure 43 INV controller error at various speeds (6mm to 25mm)

The position vs. error plot shows that most of the error is positive meaning the controller tends to underestimate the amount of phase needed for a certain position. The second thing to notice is the trend of underestimating the control needed increases as the velocity goal increases. To quantify this trend the error of each run was summed and divided by the number of data points in that run. The result of this is an error average of .044mm, .12mm, and 18mm for velocities of .5 mm/s, 1 mm/s, and 2mm/s respectively. This plot is only for going from 6mm to 25mm—a plot of going 25mm to 6mm would show error that was mostly negative. The reason the slower speed run shows a noisy error signal is because data points were taken at closer intervals then in the faster traveling runs. If the sampling frequency were greater the higher speed runs would also be noisy. The RMSe of a run using the INVv controller which has position and velocity as an input is shown in Table 4. Comparing the RMSe of the INVv and INV controllers reveals that at low speed the INV controller performs better but as the speed increases the INVv surpasses the INV controller.

The inverse plant models can be combined with a PID controller to form the iPID and iPIDv controller. The effectiveness of the iPID controller is shown in Figure 44.

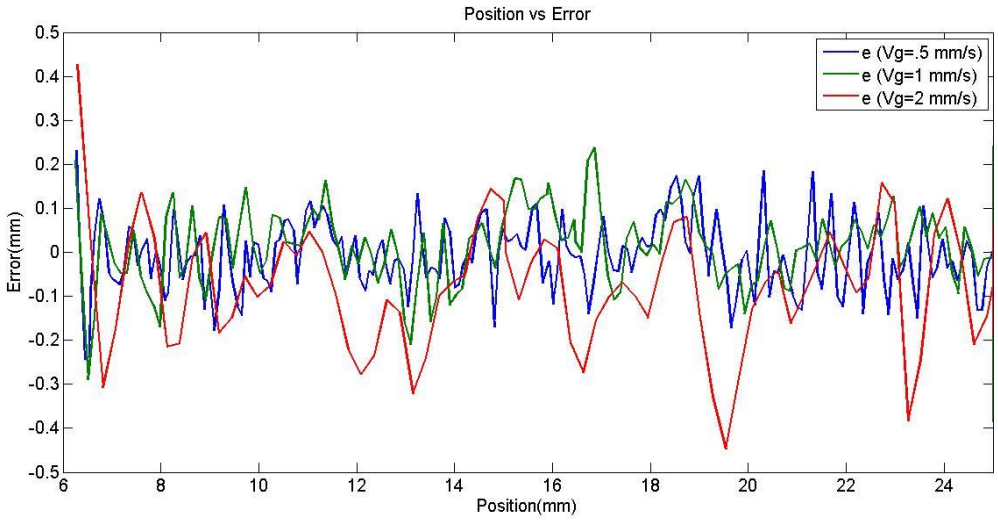


Figure 44 iPID controller error at various speeds (6mm to 25mm)

The iPID error plot shows that adding feedback drives the error to be centered about 0. Multiple gains were tried but it was found that the best results were from K_p=1, K_i=0, and K_d=0.

Table 3 RMSe of several different PID gains

No.	Xo (mm)	Vg (mm/s)	Kp	KI	KD	RMSe (mm)
1	6	0.5	1	0	0	0.078
2	6	0.5	1	0.1	0	0.118
3	25	0.5	1	0.1	0	0.131
4	6	1	1	0	0	0.089
5	6	1	1.1	0.1	0	0.187
6	25	1	1	0.1	0	0.195
7	6	2	1	0	0	0.145
8	6	2	1.05	0.05	0	0.164
9	6	2	1	0.05	0	0.167
10	25	2	1	0.05	0	0.170
11	25	2	1.05	0.1	0	0.224
12	6	2	1	0.1	0	0.285
13	25	2	1	0.1	0	0.302
14	25	2	1	0.2	0.02	0.321
15	6	2	1	0.2	0.02	0.329
16	6	2	1.2	0.2	0.02	0.339
17	25	2	1.2	0.2	0.02	0.358

Table 3 shows that the best performance based on RMSe of the gains tried was when K_p=1. When the proportion gain is just one the iPID controller simply becomes an inverse plant controller with feedback and is shown in Figure 15. The reason that the gains tried in the iPID controller are much small than the PID controller is because the iPID controller feeds back into the inverse plant model which has an input of position. The PID controller feeds back directly to the acoustic system which has an input of phase. The range of position is

from 6mm to 25mm while the range of phase is -400 degrees to 400 degrees—this means that the gains for feedback directly into the plant need to be large to have the same impact.

The next step is to compare the iPID controller performance to the iPIDv controller that uses the velocity dependent plant. Again the best results were obtained when K_p=1 and , K_i=0, and K_d=0. Figure 45shows a large amount of error around the 25mm position this 25mm position corresponds to the first few seconds of each run because this run goes from 25mm to 6mm.

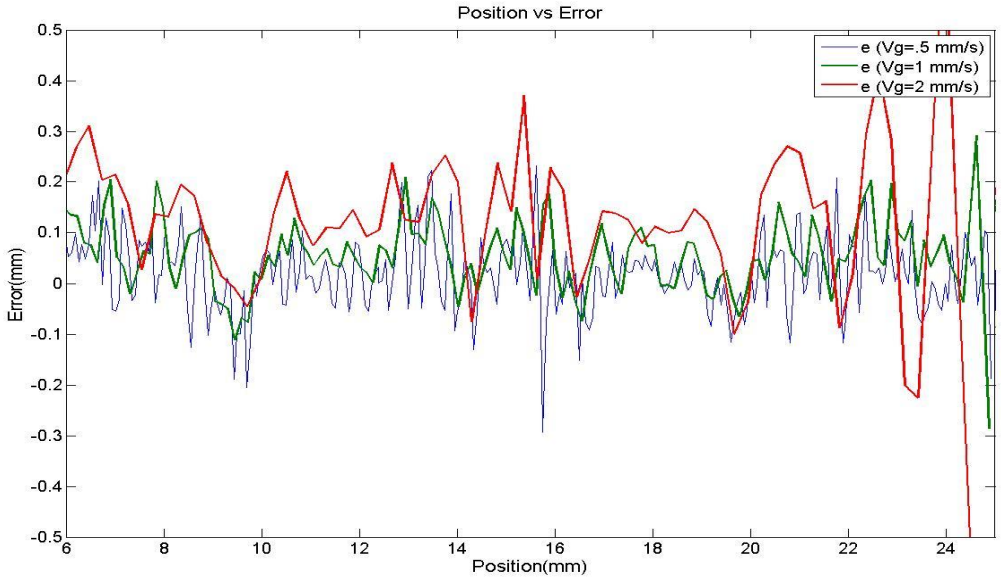


Figure 45 iPIDv controller error at various speeds (25mm to 6mm)

The internal model controller described by Figure 16 is able to apply feedback control to the nonlinear systems by comparing the nonlinear model to the actual system. Unlike the PID controllers, the IMC controller is able to provide control input based on the region the object is in. The results of using this controller are shown in Figure 46

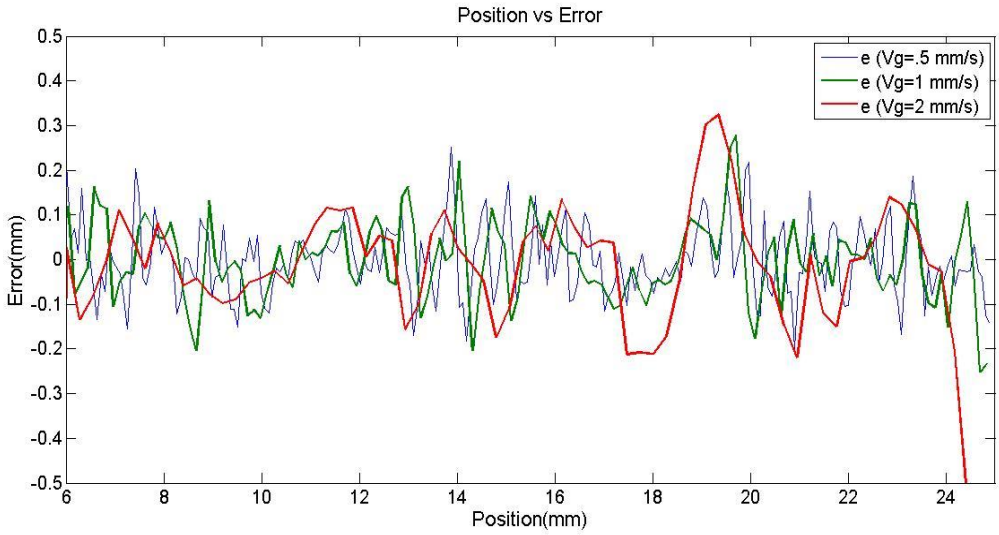


Figure 46 IMC controller error at various speeds (25mm to 6mm)

One thing to notice about the error of the IMC controller compared to the PID controllers is that as the speed in increased the amplitude of the error does not increase significantly. Figure 44 of the iPID controller clearly shows more error peaks as the velocity increases and the error curve magnitude increases as velocity increases.

The IMCv controller uses a plant model that depends on the goal velocity for its output. The results using the IMCv controller are shown in Figure 47.

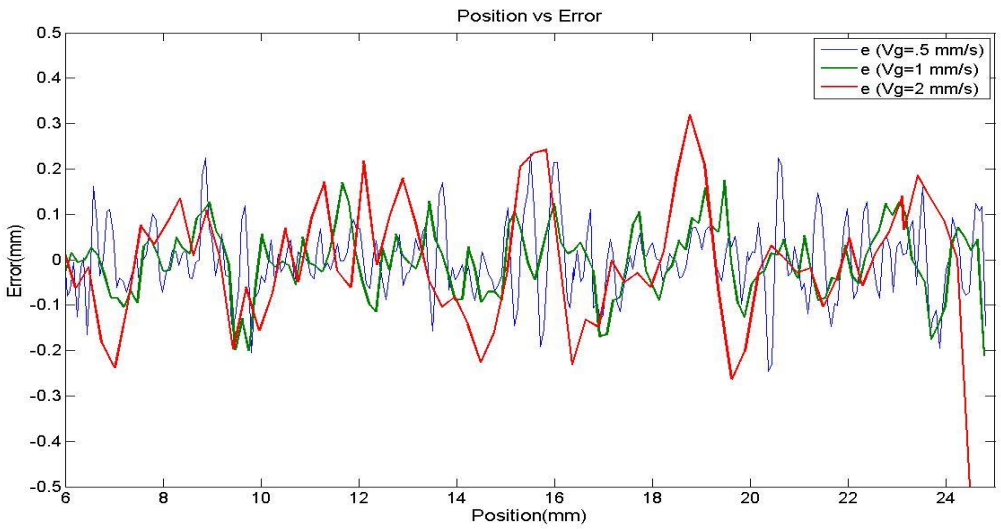


Figure 47 IMCv controller error at various speeds (25mm to 6mm)

Comparing the results of the different controllers by sight is difficult even using a plot of the errors. Table 4is used to numerically compare the performance of each controller based on the RMSe and the peak error amplitude.

The final tested controller is the NNc shown in Figure 18. This controller is tested at velocities of .5 mm/s, 1 mm/s, and 2 mm/s and the result is shown in Figure 48

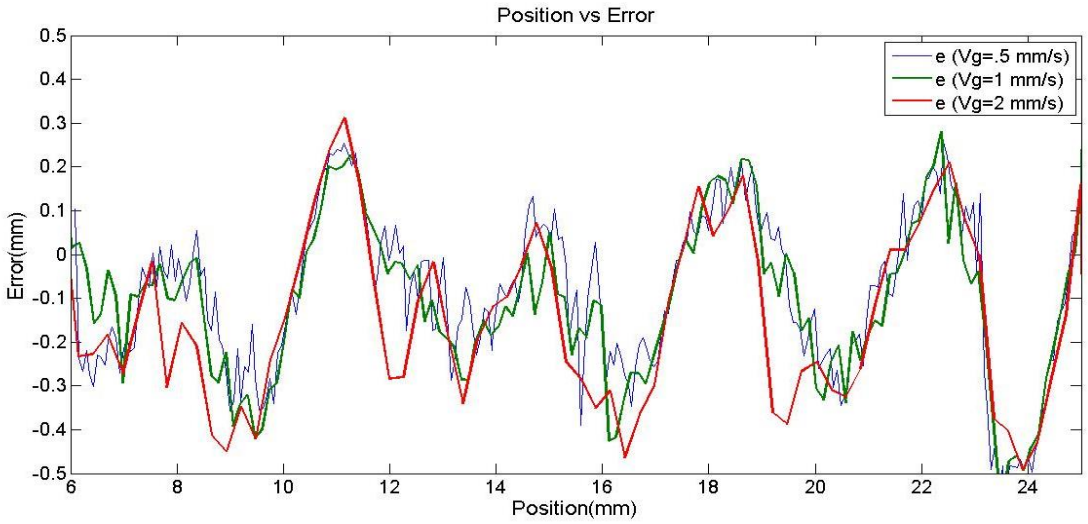


Figure 48 NNc controller error at various speeds (6mm to 25mm)

The first thing to notice about the error plot in Figure 48 is that this controller does not perform nearly as well as the other controllers including the open-loop inverse plant controller. The second thing to notice is that for each run the error profile is nearly the same. Not only is the profile the same but the magnitude is about the same as well. This is contrary to other controllers where the error increases as speed increases. One of the reasons that this controller did not perform as well as the other model based controllers is because generating the controller depended on using a model of the plant as shown in Figure 33. The other controllers were trained directly from experimental data meaning they have one less layer of complexity and less propagation of error. Although this controller did not perform as well as the other models it is still a stable controller and is a good candidate for online adaptive use. The NNc is a good candidate because online training would be able to further optimize the weights attached to the error feedback component of the controller.

The error versus position plots are useful when looking at characteristics of a single controller but to look at multiple controllers it is useful to define a performance function and compare the controls based on that. For this experiment the RMSe is the performance function used to evaluate the controllers. Although this is a useful performance measure it does not show if there is a steady state error once the object reaches the set-point. Table 4 contains the RMSe for each of the plots shown in this section. The column labeled e_{max} is the maximum amplitude of error for each trial.

Table 4 Controller performance measure

No	Controller	Xo (mm)	Vg (mm/s)	RMSe	em (mm)
1	iPIDv	25	0.5	0.0742	0.3
2	IMCv	25	0.5	0.0781	0.22
3	iPID	6	0.5	0.0781	0.15
4	IMC	25	0.5	0.0860	0.23
5	INV	6	0.5	0.1304	0.45
6	INVv	25	0.5	0.1732	0.45
7	NNc	6	0.5	0.1949	0.38
8	PID	25	0.5	0.5477	1.2
9	IMCv	25	1	0.0707	0.2
10	iPID	6	1	0.0894	0.21
11	iPIDv	25	1	0.0949	0.2
12	IMC	25	1	0.0985	0.25
13	INV	6	1	0.1581	0.45
14	INVv	6	1	0.1788	0.37
15	NNc	6	1	0.1949	0.4
16	PID	25	1	0.5291	1
17	IMCv	25	2	0.1342	0.3
18	IMC	25	2	0.1378	0.31
19	iPID	6	2	0.1449	0.45
20	INVv	25	2	0.1732	0.48
21	iPIDv	25	2	0.2049	0.33
22	NNc	6	2	0.2098	0.43
23	INV	6	2	0.2324	0.5
24	PID	25	2	0.6000	1.4

The table is sorted by the goal velocity and the RMSe value therefore each velocity is grouped together with the first in the group having the least amount of error. The highlighted rows show the best performance at each speed is a controller that includes the velocity dependent plant. The performance improvement is not large and does not guarantee a minimum value of e_{max}.

4.3 Controller Response to Plant Offset

When assessing a controllers it is not only important to see how well it performs under ideal conditions but also how it responds when the plant is different than modeled. In an actual system the plant will always be different from the model because perfect plant identification is impossible. Even if an excellent model is developed time, cyclical loading and heat take their toll on any apparatus causing plant and model mismatch. To simulate this wear the controllers are run with a plant offset. The phase is what is being controlled therefore it is the phase that will be changed. In section 3.2 Neural Network Modeling the procedure for initializing the system is described. In step five the phase of channel one of the signal generator is increased until the object is centered between the actuators. After step five is done the phase is then increased to the desired offset amount.

The effect of the plant offset can be seen by having the object track a sine wave. The desired object path is given by equation (39) where the center position is 15 mm, amplitude is 6 mm and frequency is .1 Hz. The IMC and iPID controller are tested with plant offsets of 15^o and $30^o$and shown in Figure 49 and Figure 50.

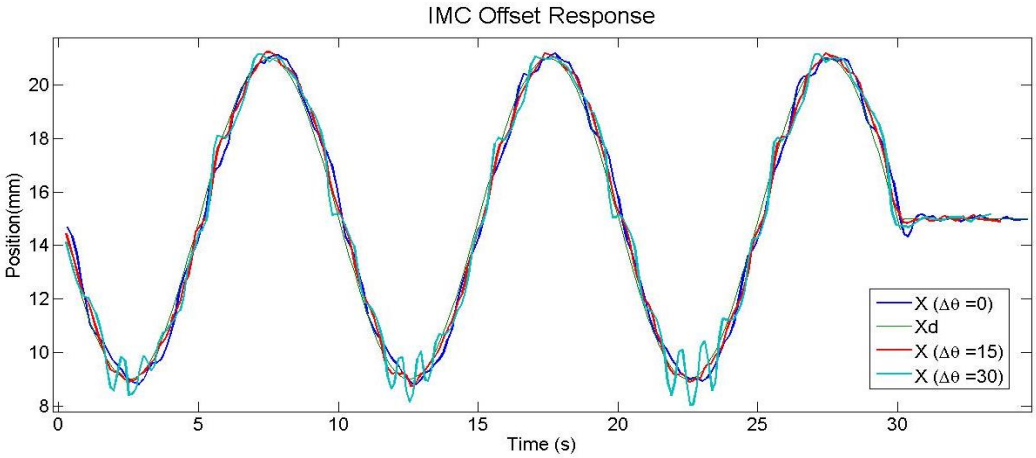


Figure 49 IMC response to a sine wave with plant offset

The IMC is able to follow the sinusoidal path at any offset but at 30^{o} or greater the object becomes less stable and oscillates at certain regions. The other thing to notice is that the controller is able to quickly settle at the 30sec set-point without any offset.

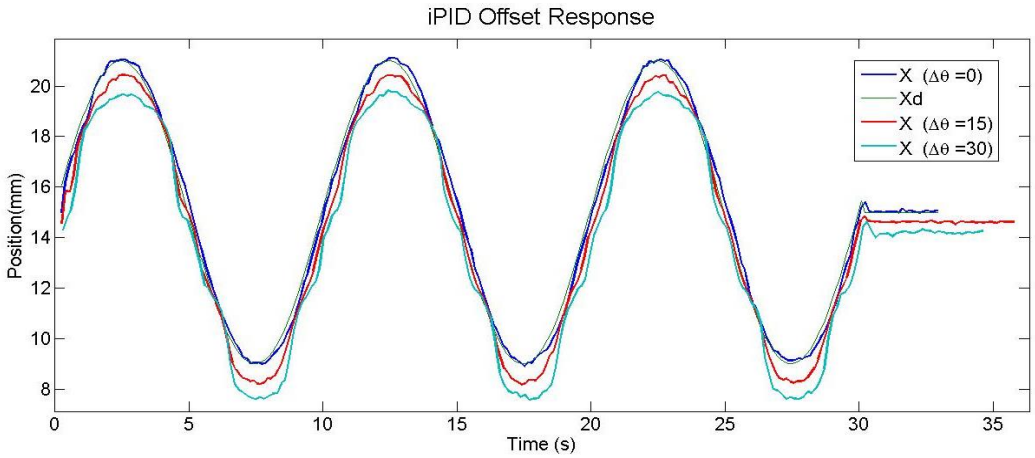


Figure 50 iPID response to a sine wave with plant offset

The iPID controller performs better than the IMC when the plant offset is zero as seen by the close match between the blue and green line. As soon as there is a plant offset the iPID controller is unable to compensate and undershoots the desired position in some regions and overshoots the desired position in others. After the run is complete at 30 seconds the desired set-point is 15mm but the iPID controller has a steady state error when there is plant offset.

Although the effects of the plant offset can best be seen by the comparing the iPID and IMC response to a sine wave trajectory the other controllers are tested as well. Table 5 shows how each of the controllers performed at various speeds travelling to either 25 mm or 6 mm.

Table 5 Controller performance with plant offset

No	Controller	Xo (mm)	Vg (mm/s)	Offset	RMSe
1	IMCv	25	0.5	15	0.0849
2	IMC	6	0.5	15	0.0911
3	IMC	25	0.5	30	0.3178
4	IMC	6	0.5	30	0.3633
5	IMC	6	1	15	0.1140
6	IMC	25	1	15	0.1225
7	IMCv	6	1	30	0.2828
8	IMC	25	1	30	0.3015
9	IMCv	6	2	15	0.1761
10	IMC	25	2	15	0.2145
11	iPID	6	2	30	0.4159
12	IMCv	25	2	30	0.4359

Table 5 is sorted by RMSe and then the velocity goal. The results in Table 5 show that the IMC outperforms the iPID controller at various speeds when there is a plant offset.

4.4 Comparison between ANSYS and experimental results

The ANSYS model is used to see the spacing between the pressure nodes and how the pressure nodes move with changing phase. These results are compared to the experimental results. The first thing that is examined is the spacing between the nodes. The spacing is theoretically half the wavelength which is $\frac{\lambda}{2} = 6.36mm$ when the frequency is 26.7 kHz and the speed of sound is 340 m/s. From Figure 22 the half wavelength of the

ANSYS model is 7mm. To determine the half wavelength for the experimental position the object will be levitated at one position and then the phase will be increased by 360 degrees meaning it will be at the next node position. The data shown in Figure 23is used to find the distance between the experimental nodes which is an average of 7.11mm with a standard deviation of .081mm. These results show that experimental, theoretical, and ANSYS model accurately show the distance between the nodes. The reason the ANSYS model does not precisely match the theoretical model is possibly because a finer mesh would have to be used to capture the exact locations of nodes.

The main interest of this thesis is how the node positions change with phase change. For the ANSYS model Figure 22 shows the location of the node at four different phases but in order see how the node moves a better method of picturing node position must be used. Not only must a better method of picturing node movement be used but also more than four phases must be shown. The phases 0 degrees through 330 degrees with 30 degree increments are used to map how the nodes moves.

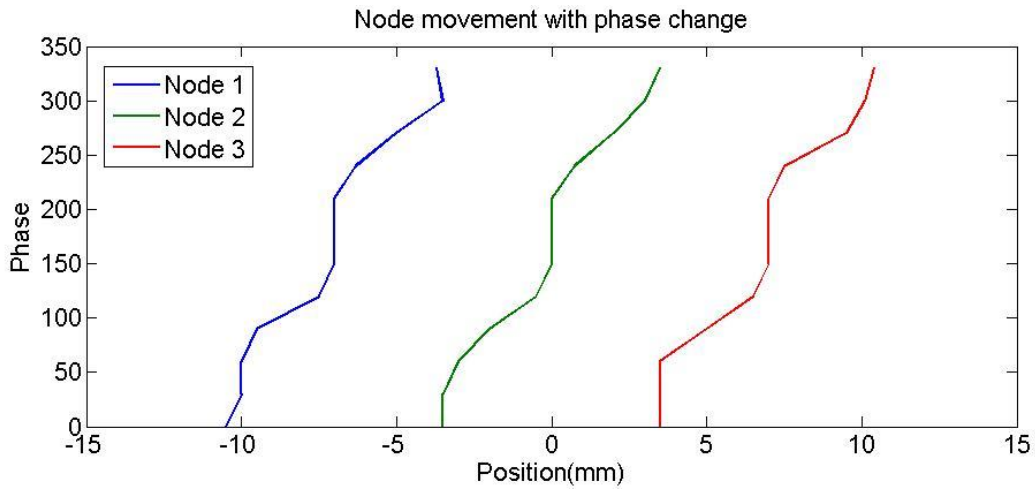


Figure 51 Modeled node positions as phase changes from 0 to 360 degrees

Similar to the experimental results the modeled results show that there are certain regions that are more phase sensitive than others. In the experimental setup it was possible to travel from one side of the gap to the other by moving node to node. To investigate how well the ANSYS model captured the node movement across the whole gap the node positions shown in Figure 51 are compiled into a single string. When calibrating the experimental setup for data collection the phase was shifted on channel 1 of the signal generator until the object was levitating in the center of the gap. Doing the same thing with the ANSYS results leads to a phase shift of 180 degrees to center the node. For the experimental setup the phase shift required to center the object was anywhere from 150 to 165 degrees. After shifting and compiling the ANSYS node position result these results are overlaid with the plant identification curve in Figure 23.

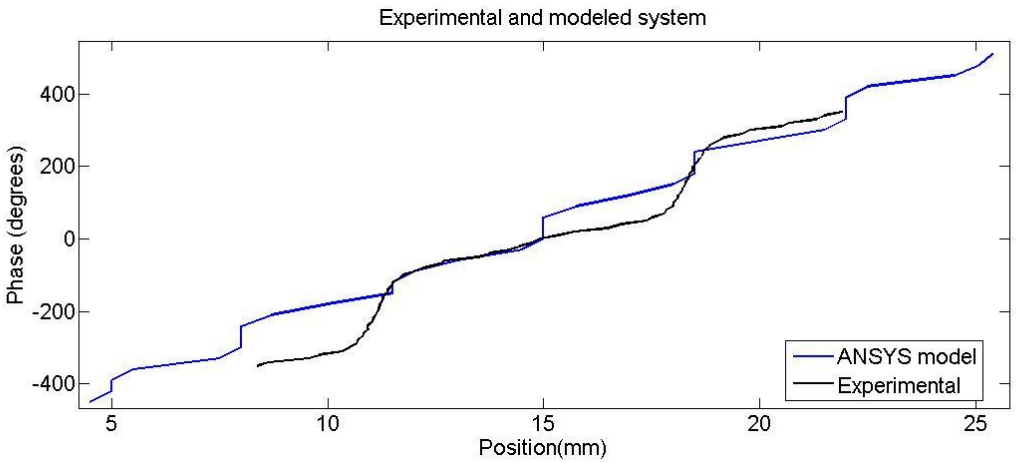


Figure 52 Comparison of modeled and experimental node position

The results of the overlaid comparison between the ANSYS model and experimental results show that while the model results have some characteristics of the experimental system they do not perfectly match. Both sets of results have a similar positive slope and are somewhat step like. One thing to notice is that when there is a step in the experimental results there is also a step in the model. The Experiment shows that every half wavelength there is a step while the model shows that every quarter wavelength there is a step. From the modeled data the RMSe is taken using a modified version of equation (42)—the resulting error value is 7.33 mm. The biggest reason for the mismatch between the model results and the experimental results is that the model does not account for the nonlinear near field effects that occur in this system. ANSYS also makes several assumptions that impact the results. The first is that the fluid is inviscid and does not dissipate energy, and the second is that mean density is uniform throughout the fluid. The physical phenomena being examined is

based on there being regions of low pressure magnitude and high pressure magnitude therefore nodes may have a different air density then anti-nodes.

4.5 Conclusion

The goal of this thesis is to be able to manipulate an object that is levitated using a standing wave. The object is trapped between two actuators allowing for 1-d motion control by adjusting the phase between the actuators. Three main types of controllers were tested including a PID controller, internal model controller and a neural network controller with feedback. A plant model and filter is introduced to improve the controller abilities. Adding a filter to the position sensor resulted in the root-mean-square error going from RMSe=.3031mm when the sensor was unfiltered to RMSe=.0781 mm when a filtering coefficient of α=.5 is used. The first PID controller tested assumed a linear plant and performed poorly. Rebuilding the controller to include a plant inverse model created using a neural network and system input-output data resulted in much improved performance. The gains that worked best were simply a proportional gain of one—making the controller a unity feedback controller with an inverse plant model. The RMSe of the PID controller without the inverse model was RMSe=.5477mm and adding an inverse plant model resulted in a RMSe=.0781mm. Not only was the average error decreased by the inverse plant but the maximum error amplitude was also reduced by a factor of 6. The internal model controller (IMC) and the PID controller with and inverse plant (iPID) performed similarly under ideal conditions but the IMC was better able to handle a plant-model mismatch. When following a

sinusoidal path with a 30^{o} plant offset the iPID controller was shifted 5mm from the desired position while the IMC controller was still able to track the sinusoidal path without a shift.

The third type of controller used was built using only a neural network. This network attempted to combine the plant inverse model with nonlinearfeedback control. Changing the gains on the iPID controller is ineffective because if the gains are tuned for one region of the plant then they will perform poorly in another region because this is a nonlinear system. A neural network controller with feedback (NNc) would be able to provide a gain that is dependent on error, position, and velocity therefore providing improved feedback control. The NNc built off of the plant input-output data is stable but it did not outperform the inverse plant controllers. The NNc was developed offline but it would be possible to improve its performance by implementing it as an adaptive controller. Implementing this controller in an adaptive scheme would allow the neural network to more thoroughly learn the relationships between position, error, phase, and velocity than it originally learned from the offline data set.

A model of the system behavior was required to implement several of the control schemes. Two models were developed—one based on an ANSYS analysis of the system, and the other from a neural network using input-output data from the experimental system. The ANSYS model was able to show the basic phase versus node position relationship but it was unable to accurately model the movement of the node with an object compared to the neural network model. Two reasons for the inaccuracy of the ANSYS model is that it did not incorporate any of the nonlinearities that exist in the real system due to near-field acoustic effects and the object itself was not incorporated into the model. A neural network model of

the inverse plant was shown to be accurate through the effectiveness of the inverse plant controller which had a RMSe=.13 mm when moving at .5 mm/s. A neural network model is what is used for all the control schemes that require an inverse plant model.

When looking at the performance of the controller at various desired velocities it was noticed that the plant performs differently. At higher speeds the object position lags the desired object position leading to decreased performance at higher speeds. An inverse neural network plant model was developed that incorporated this effect and resulted in improved performance at speeds of 2 mm/s. The RMSe performance of the inverse plant that incorporates the speed dependence is RMSe=.17 mm compared to the speed independent inverse plant of RMSe=.232 mm.

While the controllers developed were able to accurately position and move a levitated object there are several improvements that could be made to increase performance. The biggest improvement would come from decreasing the time of the control loop allowing for more data points and control action at higher speeds. The current control loop time is about .17 seconds and is dependent on which controller is being run. One way to decrease the loop speed would is to use a different camera with a higher frame rate and to improve the image processing done to locate the object position. Another way to decrease the control loop would to make the code more efficient by not recording any data for later analysis.

4.6 Future Work

The next step of this experiment is to expand the 1-d motion control into 2-d motion control using three actuators to manipulate object position. 2-d motion control could be implemented using the same technique of phase manipulation but in this case there would be three different phase relations that can be adjusted. To achieve a wider range of object motion physical parameters such as actuator angle or position can manipulated along with the phase parameter. Another area of investigation that would be useful for controlling a levitated object is finding why the plant behaves differently with different object velocities. Understanding what parameters impact this phenomenon would allow for models to be developed that accurately account for this effect. In object manipulation it is important to know the limitations of the system being used therefore further investigation into the impact of object size and density on the ability to control object motion is warranted. Changing the object size or density would impact the dynamics of the plant and must be accounted for. This experiment used a position and velocity dependent feedforward neural network to account for the plant dynamic but if more variables are introduced an alternative to a larger feedforward network with more inputs would be a recurrent neural network scheme.

REFERENCES

[1] Weber, Richard. "Aero-acoustic levitation:A method for containerlessliquie-phase processing at high temperatures ." Review of Scientific Instruments. 65.2 (1994): 456. Print.

[2] Vandaele, Vincent, Pierre Lambert, and Alain Delchambre. "Non-Contact handling in microassembly: Acoustical levitation." *Precision Engineering*. 29. (2005): 491-501. Print.

[3] Park, Joong-Kyoo, and Paul Ro. "Non-contact Manipulation of Light Objects Based on Parameter Modulations of Acoustic Pressure Nodes." (2013): n. page. Print.

[4] Chen, M., C. Tsai, and L. Fu."Design and Control of a 2-Dimensional Electro-Magnetic Suspension Actuator." *IEEE Control Applications*. 1. (2004): 93-98. Print.

[5] Erzincanli, F., J. Sharp, and S. Erhal. "Design and Operational Considerations of a Non-contact Robotic Handling System for Non-Rigid Materials."*Journal of Machine Tools and Manufacturing*.

[6] Biganzoli, F, and G Fantoni. "Contactless electrostatic handling of microcomponents." *Journal of Engineering Manufacture*. 218. (2004): n. page. Print.

[7] Hu, Liang, Hai-Peng Wang, Liu-Hui Li, and Bing-Bo Wei. "Electrostatic Levitation of Plant Seeds and Flower Buds." *Chinese Physics Letters*. 29.6 (2012): n. page. Print.

[8] King, Louis. "On the Acoustic Radiation Pressure on Spheres."*Proceedings of the Royal Society*. 147. (1934): n. page. Print.

[10] Vandaele, Vincent, Alain Delchambre, and Pierre Lambert. "Acoustic wave levitation: Handling of components."*Journal of Applied Physics*. 109. (2011): n. page. Print.

[9] Yosioka, K., and Y. Kawasima."Acoustic Radiation Pressure on a Compressible Sphere." *Acustica*. 5. (1955): n. page. Print.

[11] Kuznetsova, Larisa, and Terence Coakley. "Microparticle concentration in short path length ultrasonic resonators: Roles of radiation pressure and acoustic streaming." *Journal of Acoustic Society of America*. 116.4 (2004): 1956. Print.

[12] Tuziuti, Toru, TeruyukiKozuka, and HidetoMitome. "Measurement of Distribution of Acoustic Radiation Force Perpendicular to Sound Beam Axis." *Journal of Applied Physics*. 38. (1999): 3297-3301. Print.

[13] Nyborg, Wesly. "Radiation Pressure on a Small Rigid Sphere."*Journal of the Acoustical Society of America*. 42. (1967): 947. Print.

[14] Bullinaria, ."Introduction to Neural Computing." *University of Birmingham*. 2012. PDF. 28 Feb 2013. <http://www.cs.bham.ac.uk/~jxb/INC/l1.pdf>.

[15] Huang, Sunan, KokKiong Tan, and KokZueaTang.*Neural Network Control: Theory and Applications*. Philadelphia: Research Studies Press, 2004. Print.

[16] Tejomurtula, Sreenivas, and SubhashKak."Inverse kinematics in robotics using neural networks."*Information Sciences*. 116. (1999): 147-164. Print.

[17] Egmont-Petersen, M., D. Ridder, and H. Handels."Image processing with neural networks-a review." *Pattern Recognition*. 35. (2002): 2279-2301. Print.

[18] Guresen, Erkam, GulgunKayakutlu, and TugrulDaim."Using artificial network models in stock market index prediction." *Expert Systems with Applications*. 38. (2011): 10389-10397. Print.

[19] McCulloch, Warren, and Walter Pitts."A Logical Calculus of the Ideas Immanent in Nervous Activity."*Bulletin of Mathematical Biophysics*. 5. (1943): 115-133. Print.

[20] Rosenblatt, F. "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain."*Psychological Review*. 65.6 (1958): 386-408. Print.

[21] Narendra, K.S., and A.M. Parthasarathy. "Identification and control of dynamical systems using neural networks." *IEEE Transactions on Neural Networks*. 3. (1990): n. page. Print.

[22] Cybenko, G. "Approximation by Superpositions of a Sigmoidal Function." *Mathematic of Control, Signals, and Systems*. 2. (1989): 303. Print.

[23] Hagan, Martin, and Mohammad Menhaj."Training Feedforward Networks with the Marquardt Algorithm." *IEEE Transaction on Neural Networks*. 5.6 (1994): 989-993. Print.

[24] Hunt, K., and D. Sbarbaro. "Neural networks for nonlinear internal model control." *IEEE*. 138.5 (1991): 431-438. Print.

[25] Liu, Yuhu. "Wave Propagation Study Using Finite Element Analysis."MS thesis. University of Illinois , 2005. Print.

[26] Tetko, Igor, David Linvingstone, and Alexander Luik."Neural Network Studies.1.Comparison of Overfitting and Overtraining." *Journal of Chemical Information and Computer Science*. 35. (1995):826-833. Print.

APPENDIX

Appendix A- Neural Network Inverse Plant Model Program

```
% Creates a NN plant using a feedforward network
n=[18]
id=1; %input delays
id2=1;
traintype='trainlm' %training function
plant=feedforwardnet(n,traintype)
plant.sampletime=.25;
plant.inputs{1}.processFcns={};  %sets input process fcn
%plant.inputs{2}.processFcns={};  %sets input process fcn
plant.outputs{2}.processFcns={};  %sets output process fcn
plant.divideFcn='dividerand';
U_plant=combined(:,1);
Y_plant=combined(:,2);
U=mat2cell(U_plant',1,ones(1,length(U_plant)));  %format data
Y=mat2cell(Y_plant',1,ones(1,length(Y_plant)));  %format data

%Training Parameters
plant.trainParam.epochs=1000;              % set epochs to
train
plant.trainParam.min_grad=1e-10;           %set min gradient
plant.trainParam.max_fail=100;
plant.trainParam.mu_max=10*10^15;
plant.divideParam.trainRatio=.7;
plant.divideParam.valRatio=.15;
plant.divideParam.testRatio=.15;
plant=trainlm(plant,U_plant',Y_plant');
Ytest=plant(U_plant');
t=1:length(Ytest);
plot(cm3_15(:,2),Ytest,'r',cm3_15(:,2),Y_plant,'b','LineWidth'
,1.5)
xlabel('Position (mm)','fontsize',18)
ylabel('Phase (degrees)','fontsize',18)
title('Neural Network Validation','fontsize',20)
```

Appendix B- Neural Network Velocity Dependent Plant Model Program

```
%%%%% Internal Model Control%%%%%
% clear all
%load('F:\Research\DATA\3cm\Jan-30\cm3_15')
plant_v=network;

%%%%%%Training sets
% clear U Y x x1 x2 xdesired xdesired1 xdesired2 xdesiredt xt
%  load('F:\Research\Custom_Networks\Plant_Vg\Vg_Plant_data')

plant_v=network;

%%%%%ARCHITECTURE%%%%%%
plant_v.numInputs=2;
plant_v.numLayers=2;


plant_v.biasConnect(1)=1;
plant_v.biasConnect(2)=2;
%plant_v.biasConnect(3)=3;


plant_v.inputConnect(1,1)=1;
plant_v.inputConnect(1,2)=1;


plant_v.layerConnect=[0 0 ; 1  0]


plant_v.outputConnect=[ 0 1];
%%%%%%Example inputs%%%%%
 plant_v.inputs{1}.exampleInput=total(3,:);
 plant_v.inputs{2}.exampleInput=total(2,:);
%%%%%SUB PROPERTIES layer 1%%%%%%%%%

plant_v.inputs{1}.processFcns={'removeconstantrows','mapminmax
'};
plant_v.inputs{1}.size=1;
plant_v.inputs{2}.processFcns={'removeconstantrows','mapminmax
' };
plant_v.inputs{2}.size=1;
plant_v.layers{1}.size=50;
```

```
plant_v.layers{1}.transferFcn='tansig';
plant_v.layers{1}.initFcn='initnw';

%%%%SUB Propreties layer 2%%%%%%%

plant_v.layers{2}.size=1;
plant_v.layers{2}.transferFcn='purelin';
plant_v.layers{2}.initFcn='initnw';
%%%%SUB Propreties layer 2%%%%%%%
% plant_v.layers{3}.size=1;
% plant_v.layers{3}.transferFcn='purelin';
% plant_v.layers{3}.initFcn='initnw';
%%%% Performance Functions%%%%
plant_v.initFcn = 'initlay';
plant_v.performFcn='mse';
plant_v.divideFcn='dividerand';
plant_v.trainFcn='trainlm';
plant_v.divideFcn='dividerand';
plant_v.divideParam.trainRatio=.7;
plant_v.divideParam.valRatio=.3;
plant_v.divideParam.testRatio=0;
plant_v.trainParam.epochs=250;              % set epochs to
train
plant_v.trainParam.min_grad=1e-10;            %set min gradient
plant_v.trainParam.max_fail=200;
plant_v.trainParam.mu_max=10*10^20;
plant_v.plotFcns = {'plotperform','plottrainstate',
'ploterrhist' };
plant_v=init(plant_v);

plant_v=trainlm(plant_v,[total(3,:);total(2,:)],total(1,:));

ytest=plant_v([total(3,:);total(2,:)]);
errnet=ytest-total(1,:);
%errs=U(1,:)-U(3,:);
figure;
plot(1:length(ytest),ytest,1:length(total(1,:)),total(1,:))
phase=-400:1:400;

vg5=.5*ones(1,length(phase));
vg1=1*ones(1,length(phase));
vg2=2*ones(1,length(phase));

p5=plant_v([phase;vg5]);
```

```
p1=plant_v([phase;vg1]);
p2=plant_v([phase;vg2]);
p5n=plant_v([phase;-vg5]);
p1n=plant_v([phase;-vg1]);
p2n=plant_v([phase;-vg2]);

figure(2);
plot(p5,phase,p1,phase,p2,phase,p5n,phase,p1n,phase,p2n,phase)

%%%%%%Inverse Plant%%%%

iplant_v=network;

iplant_v.numInputs=2;
iplant_v.numLayers=2;


iplant_v.biasConnect(1)=1;
iplant_v.biasConnect(2)=2;

iplant_v.inputConnect(1,1)=1;
iplant_v.inputConnect(1,2)=1;

iplant_v.layerConnect=[0 0;1 0]

iplant_v.outputConnect=[0 1];
%%%%% Example inputs
iplant_v.inputs{1}.exampleInput=total(3,:);
iplant_v.inputs{2}.exampleInput=total(2,:);

%%%%%SUB PROPERTIES layer 1%%%%%%%%%

iplant_v.inputs{1}.size=1;
iplant_v.inputs{2}.size=1;
iplant_v.layers{1}.size=40;
iplant_v.inputs{1}.processFcns={'removeconstantrows',
'mapminmax'};
iplant_v.inputs{2}.processFcns={'removeconstantrows',
'mapminmax' };
iplant_v.layers{1}.transferFcn='tansig';
iplant_v.layers{1}.initFcn='initnw';

%%%%SUB Propreties layer 2%%%%%%%
iplant_v.layers{2}.size=1;
```

```
iplant_v.layers{2}.transferFcn='purelin';
iplant_v.layers{2}.initFcn='initnw';

%%%% Performance Functions%%%%
iplant_v.initFcn = 'initlay';
iplant_v.performFcn='mse';
iplant_v.divideFcn='dividerand';
iplant_v.trainFcn='trainlm';
iplant_v.divideFcn='dividerand';
iplant_v.divideParam.trainRatio=.7;
iplant_v.divideParam.valRatio=.15;
iplant_v.divideParam.testRatio=.15;
iplant_v.trainParam.epochs=250;              % set epochs to
train
iplant_v.trainParam.min_grad=1e-10;            %set min
gradient
iplant_v.trainParam.max_fail=100;
iplant_v.trainParam.mu_max=10*10^20;
iplant_v.plotFcns = {'plotperform','plottrainstate',
'ploterrhist' };
iplant_v=init(iplant_v);

iplant_v=trainlm(iplant_v,[total(3,:);total(2,:)],total(1,:));

ytest=iplant_v([total(3,:);total(2,:)]);
x=6:.1:25;
vg5=.5*ones(1,length(x));
vg1=1*ones(1,length(x));
vg2=2*ones(1,length(x));

ip5=iplant_v([x;vg5]);
ip1=iplant_v([x;vg1]);
ip2=iplant_v([x;vg2]);
ip5n=iplant_v([x;-vg5]);
ip1n=iplant_v([x;-vg1]);
ip2n=iplant_v([x;-vg2]);

figure(2);
plot(x,ip5,x,ip1,x,ip2,x,ip5n,x,ip1n,x,ip2n)
%errnet=ytest-total(1,:);
%errs=U(1,:)-U(3,:);
figure;
plot(total(3,1:600),ytest(1:600),total(3,1:600),total(1,1:600)
)
```

Appendix C- NNc Training

```
%%%%% Internal Model Control%%%%%
% clear all
%load('F:\Research\DATA\3cm\Jan-30\cm3_15')
net=network;

load('F:\Research\Custom_Networks\Xd_Vg_e\Open_loop_data')

%%%%%ARCHITECTURE%%%%%%
net.numInputs=3;
net.numLayers=4;


net.biasConnect(1)=1;
net.biasConnect(2)=2;
net.biasConnect(3)=3;
net.biasConnect(4)=4;



net.inputConnect(1,1)=1;
net.inputConnect(1,2)=1;
net.inputConnect(1,3)=1;
net.inputConnect(3,3)=1;


net.layerConnect=[0 0 0 0; 1 0 0 0; 0 1 0 0; 0 0 1 0]

net.outputConnect=[0 0 0 1];
%%%%%%Example inputs%%%%%
net.inputs{1}.exampleInput=total(2,:);
net.inputs{2}.exampleInput=total(3,:)-total(5,:);
net.inputs{3}.exampleInput=[-2 -1 -.5 .5 1 2];
errt=total(3,:)-total(5,:);
%%%%%SUB PROPERTIES layer 1%%%%%%%%%

net.inputs{1}.processFcns={'removeconstantrows', 'mapminmax'};
net.inputs{1}.size=1;
net.inputs{2}.processFcns={'removeconstantrows', 'mapminmax'};
net.inputs{2}.size=1;
net.inputs{3}.processFcns={'removeconstantrows', 'mapminmax'};
net.inputs{3}.size=1;
```

```
net.layers{1}.size=70;
net.layers{1}.transferFcn='tansig';
net.layers{1}.initFcn='initnw';

%%%%SUB Propreties layer 2%%%%%%%
net.layers{2}.size=1;
net.layers{2}.transferFcn='purelin';
net.layers{2}.initFcn='initnw';

%%%%SUB Propreties layer 3%%%%%%%

load('F:\Research\Control_Programs\Plants_vg');
net.layers{3}.size=50;
net.layers{3}.transferFcn='tansig';
net.LW{3,2}=plant_v.IW{1,1};
net.IW{3,3}=plant_v.IW{1,2};
net.b{3}=plant_v.b{1};
net.layerWeights{3,2}.learn=0;
net.inputWeights{3,3}.learn=0;
net.biases{3}.learn=0;

%%%%SUB Propreties output layer%%%%%%%
net.layers{4}.size=1;
net.layers{4}.transferFcn='purelin';
net.LW{4,3}=plant_v.LW{2,1};
net.b{4}=plant_v.b{2};
net.layerWeights{4,3}.learn=0;
net.biases{4}.learn=0;


%%%% Performance Functions%%%%
net.initFcn = 'initlay';
net.performFcn='mse';
net.divideFcn='dividerand';
net.trainFcn='trainlm';
net.divideFcn='dividerand';
net.divideParam.trainRatio=.8;
net.divideParam.valRatio=.2;
net.divideParam.testRatio=0;
net.trainParam.epochs=500;              % set epochs to train
net.trainParam.min_grad=1e-10;            %set min gradient
net.trainParam.max_fail=100;
net.trainParam.mu_max=10*10^20;
```

```
net.plotFcns = {'plotperform','plottrainstate', 'ploterrhist'
};
net=init(net);

net=trainlm(net,[total(2,:);errt;total(4,:)],total(2,:));

ytest=net([total(2,:);errt;total(4,:)]);
errnet=ytest-total(2,:);
errs=total(3,:)-total(5,:);
figure;
plot(total(3,:),errs,total(3,:),errnet)
figure;
plot(1:length(ytest),ytest,1:length(total(3,:)),total(3,:));
%%%%%%Control Network%%%%
figure;
plot(1:length(errs),errs,1:length(errnet),errnet)

NNc=network;

NNc.numInputs=3;
NNc.numLayers=2;

NNc.biasConnect(1)=1;
NNc.biasConnect(2)=2;

NNc.inputConnect(1,1)=1;
NNc.inputConnect(1,2)=1;
NNc.inputConnect(1,3)=1;

NNc.layerConnect=[0 0;1 0]

NNc.outputConnect=[0 1];
%%%%% Example inputs
NNc.inputs{1}.exampleInput=total(2,:);
NNc.inputs{2}.exampleInput=total(3,:)-total(5,:);
NNc.inputs{3}.exampleInput=[-2 -1 -.5 .5 1 2];
%%%%%SUB PROPERTIES layer 1%%%%%%%%%

NNc.inputs{1}.size=1;
NNc.inputs{2}.size=1;
NNc.inputs{3}.size=1;
NNc.layers{1}.size=70;
NNc.inputs{1}.processFcns={'removeconstantrows', 'mapminmax'};
```

```
NNc.inputs{2}.processFcns={'removeconstantrows', 'mapminmax'
};
NNc.inputs{3}.processFcns={'removeconstantrows', 'mapminmax'
};
NNc.layers{1}.transferFcn='tansig';
NNc.IW{1,1}=net.IW{1,1};
NNc.inputWeights{1,1}.learn=0;
NNc.IW{1,2}=net.IW{1,2};
NNc.inputWeights{1,2}.learn=0;
NNc.IW{1,3}=net.IW{1,3};
NNc.inputWeights{1,2}.learn=0;
NNc.b{1}=net.b{1};
NNc.biases{1}.learn=0;

%%%%SUB Propreties layer 2%%%%%%%
NNc.layers{2}.size=1;
NNc.layers{2}.transferFcn='purelin';
NNc.LW{2,1}=net.LW{2,1};
NNc.layerWeights{2,1}.learn=0;
NNc.b{2}=net.b{2};
NNc.biases{2}.learn=0;
```

Appendix D- IMC Implementation

NOTE: This is the controls portion of the GUI not the entire GUI program

```
if handles.control==1;
    handles.error_pos(n)=handles.target_pos-handles.real_x(n);

switch handles.motion
 case 1
   errf=handles.real_x(n)-handles.delta(n-1);

   handles.plant_error(n)=errf*(handles.alpha)+handles.plant_e
   rror(n-1)*(1-handles.alpha);

   handles.xd(n)=handles.real_x(3)+handles.velocity_goal*(hand
   les.real_time(n)+.141);

   handles.delta(n)=handles.real_x(3)+handles.velocity_goal*(h
   andles.real_time(n)+.141)-handles.plant_error(n);
 if handles.real_time(n) >= stopper

   handles.xd(n)=handles.target_pos;

   handles.delta(n)=handles.target_pos-handles.plant_error(n);
 end

 if   handles.error_pos(n) >.05  || handles.error_pos(n) <-.05

handles.real_b_pha(n)=round(handles.iplant(handles.delta(n)));


handles.b_pha(n)=handles.real_b_pha(n);
  if handles.real_b_pha(n) >360

     rd=floor(handles.real_b_pha(n)/(360));

     handles.b_pha(n)=handles.real_b_pha(n)-360*rd;
  end

  if handles.real_b_pha(n) <=-360

     rd=ceil(handles.real_b_pha(n)/(360));
```

```
      handles.b_pha(n)=handles.real_b_pha(n)-360*rd;
   end

      fprintf(handles.g,'SOURCE2:PHASE:ADJUST
      %sDEG',num2str(handles.b_pha(n)))

      set(handles.fg_b_pha_edit,'String',num2str(handles.real_b
      _pha(n)));

   else

      handles.b_pha(n)=handles.b_pha(n-1);

      handles.real_b_pha(n)=handles.real_b_pha(n-1);

      fprintf(handles.g,'SOURCE2:PHASE:ADJUST
      %sDEG',num2str(handles.b_pha(n)))

      set(handles.fg_b_pha_edit,'String',num2str(handles.real_b
      _pha(n)));

 end
case 2
   errf=handles.real_x(n)-handles.delta(n-1);

handles.plant_error(n)=errf*handles.alpha+handles.plant_error(
n-1)*(1-handles.alpha) ;

handles.xd(n)=handles.target_pos-
handles.amplitude*sin(2*pi*handles.freq*(handles.real_time(n)+
.141));
 handles.delta(n)=handles.target_pos-
handles.amplitude*sin(2*pi*handles.freq*handles.real_time(n))-
handles.plant_error(n);

if handles.real_time(n) >= stopper2

handles.xd(n)=handles.target_pos+handles.amplitude*sin(2*pi*ha
ndles.freq*stopper2);

handles.delta(n)=handles.target_pos+handles.amplitude*sin(2*pi
*handles.freq*stopper2)-handles.plant_error(n);
 end
```

```
     handles.real_b_pha(n)=round(handles.iplant(handles.delta(
     n)));

     handles.b_pha(n)=handles.real_b_pha(n);
if handles.real_b_pha(n) >360

rd=floor(handles.real_b_pha(n)/(360));

handles.b_pha(n)=handles.real_b_pha(n)-360*rd;
end

if handles.real_b_pha(n) <=-360

rd=ceil(handles.real_b_pha(n)/(360));

handles.b_pha(n)=handles.real_b_pha(n)-360*rd;
end

fprintf(handles.g,'SOURCE2:PHASE:ADJUST
%sDEG',num2str(handles.b_pha(n)))

set(handles.fg_b_pha_edit,'String',num2str(handles.real_b_pha(
n)));

end
end
end
```

Appendix E- iPID and PID Implementation

NOTE: This is the controls portion of the GUI not the entire GUI program

```
if handles.control==1;
   handles.error_pos(n)=handles.target_pos-handles.real_x(n);

  switch handles.motion
  case 1


handles.xd(n)=handles.real_x(4)+handles.velocity_goal*(handles
.real_time(n)+.129);

   if handles.real_time(n) >= stopper
      handles.xd(n)=handles.target_pos;
   end

   if   handles.error_pos(n) >.03

    handles.error_xd(n)=handles.xd(n-1)-xf(n-2);

    P_dpha=handles.error_xd(n)*handles.P;
    I_dpha=sum(handles.error_xd(4:n))*handles.I;
    D_dpha=((handles.error_xd(n)-handles.error_xd(n-2))/...
     (handles.real_time(n)-handles.real_time(n-2)))*handles.D;

    handles.delta(n)=P_dpha+I_dpha+D_dpha;

handles.real_b_pha(n)=round(handles.plant(handles.xd(n)+handle
s.delta(n)));
    handles.b_pha(n)=handles.real_b_pha(n);
    if handles.real_b_pha(n) >360
      rd=floor(handles.real_b_pha(n)/(360));
      handles.b_pha(n)=handles.real_b_pha(n)-360*rd;
    end

    if handles.real_b_pha(n) <=-360
     rd=ceil(handles.real_b_pha(n)/(360));
     handles.b_pha(n)=handles.real_b_pha(n)-360*rd;
    end
     fprintf(handles.g,'SOURCE2:PHASE:ADJUST
%sDEG',num2str(handles.b_pha(n)))
```

```
set(handles.fg_b_pha_edit,'String',num2str(handles.real_b_pha(
n)));

    elseif handles.error_pos(n) <-.03
     handles.error_xd(n)=handles.xd(n-1)-xf(n-2);

     P_dpha=handles.error_xd(n)*handles.P;
     I_dpha=sum(handles.error_xd(4:n))*handles.I;
     D_dpha=((handles.error_xd(n)-handles.error_xd(n-2))/...
       (handles.real_time(n)-handles.real_time(n-
2)))*handles.D;

      handles.delta(n)=P_dpha+I_dpha+D_dpha;


handles.real_b_pha(n)=round(handles.plant(handles.xd(n)+handle
s.delta(n)));
     handles.b_pha(n)=handles.real_b_pha(n);
     if handles.real_b_pha(n) >=360
       rd=floor(handles.real_b_pha(n)/(360));
       handles.b_pha(n)=handles.real_b_pha(n)-360*rd;
     end

      if handles.real_b_pha(n) <=-360
       rd=ceil(handles.real_b_pha(n)/(360));
       handles.b_pha(n)=handles.real_b_pha(n)-360*rd;
      end
       fprintf(handles.g,'SOURCE2:PHASE:ADJUST
%sDEG',num2str(handles.b_pha(n)))

set(handles.fg_b_pha_edit,'String',num2str(handles.real_b_pha(
n)));

   else
     handles.b_pha(n)=handles.b_pha(n-1);
     handles.real_b_pha(n)=handles.real_b_pha(n-1);
     fprintf(handles.g,'SOURCE2:PHASE:ADJUST
%sDEG',num2str(handles.b_pha(n)))

set(handles.fg_b_pha_edit,'String',num2str(handles.real_b_pha(
n)));

case 2
```

```
handles.xd(n)=handles.target_pos+handles.amplitude*sin(2*pi*ha
ndles.freq*(handles.real_time(n)+.13));
  if handles.real_time(n) >= stopper2

handles.xd(n)=handles.target_pos+handles.amplitude*sin(2*pi*ha
ndles.freq*stopper2);
  end

    handles.error_xd(n)=handles.xd(n-1)-xf(n-2);

    P_dpha=handles.error_xd(n)*handles.P;
    I_dpha=sum(handles.error_xd(4:n))*handles.I;
    D_dpha=((handles.error_xd(n)-handles.error_xd(n-2))/...
     (handles.real_time(n)-handles.real_time(n-2)))*handles.D;

    handles.delta(n)=P_dpha+I_dpha+D_dpha;

handles.real_b_pha(n)=round(handles.plant(handles.xd(n))+handl
es.delta(n));
    handles.b_pha(n)=handles.real_b_pha(n);

   if handles.real_b_pha(n) >360
    rd=floor(handles.real_b_pha(n)/(360));
    handles.b_pha(n)=handles.real_b_pha(n)-360*rd;
   end

   if handles.real_b_pha(n) <=-360
   rd=ceil(handles.real_b_pha(n)/(360));
   handles.b_pha(n)=handles.real_b_pha(n)-360*rd;
   end
    fprintf(handles.g,'SOURCE2:PHASE:ADJUST
%sDEG',num2str(handles.b_pha(n)))

set(handles.fg_b_pha_edit,'String',num2str(handles.real_b_pha(
n)));

case 3


handles.xd(n)=handles.real_x(4)+handles.velocity_goal*(handles
.real_time(n)+.129);

 if handles.real_time(n) >= stopper
```

```
  handles.xd(n)=handles.target_pos;
 end

 if   handles.error_pos(n) >.03 || handles.error_pos(n) <-.03

  handles.error_xd(n)=handles.xd(n-1)-xf(n-2);

  P_dpha=handles.error_xd(n)*handles.P;
  I_dpha=sum(handles.error_xd(4:n))*handles.I;
  D_dpha=((handles.error_xd(n)-handles.error_xd(n-2))/...
   handles.real_time(n)-handles.real_time(n-2)))*handles.D;

  handles.delta(n)=P_dpha+I_dpha+D_dpha;
  handles.real_b_pha(n)=round(44.84*handles.xd(n)-
679+handles.delta(n));
  handles.b_pha(n)=handles.real_b_pha(n);
  if handles.real_b_pha(n) >360
   rd=floor(handles.real_b_pha(n)/(360));
   handles.b_pha(n)=handles.real_b_pha(n)-360*rd;
  end

  if handles.real_b_pha(n) <=-360
   rd=ceil(handles.real_b_pha(n)/(360));
   handles.b_pha(n)=handles.real_b_pha(n)-360*rd;
  end
   fprintf(handles.g,'SOURCE2:PHASE:ADJUST
%sDEG',num2str(handles.b_pha(n)))

set(handles.fg_b_pha_edit,'String',num2str(handles.real_b_pha(
n)));

  else
   handles.b_pha(n)=handles.b_pha(n-1);
   handles.real_b_pha(n)=handles.real_b_pha(n-1);
   fprintf(handles.g,'SOURCE2:PHASE:ADJUST
%sDEG',num2str(handles.b_pha(n)))

set(handles.fg_b_pha_edit,'String',num2str(handles.real_b_pha(
n)));
  end
  otherwise
  end
  end
  end
```

Appendix F- NNc Implementation

NOTE: This is the controls portion of the GUI not the entire GUI program

```
if handles.control==1;
 handles.error_pos(n)=handles.target_pos-handles.real_x(n);

switch handles.motion
case 1

handles.xf(n)=handles.real_x(n)*handles.alpha+(1-
handles.alpha)*handles.xf(n-1);

handles.xd(n)=handles.real_x(3)+handles.velocity_goal*(handles
.real_time(n)+.166);
handles.error(n)=handles.xd(n-1)-handles.real_x(n);

if handles.error_pos(n) >=.03 || handles.eror_pos(n) <= -.03;
   if handles.real_time(n) >= stopper

     handles.xd(n)=handles.target_pos;
   end
  U=[handles.xd(n);handles.xd(n-1)-
handles.xf(n);handles.velocity_goal];

handles.real_b_pha(n)=round(handles.NNc(U));

handles.b_pha(n)=handles.real_b_pha(n);
if handles.real_b_pha(n) >360

rd=floor(handles.real_b_pha(n)/(360));

handles.b_pha(n)=handles.real_b_pha(n)-360*rd;
end

if handles.real_b_pha(n) <=-360

rd=ceil(handles.real_b_pha(n)/(360));

handles.b_pha(n)=handles.real_b_pha(n)-360*rd;
end
```

```
else
handles.real_b_pha(n)=handles.real_b_pha(n-1)
handles.b_pha(n)=handles.real_b_pha(n);
end

fprintf(handles.g,'SOURCE2:PHASE:ADJUST
%sDEG',num2str(handles.b_pha(n)))

set(handles.fg_b_pha_edit,'String',num2str(handles.real_b_pha(
n)));

otherwise
end
end
end
```

Appendix G- List of Experimental Results

Table 6 Experimental results for linear motion

Run #	Controller	Xo mm	Vg mm/s	Filter	P	I	D	$\Delta\theta$	RMSe	Date
6	PIDv	25	0.5	0.5	1	0	0	0	0.07416	4-Apr
7	PID	6	0.5	0.5	1	0	0	0	0.07746	3-Apr
7	IMC	6	0.5	0.5	n/a	n/a	n/a	0	0.07746	3-Apr
5	IMCv	6	0.5	0.5	n/a	n/a	n/a	0	0.07746	4-Apr
1	PID	6	0.5	0.5	1	0	0	0	0.0781	14-Mar
4	IMCv	25	0.5	0.5	n/a	n/a	n/a	0	0.0781	4-Apr
5	PID	6	0.5	0.5	1	0	0	0	0.08124	20-Mar
0	PID	25	0.5	0.5	1	0	0	0	0.08367	14-Mar
16	IMCv	25	0.5	0.5	n/a	n/a	n/a	15	0.08485	4-Apr
6	IMC	25	0.5	0.5	n/a	n/a	n/a	0	0.08602	3-Apr
5	IMC	6	0.5	0.5	n/a	n/a	n/a	0	0.08718	14-Mar
8	PID	25	0.5	0.5	1	0	0	0	0.08718	3-Apr
0	IMC	25	0.5	0.7	n/a	n/a	n/a	0	0.08832	20-Mar
4	IMC	25	0.5	0.5	n/a	n/a	n/a	0	0.09	14-Mar
6	PID	25	0.5	0.5	1	0	0	0	0.09	20-Mar
13	IMC	6	0.5	0.5	n/a	n/a	n/a	15	0.0911	3-Apr
12	IMC	25	0.5	0.5	n/a	n/a	n/a	15	0.09274	3-Apr
1	PID	6	0.5	0.8	1	0	0	0	0.09487	14-Mar
17	IMCv	6	0.5	0.5	n/a	n/a	n/a	15	0.09592	4-Apr
7	PIDv	6	0.5	0.5	1	0	0	0	0.09644	4-Apr
0	PID	25	0.5	0.8	1	0	0	0	0.10488	14-Mar
3	PID	6	0.5	0.5	1	0.1	0	0	0.11832	14-Mar
1	IMC	6	0.5	0.7	n/a	n/a	n/a	0	0.12042	20-Mar
0	PID	25	0.5	0.5	0	0	0	0	0.13038	22-Mar
1	PID	6	0.5	0.5	0	0	0	0	0.13038	22-Mar
2	PID	25	0.5	0.5	1	0.1	0	0	0.13077	14-Mar
0	IMC1	25	0.5	0.6	n/a	n/a	n/a	0	0.13784	20-Mar
1	IMC	6	0.5	0.7	n/a	n/a	n/a	0	0.14491	14-Mar
5	IMCv	6	0.5	0.8	n/a	n/a	n/a	0	0.17321	1-Apr
0	PIDv	25	0.5	0.5	0	0	0	0	0.17321	24-Apr

Table 6 Continued

1	IMC1	6	0.5	0.6	n/a	n/a	n/a	0	0.17889	20-Mar
0	IMC	25	0.5	0.7	n/a	n/a	n/a	0	0.18166	14-Mar
3	PID	10	0.5	0.8	1	0.3	0	0	0.18166	14-Mar
8	NNc	25	0.5	0.5	n/a	n/a	n/a	0	0.18974	15-Apr
1	PIDv	6	0.5	0.5	0	0	0	0	0.18974	24-Apr
9	NNc	6	0.5	0.5	n/a	n/a	n/a	0	0.19494	15-Apr
7	PIDv	25	0.5	0.8	1	0	0	0	0.19748	1-Apr
10	NNc	25	0.5	0.5	n/a	n/a	n/a	0	0.19748	15-Apr
6	PIDv	6	0.5	0.8	1	0	0	0	0.20248	1-Apr
11	NNc	6	0.5	0.5	n/a	n/a	n/a	0	0.20248	15-Apr
2	PID	25	0.5	0.8	1	0.2	0	0	0.21679	14-Mar
13	PIDv	6	0.5	0.5	1	0	0	15	0.22136	4-Apr
4	IMCv	25	0.5	0.8	n/a	n/a	n/a	0	0.22804	1-Apr
0	PIDv	6	0.5	0.8	0	0	0	0	0.23452	1-Apr
14	PID	6	0.5	0.5	1	0	0	15	0.26058	3-Apr
13	PID	25	0.5	0.5	1	0	0	15	0.28844	3-Apr
18	IMC	25	0.5	0.5	n/a	n/a	n/a	30	0.3178	3-Apr
22	IMCv	25	0.5	0.5	n/a	n/a	n/a	30	0.32558	4-Apr
12	PIDv	25	0.5	0.5	1	0	0	15	0.33015	4-Apr
1	PIDv	25	0.5	0.8	0	0	0	0	0.34641	1-Apr
19	IMC	6	0.5	0.5	n/a	n/a	n/a	30	0.36332	3-Apr
23	IMCv	6	0.5	0.5	n/a	n/a	n/a	30	0.36332	4-Apr
19	PIDv	6	0.5	0.5	1	0	0	30	0.52154	4-Apr
17	PIDs	25	0.5	0.5	35	0.5	0	0	0.54772	11-Apr
19	PIDs	6	0.5	0.5	35	0.5	0	0	0.56569	11-Apr
20	PID	6	0.5	0.5	1	0	0	30	0.57359	3-Apr
19	PID	25	0.5	0.5	1	0	0	30	0.61074	3-Apr
18	PIDv	25	0.5	0.5	1	0	0	30	0.67157	4-Apr
0	PIDs	25	0.5	0.7	1	0.3	0	0	0.8775	9-Apr
2	PIDs	25	0.5	0.7	1.2	0.1	0	0	0.88882	9-Apr
1	PIDs	6	0.5	0.7	1	0.3	0	0	0.96954	9-Apr
3	PIDs	6	0.5	0.7	1.2	0.1	0	0	0.98995	9-Apr
2	IMCv	25	1	0.5	n/a	n/a	n/a	0	0.07071	4-Apr
22	PID	6	1	0.5	1	0	0	0	0.0781	3-Apr
21	IMC	6	1	0.5	n/a	n/a	n/a	0	0.07937	3-Apr
3	IMCv	6	1	0.5	n/a	n/a	n/a	0	0.08367	4-Apr

Table 6 Continued

10	PID	6	1	0.5	1	0	0	0	0.08944	20-Mar
21	PID	25	1	0.5	1	0	0	0	0.0911	3-Apr
5	PID	6	1	0.5	1	0	0	0	0.09487	3-Apr
5	IMCv	6	1	0.5	n/a	n/a	n/a	0	0.09487	3-Apr
4	PIDv	25	1	0.5	1	0	0	0	0.09487	4-Apr
5	PID	6	1	0.5	1	0	0	0	0.09644	14-Mar
4	IMC	25	1	0.5	n/a	n/a	n/a	0	0.09849	3-Apr
2	IMC	25	1	0.7	n/a	n/a	n/a	0	0.10392	20-Mar
4	PID	25	1	0.5	1	0	0	0	0.10488	14-Mar
9	PID	25	1	0.5	1	0	0	0	0.10488	20-Mar
5	IMC	6	1	0.5	n/a	n/a	n/a	0	0.10488	3-Apr
4	IMCv	25	1	0.5	n/a	n/a	n/a	0	0.10488	3-Apr
20	IMC	25	1	0.5	n/a	n/a	n/a	0	0.10954	3-Apr
3	IMC	6	1	0.7	n/a	n/a	n/a	0	0.11091	20-Mar
6	PID	25	1	0.5	1	0	0	0	0.11402	3-Apr
11	IMC	6	1	0.5	n/a	n/a	n/a	15	0.11402	3-Apr
15	IMCv	6	1	0.5	n/a	n/a	n/a	15	0.11874	4-Apr
1	IMC	6	1	0.5	n/a	n/a	n/a	0	0.12247	14-Mar
3	PIDv	6	1	0.7	1	0	0	0	0.12247	1-Apr
10	IMC	25	1	0.5	n/a	n/a	n/a	15	0.12247	3-Apr
8	PID	25	1	0.5	1.1	0	0	0	0.12649	14-Mar
5	PIDv	6	1	0.5	1	0	0	0	0.13038	4-Apr
2	PIDv	25	1	0.7	1	0	0	0	0.13416	1-Apr
2	IMC1	25	1	0.6	n/a	n/a	n/a	0	0.13784	20-Mar
0	PID	25	1	0.5	0	0	0	0	0.14832	20-Mar
2	PID	25	1	0.5	0	0	0	0	0.14832	22-Mar
2	IMCv	25	1	0.7	n/a	n/a	n/a	0	0.14832	1-Apr
14	IMCv	25	1	0.5	n/a	n/a	n/a	15	0.14866	4-Apr
0	IMC	25	1	0.5	n/a	n/a	n/a	0	0.15166	14-Mar
3	IMC	6	1	0.7	n/a	n/a	n/a	0	0.15166	14-Mar
3	PID	6	1	0.5	0	0	0	0	0.15811	22-Mar
2	IMCv	25	1	0.8	n/a	n/a	n/a	0	0.15811	1-Apr
2	PID	25	1	0.5	0	0	0	0	0.16125	20-Mar
3	IMC1	6	1	0.6	n/a	n/a	n/a	0	0.16125	20-Mar
2	IMC	25	1	0.7	n/a	n/a	n/a	0	0.17321	14-Mar
3	PIDv	6	1	0.5	0	0	0	0	0.17889	24-Apr

Table 6 Continued

7	PID	6	1	0.5	1.1	0.1	0	0	0.18708	14-Mar
3	IMCv	6	1	0.8	n/a	n/a	n/a	0	0.18708	1-Apr
6	PID	25	1	0.5	1	0.1	0	0	0.19494	14-Mar
3	PID	6	1	0.5	0	0	0	0	0.19494	20-Mar
3	IMCv	6	1	0.7	n/a	n/a	n/a	0	0.19494	1-Apr
7	NNc	6	1	0.5	n/a	n/a	n/a	0	0.19494	15-Apr
1	PID	6	1	0.5	0	0	0	0	0.2	20-Mar
5	NNc	6	1	0.5	n/a	n/a	n/a	0	0.2	15-Apr
2	PIDv	25	1	0.5	0	0	0	0	0.20494	24-Apr
11	PIDv	6	1	0.5	1	0	0	15	0.20976	4-Apr
6	NNc	25	1	0.5	n/a	n/a	n/a	0	0.22136	15-Apr
2	PIDv	6	1	0.8	0	0	0	0	0.22804	1-Apr
8	PIDv	6	1	0.8	1	0	0	0	0.251	1-Apr
12	PID	6	1	0.5	1	0	0	15	0.2569	3-Apr
21	IMCv	6	1	0.5	n/a	n/a	n/a	30	0.28284	4-Apr
4	NNc	25	1	0.5	n/a	n/a	n/a	0	0.28636	15-Apr
9	PIDv	25	1	0.8	1	0	0	0	0.29326	1-Apr
16	IMC	25	1	0.5	n/a	n/a	n/a	30	0.3015	3-Apr
11	PID	25	1	0.5	1	0	0	15	0.30496	3-Apr
17	IMC	6	1	0.5	n/a	n/a	n/a	30	0.35637	3-Apr
20	IMCv	25	1	0.5	n/a	n/a	n/a	30	0.36332	4-Apr
10	PIDv	25	1	0.5	1	0	0	15	0.37417	4-Apr
23	NNc	25	1	0.5	n/a	n/a	n/a	15	0.41231	15-Apr
17	PIDv	6	1	0.5	1	0	0	30	0.45935	4-Apr
3	PIDv	25	1	0.8	0	0	0	0	0.499	1-Apr
10	PIDs	6	1	0.5	48	0.3	0	0	0.5	11-Apr
18	PID	6	1	0.5	1	0	0	30	0.501	3-Apr
9	PIDs	25	1	0.5	24	0.3	0	0	0.52915	11-Apr
13	PIDs	25	1	0.5	35	0.5	0	0	0.5831	11-Apr
14	PIDs	6	1	0.5	35	0.5	0	0	0.5831	11-Apr
12	PIDs	6	1	0.5	30	0.5	0	0	0.59161	11-Apr
17	PID	25	1	0.5	1	0	0	30	0.59749	3-Apr
11	PIDs	25	1	0.5	30	0.5	0	0	0.60828	11-Apr
7	PIDs	25	1	0.5	12	0.3	0	0	0.63246	11-Apr
8	PIDs	6	1	0.5	24	0.3	0	0	0.65574	11-Apr
16	PIDv	25	1	0.5	1	0	0	30	0.68264	4-Apr

Table 6 Continued

21	NNc	25	1	0.5	n/a	n/a	n/a	15	0.8	15-Apr
5	PIDs	25	1	0.5	6	0.3	0	0	0.80623	11-Apr
6	PIDs	6	1	0.5	12	0.3	0	0	0.82462	11-Apr
3	PIDs	25	1	0.5	6	0.3	0	0	0.8775	11-Apr
4	PIDs	6	1	0.5	6	0.3	0	0	0.89889	11-Apr
6	PIDs	25	1	0.5	1	0	0	0	0.93808	9-Apr
14	PIDs	25	1	0.5	1	0.1	0	0	0.95394	9-Apr
13	PIDs	6	1	0.5	0.7	0.1	0	0	0.99499	9-Apr
4	PIDs	25	1	0.5	1.2	0.1	0	0	1	9-Apr
5	PIDs	6	1	0.5	1.2	0.1	0	0	1.00499	9-Apr
7	PIDs	6	1	0.5	1	0	0	0	1.01489	9-Apr
1	PIDv	6	2	0.7	1	0	0	0	0.10954	4-Apr
3	IMC	6	2	0.5	n/a	n/a	n/a	0	0.10954	4-Apr
1	IMC	6	2	0.7	n/a	n/a	n/a	0	0.11402	4-Apr
3	IMC	6	2	0.5	n/a	n/a	n/a	0	0.11832	14-Mar
11	PID	6	2	0.5	1.1	0	0	0	0.11832	22-Mar
1	IMCv	6	2	0.5	n/a	n/a	n/a	0	0.12247	4-Apr
2	PID	6	2	0.7	1	0	0	0	0.1261	3-Apr
7	IMC	6	2	0.7	n/a	n/a	n/a	0	0.12649	20-Mar
13	PID	25	2	0.5	1.1	0	0	0	0.13038	20-Mar
5	IMC	6	2	0.7	n/a	n/a	n/a	0	0.13038	20-Mar
10	PID	25	2	0.5	1.1	0	0	0	0.13038	22-Mar
3	IMC	6	2	0.7	n/a	n/a	n/a	0	0.13038	3-Apr
11	PID	25	2	0.5	1	0	0	0	0.13416	20-Mar
1	IMC	6	2	0.5	n/a	n/a	n/a	0	0.13416	3-Apr
0	IMCv	25	2	0.5	n/a	n/a	n/a	0	0.13416	4-Apr
2	IMC	25	2	0.5	n/a	n/a	n/a	0	0.13784	4-Apr
0	PID	6	2	0.5	1	0	0	0	0.14491	3-Apr
1	IMCv	6	2	0.5	n/a	n/a	n/a	0	0.14491	3-Apr
9	PID	6	2	0.5	1	0	0	0	0.14832	14-Mar
6	IMC	25	2	0.7	n/a	n/a	n/a	0	0.14832	20-Mar
0	PIDv	25	2	0.7	1	0	0	0	0.14832	4-Apr
10	PID	25	2	0.5	1	0	0	0	0.15166	14-Mar
2	IMC	25	2	0.5	n/a	n/a	n/a	0	0.15492	14-Mar
12	PID	6	2	0.5	1	0	0	0	0.15492	20-Mar
4	IMC	25	2	0.7	n/a	n/a	n/a	0	0.15492	20-Mar

Table 6 Continued

2	IMCv	25	2	0.7	n/a	n/a	n/a	0	0.15492	3-Apr
0	IMC	25	2	0.5	n/a	n/a	n/a	0	0.16125	3-Apr
16	PID	6	2	0.5	1.1	0.05	0	0	0.16432	20-Mar
14	PID	6	2	0.5	1	0.05	0	0	0.16733	20-Mar
1	PID	25	2	0.5	1	0	0	0	0.16733	3-Apr
7	PID	6	2	0.8	1	0.1	0	0	0.17029	14-Mar
15	PID	25	2	0.5	1	0.05	0	0	0.17029	20-Mar
1	PIDv	6	2	0.7	1	0	0	0	0.17029	1-Apr
1	IMCv	6	2	0.8	n/a	n/a	n/a	0	0.17321	1-Apr
4	PID	25	2	0.7	1	0	0	0	0.17321	3-Apr
2	IMC	25	2	0.7	n/a	n/a	n/a	0	0.17321	3-Apr
4	PIDv	25	2	0.5	0	0	0	0	0.17321	24-Apr
1	IMCv	6	2	0.7	n/a	n/a	n/a	0	0.17607	1-Apr
0	IMCv	25	2	0.5	n/a	n/a	n/a	0	0.17607	3-Apr
13	IMCv	6	2	0.5	n/a	n/a	n/a	15	0.17607	4-Apr
4	IMC1	25	2	0.6	n/a	n/a	n/a	0	0.18166	20-Mar
3	PIDv	6	2	0.5	1	0	0	0	0.18439	4-Apr
0	IMC	25	2	0.7	n/a	n/a	n/a	0	0.18708	4-Apr
4	IMCv	25	2	0.7	n/a	n/a	n/a	0	0.18974	1-Apr
5	IMCv	6	2	0.7	n/a	n/a	n/a	0	0.18974	1-Apr
3	IMCv	6	2	0.7	n/a	n/a	n/a	0	0.18974	3-Apr
5	PIDv	6	2	0.7	1	0	0	0	0.19494	1-Apr
4	PIDv	25	2	0.7	1	0	0	0	0.19748	1-Apr
2	PIDv	25	2	0.5	1	0	0	0	0.20494	4-Apr
0	PIDv	25	2	0.7	1	0	0	0	0.20976	1-Apr
1	NNc	6	2	0.5	n/a	n/a	n/a	0	0.20976	15-Apr
4	PIDv	6	2	0.8	0	0	0	0	0.21213	1-Apr
8	IMC	25	2	0.5	n/a	n/a	n/a	15	0.21448	3-Apr
9	IMC	6	2	0.5	n/a	n/a	n/a	15	0.21448	3-Apr
13	PID	6	2	0.5	1.1	0.05	0	0	0.21909	22-Mar
0	IMCv	25	2	0.7	n/a	n/a	n/a	0	0.22136	1-Apr
5	IMC1	6	2	0.6	n/a	n/a	n/a	0	0.22361	20-Mar
12	PID	25	2	0.5	1.1	0.1	0	0	0.22361	22-Mar
0	IMCv	25	2	0.8	n/a	n/a	n/a	0	0.22361	1-Apr
12	IMCv	25	2	0.5	n/a	n/a	n/a	15	0.22361	4-Apr
3	NNc	6	2	0.5	n/a	n/a	n/a	0	0.22361	15-Apr

Table 6 Continued

6	PID	25	2	0.8	1	0.1	0	0	0.22583	14-Mar
4	PID	25	2	0.5	0	0	0	0	0.22804	20-Mar
9	IMCv	6	2	0.8	n/a	n/a	n/a	0	0.22804	1-Apr
6	PIDv	25	2	0.7	1	0.2	0	0	0.23022	1-Apr
5	PID	6	2	0.5	0	0	0	0	0.23238	22-Mar
9	PIDv	6	2	0.5	1	0	0	15	0.23238	4-Apr
5	PID	6	2	0.8	1	0	0	0	0.24083	14-Mar
10	PID	6	2	0.5	1	0	0	15	0.2429	3-Apr
5	PIDv	6	2	0.5	0	0	0	0	0.251	24-Apr
8	IMCv	25	2	0.8	n/a	n/a	n/a	0	0.26833	1-Apr
7	PIDv	6	2	0.7	1	0.2	0	0	0.27749	1-Apr
4	PID	25	2	0.5	0	0	0	0	0.28284	22-Mar
11	PID	6	2	0.5	1	0.1	0	0	0.2846	14-Mar
5	IMC	6	2	0.7	n/a	n/a	n/a	0	0.28636	14-Mar
9	PID	25	2	0.5	1	0	0	15	0.28636	3-Apr
4	IMC	25	2	0.7	n/a	n/a	n/a	0	0.30166	14-Mar
12	PID	25	2	0.5	1	0.1	0	0	0.30166	14-Mar
8	PID	6	2	0.5	0	0	0	0	0.31145	20-Mar
22	NNc	6	2	0.5	n/a	n/a	n/a	15	0.31623	15-Apr
14	PID	25	2	0.5	1	0.2	0.02	0	0.32094	14-Mar
13	PID	6	2	0.5	1	0.2	0.02	0	0.32863	14-Mar
15	PID	6	2	0.5	1.2	0.2	0.02	0	0.33867	14-Mar
10	PIDv	6	2	0.8	1	0	0	0	0.34641	1-Apr
16	PID	25	2	0.5	1.2	0.2	0.02	0	0.35777	14-Mar
11	PIDv	25	2	0.8	1	0	0	0	0.36056	1-Apr
16	PID	6	2	0.5	1	0	0	30	0.41593	3-Apr
15	IMC	6	2	0.5	n/a	n/a	n/a	30	0.41833	3-Apr
18	IMCv	25	2	0.5	n/a	n/a	n/a	30	0.43589	4-Apr
14	IMC	25	2	0.5	n/a	n/a	n/a	30	0.43704	3-Apr
15	PIDv	6	2	0.5	1	0	0	30	0.44272	4-Apr
8	PID	25	2	0.8	1.2	0	0	0	0.45717	14-Mar
9	PID	6	2	0.8	1.2	0.1	0	0	0.4626	14-Mar
19	IMCv	6	2	0.5	n/a	n/a	n/a	30	0.4626	4-Apr
8	PIDv	25	2	0.5	1	0	0	15	0.47906	4-Apr
16	PIDs	6	2	0.5	35	0.5	0	0	0.59161	11-Apr
15	PIDs	25	2	0.5	35	0.5	0	0	0.6	11-Apr

Table 6 Continued

15	PID	25	2	0.5	1	0	0	30	0.61968	3-Apr
5	PIDv	25	2	0.8	0	0	0	0	0.6245	1-Apr
4	PID	25	2	0.8	1	0	0	0	0.6775	14-Mar
14	PIDv	25	2	0.5	1	0	0	30	0.77136	4-Apr
0	PIDs	25	2	0.5	3	0.3	0	0	0.86603	11-Apr
1	PIDs	25	2	0.5	3	0.3	0	0	0.88318	11-Apr
2	PIDs	6	2	0.5	6	0.3	0	0	0.89443	11-Apr
11	PIDs	6	2	0.5	1.2	0.3	0	0	0.92195	9-Apr
12	PIDs	25	2	0.5	0.7	0.1	0	0	0.93274	9-Apr
9	PIDs	6	2	0.5	1	0	0	0	0.93808	9-Apr
8	PIDs	25	2	0.5	1	0	0	0	0.9434	9-Apr
10	PIDs	25	2	0.5	1.2	0.3	0	0	0.95917	9-Apr
20	NNc	25	2	0.5	n/a	n/a	n/a	15	1.08628	15-Apr
0	NNc	25	2	0.5	n/a	n/a	n/a	0	1.10454	15-Apr
2	NNc	25	2	0.5	n/a	n/a	n/a	0	1.1225	15-Apr
10	IMC	6	3	0.7	n/a	n/a	n/a	0	0.16733	20-Mar
5	IMC	6	3	0.7	n/a	n/a	n/a	0	0.17889	4-Apr
4	IMC	25	3	0.7	n/a	n/a	n/a	0	0.1957	4-Apr
6	IMC1	25	3	0.6	n/a	n/a	n/a	0	0.22136	20-Mar
9	IMCv	25	3	0.7	n/a	n/a	n/a	0	0.22804	3-Apr
7	IMCv	25	3	0.5	n/a	n/a	n/a	0	0.24495	3-Apr
6	IMCv	25	3	0.8	n/a	n/a	n/a	0	0.24495	4-Apr
9	IMC	25	3	0.7	n/a	n/a	n/a	0	0.2569	20-Mar
9	IMCv	6	3	0.7	n/a	n/a	n/a	0	0.27019	4-Apr
8	IMCv	25	3	0.7	n/a	n/a	n/a	0	0.27749	4-Apr
8	IMCv	6	3	0.5	n/a	n/a	n/a	0	0.28284	3-Apr
10	IMCv	6	3	0.7	n/a	n/a	n/a	0	0.30659	3-Apr
7	IMCv	6	3	0.8	n/a	n/a	n/a	0	0.4	4-Apr
13	NNc	6	3	0.7	n/a	n/a	n/a	0	0.66332	15-Apr
15	NNc	6	3	0.7	n/a	n/a	n/a	0	0.68557	15-Apr
14	NNc	25	3	0.7	n/a	n/a	n/a	0	0.87178	15-Apr
12	NNc	25	3	0.7	n/a	n/a	n/a	0	1.23288	15-Apr
12	IMC	6	4	0.7	n/a	n/a	n/a	0	0.20248	20-Mar
9	PID	6	4	0.5	1	0	0	0	0.23238	22-Mar
11	IMCv	6	4	0.7	n/a	n/a	n/a	0	0.24698	4-Apr
7	IMC	6	4	0.7	n/a	n/a	n/a	0	0.26077	4-Apr

Table 6 Continued

10	IMCv	25	4	0.7	n/a	n/a	n/a	0	0.27203	4-Apr
8	PID	25	4	0.5	1	0	0	0	0.29155	22-Mar
6	IMC	25	4	0.7	n/a	n/a	n/a	0	0.29833	4-Apr
11	IMC	25	4	0.7	n/a	n/a	n/a	0	0.30984	20-Mar
11	IMCv	25	4	0.7	n/a	n/a	n/a	0	0.36056	3-Apr
6	IMCv	25	4	0.8	n/a	n/a	n/a	0	0.4	1-Apr
7	PID	6	4	0.5	0	0	0	0	0.4111	22-Mar
6	PID	25	4	0.5	0	0	0	0	0.43128	22-Mar
7	IMC1	6	4	0.6	n/a	n/a	n/a	0	0.5	20-Mar
12	PIDv	6	4	0.8	1	0	0	0	0.5	1-Apr
13	PIDv	25	4	0.8	1	0	0	0	0.52915	1-Apr
7	IMCv	6	4	0.8	n/a	n/a	n/a	0	0.55678	1-Apr
17	NNc	6	4	0.7	n/a	n/a	n/a	0	0.76158	15-Apr
16	NNc	25	4	0.7	n/a	n/a	n/a	0	1.00995	15-Apr
13	IMC	25	5	0.7	n/a	n/a	n/a	0	0.31623	20-Mar
8	IMC1	25	5	0.6	n/a	n/a	n/a	0	0.51381	20-Mar
19	NNc	6	5	0.7	n/a	n/a	n/a	0	1.05357	15-Apr
18	NNc	25	5	0.7	n/a	n/a	n/a	0	1.26491	15-Apr

Table 7 Sinusoidal tracking results

Run #	Controller	Sine	f (Hz)	Filter	P	Amplitude	$\Delta\theta$	RMSE	Date
15	PID	Sin	0.1	0.5	1	6	0	0.3178	9-Apr
1	IMC	Sin	0.1	0.5	n/a	6	15	0.3873	9-Apr
0	IMC	Sin	0.1	0.7	n/a	6	0	0.3873	11-Apr
1	IMC	Sin	0.1	0.5	n/a	6	0	0.4	11-Apr
0	IMC	Sin	0.1	0.5	n/a	6	0	0.4266	9-Apr
2	IMC	Sin	0.1	0.5	n/a	6	15	0.4359	11-Apr
2	IMC	Sin	0.1	0.5	n/a	6	30	0.5745	9-Apr
16	PID	Sin	0.1	0.5	1	6	15	0.614	9-Apr
3	IMC	Sin	0.1	0.5	n/a	6	30	0.6782	11-Apr
17	PID	Sin	0.1	0.5	1	6	30	1.1402	9-Apr