Machine Tool Positioning Error Compensation Using Artificial Neural Networks

By

John M. Fines
B.S. Electrical Engineering
University of Maryland College Park, 1995

Submitted to the Department of Electrical Engineering and Computer Science and the faculty of the Graduate School of the University of Kansas in partial fulfillment of the requirements for the degree of Master of Science.

_____
Dr. Arvin Agah
(Committee Chair)

_____
Dr. Swapan Chakrabarti
(Committee Member)

_____
Dr. Frank Brown
(Committee Member)

_____
Date of Acceptance

Abstract

This thesis is a study of the application of artificial neural networks to the problem of calculating error compensation values for axis positioning on a machine tool. The primary focus is on the development of a neural network-based system that could be implemented and integrated into the open architecture control system of an actual machine. A number of neural network architectures were examined for applicability to the problem and one was selected and implemented on the machine. Positioning error compensation capabilities were tested using industry standard equipment and procedures, and the results obtained were compared with the capabilities of standard error compensation routines in machine tool controls.

Acknowledgements

I have to start out by acknowledging the people I work with and those who supported me in attending graduate school through the Technical Fellowship Program. Specifically I want to thank Jim Dycus and my manager Sam Allen for their support in getting me into the fellowship program and the continued support while I did "school" work instead of "work" work.

I also want to thank Rich Moreno and Ken Mitchell for their help in figuring out how to write part programs to run a lathe and how to operate the lathe safely and correctly. Since I managed to get through the project without wrecking the machine and I still have all my fingers, they must have done a good job. Additionally, I want to thank Mike King for his help in answering an endless series of questions about machine tool metrology and Lloyd Lazarus for the use of the machine in his department and another series of endless questions.

I want to thank Dr. Arvin Agah for his patience in dealing with a part-time student with a job and a family and his guidance and assistance in this project. Also, I would like to thank Dr. Swapan Chakrabarti and Dr. Frank Brown for consenting to be on my graduate committee.

Finally, I've got to thank my wife for her patience and support. I finally got it done.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Motivation

In a modern high-precision industrial machining environment, there is a general push to constantly produce better and better parts by reducing the dimensional errors of the machined parts. A large portion of the dimensional errors in these parts is caused by positioning errors in the axes of motion on a machine tool. Efforts to reduce these positioning errors have been on-going for many years. Much of the work has gone into making the machine tool itself more accurately. At installation, the technicians will spend many hours ensuring and correcting the structural alignment of the machine parts to reduce geometric errors such as perpendicularity of axes in a milling machine, or concentricity of a spindle and axis on a lathe. But, no matter how much work is put into making an accurate machine, it is impossible to make it perfect. And, if the dimensional tolerances required are fine enough, the basic mechanical errors in machine motion will impact the quality of the parts that are produced.

In the continuing effort to improve the performance of machining tools, control routines to compensate for uncorrected positioning errors have been developed. Machine errors are measured with some type of a high-precision device, such as a laser interferometer system. This information is then entered into the control system of the machine tool, and it is used to correct for known errors in the machine. For example, if a machine consistently falls 0.002 inches short of the desired position when moving an axis, the control system will command a move 0.002 inches longer than desired to compensate for the expected error. Using this method, modern machine tools have been able to reduce positioning errors to a level lower than the mechanical accuracy of the machine.

Typical positioning error compensation routines in modern machine tool control systems have one basic characteristic in common. They allow for a limited number of compensation values to be entered and the values must be at fixed intervals of motion. Some systems will allow the user to pick the interval, and, in some systems, this value is determined at design time by the machine tool builder.

With the advent of open architecture control systems, which allow for the integration of external control routines into the basic function of the control system, there is now interest in more advanced and flexible error compensation routines.

1.2 Research Methodology

The focus of this thesis is the design and implementation of a neural network-based positioning error compensation system for an actual machine tool in an industrial environment. A large, standard configuration, two-axis lathe was chosen as a test machine, and the machine errors were measured using a laser interferometer system already available in the factory for this purpose. The lathe was chosen because it had recently been retrofitted with a PC-based open architecture control system, which allowed for the integration of external error compensation routines. The neural network design and development was done using Matlab® and it's associated Neural Network

Toolbox from The MathWorks, Inc (Demuth and Beale). The programming required to implement the system was accomplished with standard PC programming tools.

In the course of this thesis, several different neural network architectures were examined, with three being closely studied for their application to this problem For reasons of training speed and the ability to reduce the errors to an absolute minimum, one of the networks was selected and finally implemented in the live machine. A small set of learning algorithms for the networks was examined with a determination of one as best suited for the problem at hand.

The neural network based system was implemented in its most basic form and tested for its ability to compensate for the mechanical errors of the lathe. The resulting error compensation system was shown to be as good as the current state-of-the-art in commercially available machine tool control systems. This was based on its ability to reduce the positioning errors as measured by the laser interferometer system to a level slightly below that of the controls standard error compensation capability.

1.3 Thesis Structure

This thesis is organized into six chapters and five appendices. Chapter 1 is the introduction, which discusses the motivation for the thesis and the basic methodology for the experiments conducted. This chapter highlights the desire for improving the performance of machine tools and how that will be attempted by improving the machine tools ability to compensate for positioning errors in the axes of motion. Chapter 2 is the background information for the thesis. In it is discussed the equipment, machinery and systems used in the experiments and a very basic discussion of artificial neural networks. Chapter 3 is the related works section of the thesis. In it is discussed the current methods of positioning error compensation and some applications of artificial neural networks that exist in the literature. Chapter 4 is the experimental setup discussion. This chapter presents the measurement system used to evaluate positioning error on the machine tool, how the artificial neural networks were designed and implemented, and how the resulting neural network was integrated into the machine tool control. Chapter 5 is the experimental results. In it is discussed the numerical results achieved by the artificial neural network positioning error compensation system and a comparison with the results achieved by traditional methods of compensation on the same machine. Chapter 6 is the conclusions section. The discussion in the chapter is on the conclusions reached after examining the experimental results, the limitations of the work conducted in this thesis, and a discussion of future work.

The five appendices contain information that expands on the discussions in the main chapters. Appendix A contains detailed photographs of the measurement equipment and the machines that highlight key features for this thesis. Appendix B contains the complete listing of the Visual Basic macro and resultant machine control code that was used in the course of this thesis. Appendix C is the complete listing of the Matlab m-file script that was used to generate the artificial neural networks for this thesis. Appendix D is the complete listing of the Matlab m-file used for the extraction of key neural network paramaters required for integration into the machine tool control. Appendix E is the listing for the C program that provides the necessary calculations to integration the artificial neural network into the machine tool control.

Chapter 2 – Background

The technological basis for this thesis lies in four areas: Computer Numerically Controlled (CNC) machine tools, Open Architecture Controls (OACs) for machine tools, laser interferometry, and artificial neural networks. From the perspective of this thesis, three of these areas were fully developed and available equipment and systems were used. The CNC machine used is an American Hustler lathe (American Tool), the OAC on the lathe is OpenCNC® from Manufacturing Data Systems, Inc. (MDSI, *V6.2 Datasheet*), and the laser interferometer is a 5/6-D Laser Measuring System from Automated Precision, Inc (Automated Precision). This thesis focuses on the application of neural networks to the problem of compensating for positioning errors in the axes of motion on machine tools.

2.1 American Hustler Lathe

The American Hustler lathe is a standard, single spindle, two-axis CNC lathe, as shown in Figure 2.1. The two axes are labeled X and Z with the X axis being the cross-slide which moves perpendicular to the spindle center line and the Z axis being the major carriage axis which moves parallel to the spindle center line (American Tool).

In most operations, control of the lathe is handled by the automatic control system, as programmed by the operator. In other words, a predetermined sequence of operations is programmed into the control using standard G and M code programming. The control is then set into the "Automatic" mode and a "Cycle Start" is initiated. It will then step through the pre-programmed sequence of operations with no further intervention by the operator. The basic operations available to the operator are control of axis motion (direction and rate of travel), control of the spindle (direction and rate of rotation), and control of other machine functions such as use of coolant or selection of cutting tools. More advanced functions are available for capabilities such as interpolation of arcs and thread-cutting operations (American Tool).

Since the motions studied were single-axis and linear, the programming of the lathe was restricted to only those commands required to produce these movements. The normal pattern

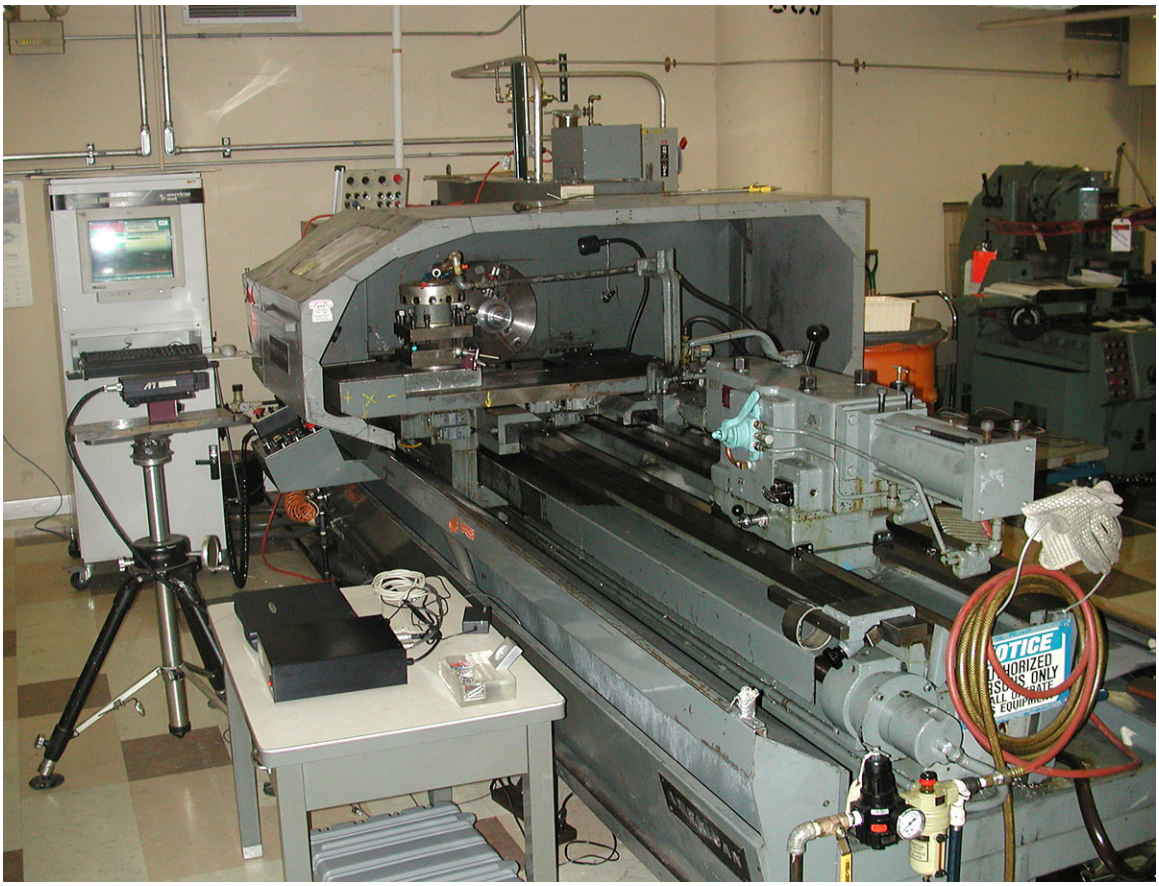


Figure 2.1 The Hustler lathe by American Tool, Inc.

of motion could be accomplished by repeating these two commands:

```
G01X#1F#2
G04F#3
```

Where G01 is the G code command for single axis, controlled feedrate motion. The first line would command the X axis to move to position '#1', at feedrate '#2' (normally inches per minute). G04 is the command for a program dwell. Therefore, the second line would command a program dwell for '#3' seconds at the current position. An example of the complete command lines is:

```
G01X3.5 F20
G04F6
```

These two lines would command the X axis to move to the 3.5 inch position at 20 inches per minute, then dwell at this point for 6 seconds. The entire program would be constructed by a series of these moves either specifically listed out in the program, or some type of looping structure would be used in the program, while setting the values to #1, #2, and #3.

2.2 Open Architecture Control

According to the Open Modular Architecture Controls User's Group (OMAC), a CNC control is generally considered to be an Open Architecture Control (OAC) if it has

the characteristics of being open, economical, maintainable, modular, and scalable (OMAC). The key characteristic applied to this thesis was that of being "open". OMAC presents a definition of an open control as: "allowing the integration of off-the-shelf hardware and software components into a controller infrastructure that supports a de facto standard environment." (OMAC). It is the ability to integrate external software components into the controller that makes it possible to write custom error compensation routines and use them in real-time. In the case of the lathe used in this project, the control system was OpenCNC from Manufacturing Data Systems, Inc. (MDSI).

OpenCNC is an open-architecture, software based CNC control system (MDSI, *V6.2_Datasheet*) suitable for use on most typical CNC controlled machines. OpenCNC runs on the Microsoft Windows NT (Microsoft) operating system and uses the add-on Real-Time Extensions from VentureCom, Inc. (VenturCom) to give it the hard, real-time, deterministic response required to maintain control of the machine tool. Since OpenCNC uses a standard PC operating system, it runs on generic PC hardware. It also has a published Application Programming Interface (API) that allows the control designer to develop external software routines and interface them with the control system using standard PC software development tools.

The internal architecture of OpenCNC is centered around its Real-Time Database (MDSI *API Datasheet*). This database contains all the system variables for the control. These system variables contain the current state of the machine under control, and define all of the control characteristics. Literally, every function of the machine and the control are defined and represented in the system variables. As part of the database, there are system variables for the location, velocity, and direction of motion of each of the axes on the machine, along with any error compensation values that should be applied to the positioning of the axis.

The API that is available with OpenCNC provides functionality at two levels. The API Level 1 provides access to the system variables in the database. It is often used to develop additional user-interfaces to the control or to add functionality such as e-mail notices or system monitoring capability. However, the API Level 1 is used for non real-time applications and does not have the functionality to create scheduled, deterministic, hard real-time applications. Applications created using the API Level 1 run with the same priority and scheduling scheme as any other application running on the Windows NT operating system.

OpenCNC's API Level 2 is used to develop those applications that require deterministic, hard real-time response. It also provides access to the system variables in the database, and additionally provides the ability to schedule the execution of processes at predetermined time intervals and set the priority of processes to levels normally reserved for internal control functions. Examples of applications created using API Level 2 are thermal compensation routines, such as monitoring machine temperature and adjusting for positioning errors induced by thermal expansion, or other machine error-compensation routines (MDSI *API Datasheet*).

While the software development tools used for both levels of the API are standard tools, there are restrictions on the functions available for use when developing API Level 2 applications. These are based on the list of functions supported by the VentureCom RTX in the C Run-Time Library-Supported API (VCI).

2.3 Laser Interferometers

The laser interferometers have been around for many years, and today are commonly used to make extremely accurate measurements of linear displacement. The basic theory for laser interferometers dates back to the early 1900s with the Michelson Interferometer being one of the earliest devices to demonstrate interferometric measurements of length (Jenkins and White). Modern interferometers are highly portable devices that are relatively easy to set-up and use. They are normally computer controlled devices that offer an array of analysis and recording capabilities.

2.3.1 General Theory

A laser interferometer uses a laser source that emits a very focused, monochromatic, light beam. In its simplest form, the normal set-up for a laser interferometer uses a stationary light source, a beam splitter, a stationary reference reflective target, and a mobile reflective target, as illustrated in Figure 2.2. The emitted beam is projected through the beam splitter which results in two separate beams of light that start out with matching phases. These beams are then projected onto the two targets and reflected back to the beam splitter which combines them into a single beam again. The resultant combined beam is then examined for fringes created by a mismatch in the phase relationship between the two returned beams. By counting these fringes as the mobile target is moved, the interferometer system can determine linear displacement as a function of the wavelength of the light beam.

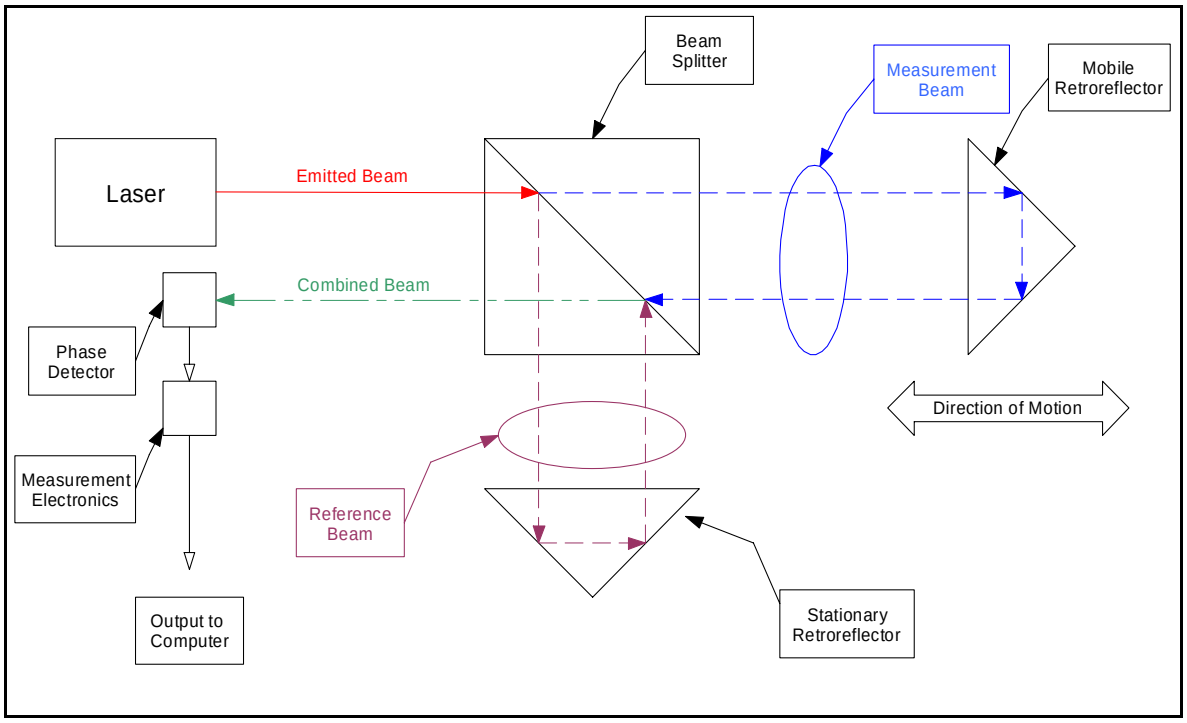


Figure 2.2 Simplified block diagram for laser interferometer.

Since the measurement of distances with an interferometer is accomplished by comparing phase relationships between the reference beam and the measurement beam,

the accuracy of the system is dependent on the wavelength of light used and the system's ability to distinguish fringes created by phase mismatches. The 5/6-D Laser system used for this thesis is a Helium-Neon laser with a wavelength of 0.6329 microns. The system is able to count fringes in ¼ fringe increments. This leads to a resolution of $0.6329 \times 1/4 = 0.1582$ microns or approximately 6.23 microinches (Automated Precision, Inc.).

2.3.2 Physical Setup

The physical set up of the laser interferometer consists of arranging the stationary and mobile components of the system so that the relative motion between these components accurately reflects the motion of the machine under measurement. Mounting of the mobile reflector is normally done with a magnetic base, and it is mounted, as closely as possible, to mimic the motion of the cutting tool of the machine. The stationary components are mounted off the machine, on a tripod, to insulate them from vibration of the machine in motion. For a linear measurement (as in this case) they are aligned so that the emitted beam leaves the interferometer from the lower aperture and is returned from the mobile reflector to the upper aperture throughout the entire range of motion to be measured. Figure 2.3 shows a view of the physical setup and Figure A.1 and Figure A.2 illustrate detailed views of the components.

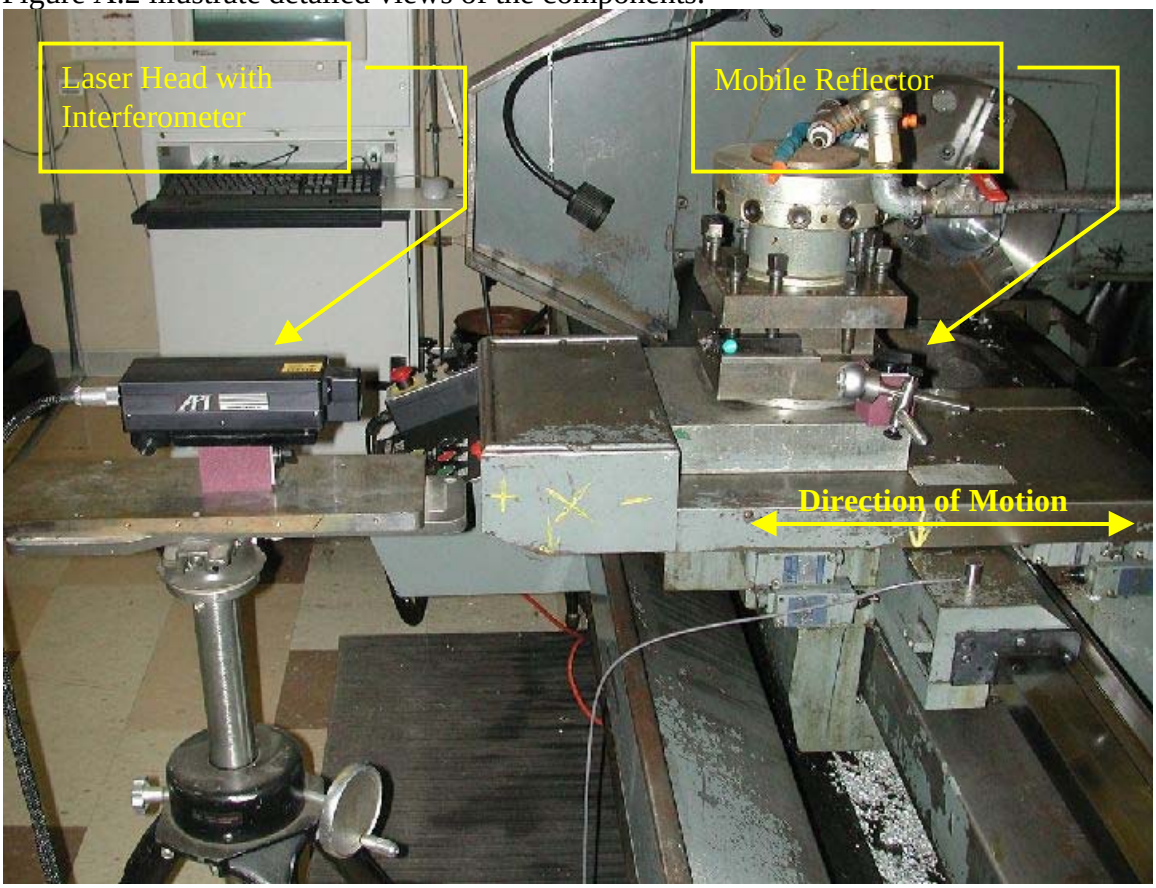


Figure 2.3 Physical setup of laser interferometer.

2.3.3 Software Setup

The 5/6-D Laser system is controlled by software on a standard laptop computer connected via an RS-232 serial connection. For this project, the APInc's Winner v1.3.1 package was used (Automated Precision). To successfully collect information from the laser interferometer system, the software must be setup to coordinate the reading of data from the system with the motion of the machine tool. In this case, the motion of the lathe was under automatic control of a part program written specifically for this purpose.

Setting up the Winner software for linear motion testing is done by filling in entries on the setup screen when it is presented, as shown in Figure 2.4. The key entries for linear testing are explained in Table 2-1.

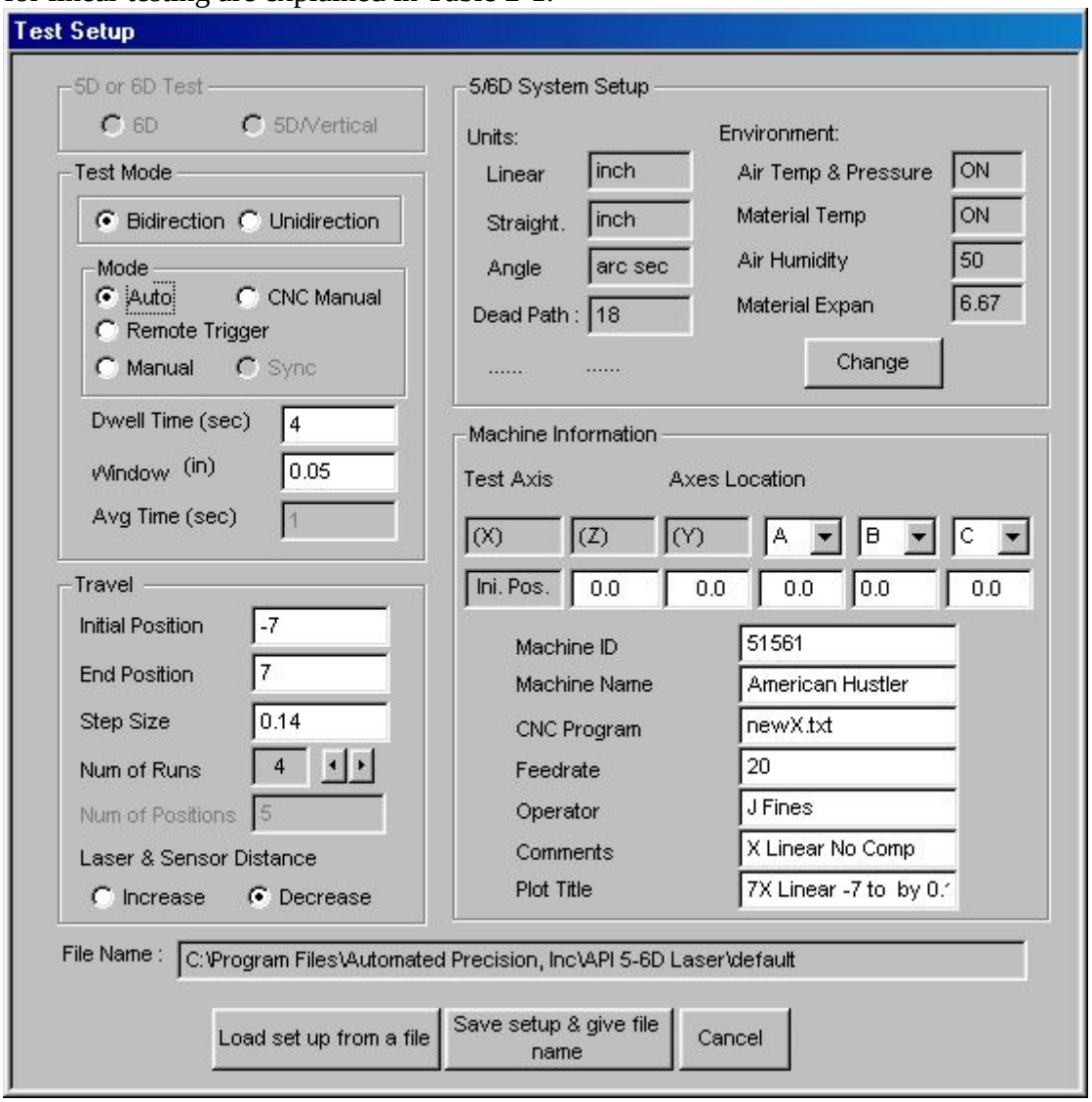


Figure 2.4 Setup screen for control of the 5/6-D laser system.

Entry	Explanation of Selections
Test Mode	In a Bidirection test, readings are taken in both directions of motion. Therefore the machine is expected to move, by the specified interval, forward from the starting point, and then in reverse from the ending point.
Mode	In the Auto mode, machine motion is controlled automatically by a part program. The laser system is set to expect a specified sequence of moves and uses the Dwell Time and Window parameters to trigger readings.
Dwell Time	Specifies the time (in seconds) for the laser to read at each position. In Auto mode, when the machine moves to within the Window parameter of the next position, the laser system will begin its Dwell timer. Halfway through the Dwell time, the laser will begin taking its reading with the final reading at the end of the Dwell time. This allows for the machine to complete the move to the next position and settle in place.
Window	Specifies how close the machine has to be to the next position to start the Dwell timer.
Initial Position	This is the starting point of the laser run. For multiple Bidirectional runs, the system expects the machine to incrementally move back to this position. When the prior run is complete, the machine is expected to move past this position (outside the Windows distance), and then return to the starting point for the first reading of the next run.
End Position	This is the ending point of the laser run. For a Bidirectional run, the system expects the machine to incrementally move to this position in the forward direction, then move past it (outside the Window distance) and then return to this position for the first reading in the reverse direction.
Step Size	This is the distance between measurement points.
Num of Runs	This is the number of complete runs to measure. A Bidirectional run is complete when the machine has moved from the starting point, to the ending point, and back to the starting point for a final reading.

Table 2-1 Key entries for software setup.

The other entries on the setup screen are for general system parameters and labeling the results with name, title, comments, etc.

Just as the 5/6-D Laser system requires setup to correctly measure the motion of the lathe, the lathe itself requires programming so that it would make the expected sequence of moves for correct measurement. The required machine programming is accomplished by a Visual Basic macro that produces the G and M code commands for the machine. This macro is called by a small part program that establishes the parameters for the test being run. Changing this program for a different series of moves (i.e., moving a different interval or a different axis of motion) is accomplished by editing the values of

the variables in the first several lines of the program. Table 2-2 is an extract of the key lines in the part program matching the setup illustrated in Figure 2.4.

vb #1 = 1	'Axis 1=X, 2=Y, 3=Z
vb #3 = -7	'Starting Position in Machine Coordinates
vb #4 = 7	'End Position in Machine Coordinates
vb #5 = 0.14	'Increment of motion between points
vb #6 = 6	'Dwell time at points
vb #7 = 4	'Number of complete runs
vb #8 = 20	'Feedrate for motion
vb #9 = .1	'Overshoot for backlash moves

Table 2-2 Part program entries for X-axis motion.

Close examination and comparison between the two shows one key difference – the Dwell parameter on the machine is set for 6 seconds while the Dwell parameter on the 5/6-D Laser is set for 4 seconds. After several runs of collecting data it was determined that these were the best real-world settings for reliably collecting accurate data. Attempting to too closely time the moves of the machine with reading the laser system sometimes leads to faulty readings and errors. These settings are conservative and insure that the machine motion will settle before the laser system takes its reading, and the machine will not begin motion to the next position until well after the laser system has completed the previous reading. Figure 2.5 diagrams the relationship between the motion of the machine and the activity of the 5/6-D Laser system for taking measurements. The key point here is that although there is no direct connection between the control of the laser system and the lathe motion, the laser expects a specific series of moves to be made, and the lathe is programmed to make those exact moves.

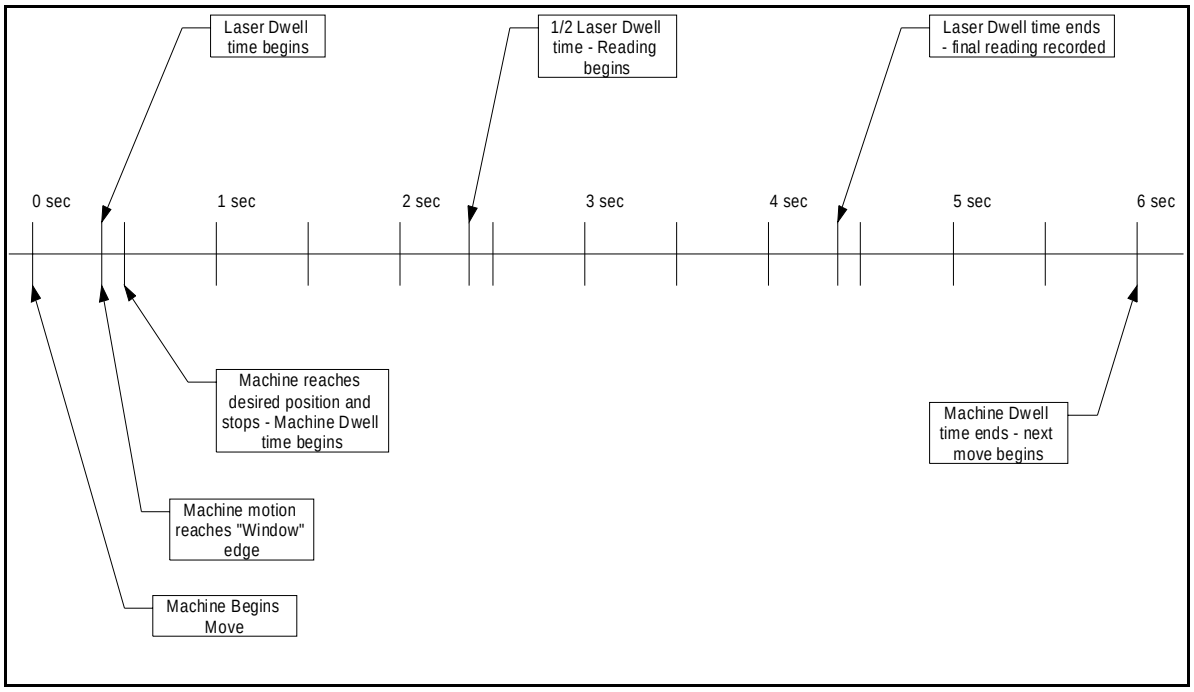


Figure 2.5 Machine motion - laser timing diagram.

2.3.4 Graphical Results and Output Data

The results of a successful laser interferometer run are presented using three methods. During the run, a screen is displayed on the controlling laptop that shows the current state and activity of the system. After the run is complete an ASCII test data file is produced which can be manually analyzed or opened from within the 5/6-D laser system for a graphical display of the results.

While a run of the 5/6-D laser system is in progress, the screen shown in Figure 2.6 is displayed. This shows the current reading of the laser distance measurement in the upper-left block. A graphical display of the results collected in this run is plotted in the upper-right block. Operating parameters of the current run are displayed in the lower-left area, with the bottom-right blocks showing the current status, target position, and run number.

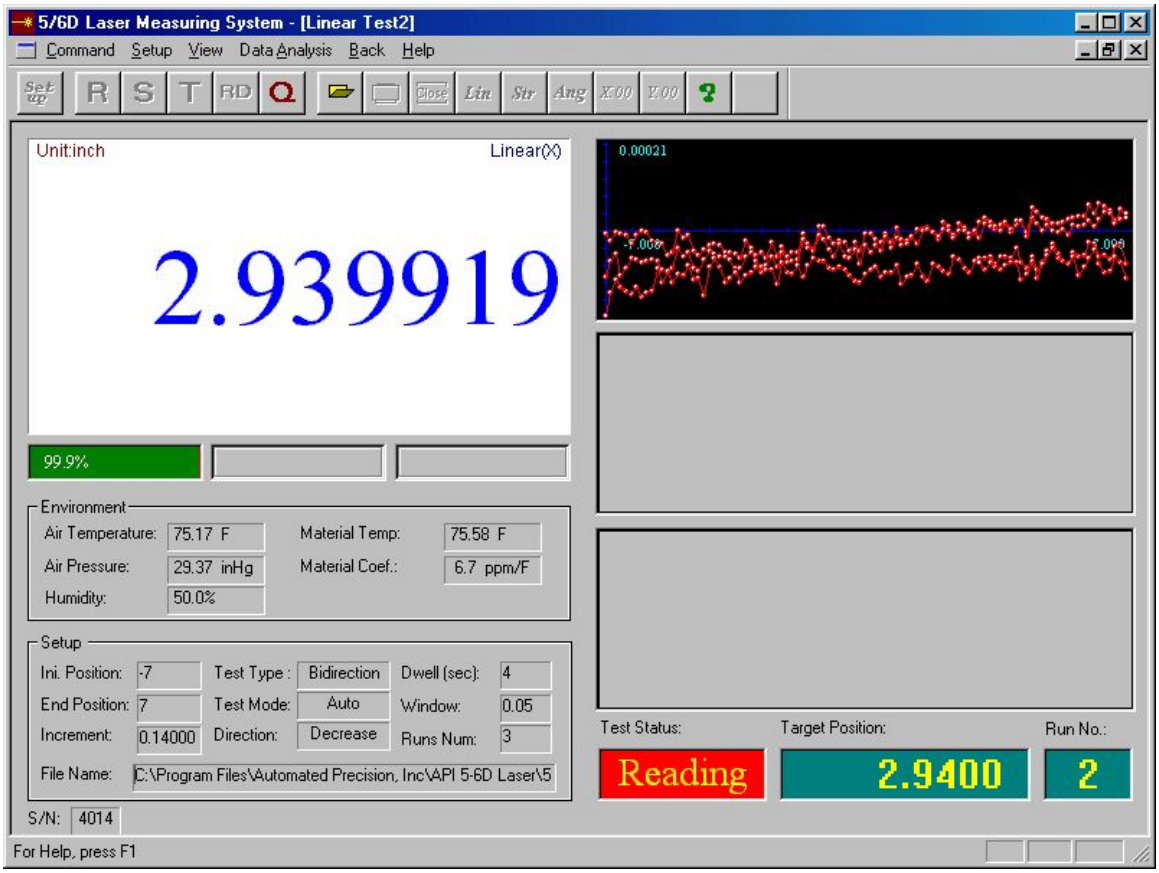


Figure 2.6 5/6-D laser system in progress screen.

After all runs of the current test are completed, the 5/6-D laser system closes the ASCII text data file for this run. This file contains all of the data collected for this run and other summary data from the 5/6-D laser system. Table 2-3 is a partial list of results from the 5/6-D laser system run used as an example in the previous screens. As seen in the table, the system records the current position, laser reading and the calculated linear error. The position and laser reading are different because the system is "zeroed" at the starting point of the run – therefore in this case, a position of –7.00000 inches is equivalent to a laser reading of 0.00000 inch. Since the laser reading was –0.000004 inches the calculated error is +0.000004 inches. The system also records the current air temperature and pressure, and the current material temperature of the machine under test. This data is collected from sensors attached to the 5/6-D laser system and is used in internal laser wavelength and material temperature compensation algorithms (Automated Precision, Inc.).

Position	LasRead	LinearError	Status	AirTemp	AirPress	MatTemp
-7.00000	-0.000004	0.000004	0	73.76	29.33	74.65
-6.86000	-0.139901	-0.000099	0	73.76	29.33	74.65
-6.72000	-0.279949	-0.000051	0	73.76	29.34	74.65
-6.58000	-0.419945	-0.000055	0	73.76	29.34	74.65
-6.44000	-0.559952	-0.000048	0	73.76	29.34	74.65
-6.30000	-0.699888	-0.000112	0	73.76	29.34	74.65

Table 2-3 Partial list of output data from 5/6-D laser system.

The resulting data from a complete 5/6-D laser system run may also be graphically displayed using APInc's Winner v1.3.1 software. Figure 2.7 displays the

results of the example laser run. This screen is a complete display of operating parameters of the recorded run, a plot of all collected data points, and calculated error values for the entire run. For a perfect machine, all the data points would lie directly on the 0.000000 line in the middle of the plot. Variation from this line is the error in the actual motion of the machine from the commanded motion. In Figure 2.7, the lower curve is the data from the forward motion of the machine, while the upper curve is the data from reverse motion. The difference between the two curves is a result of the mechanical backlash of the machine that occurs when the leadscrew-ballnut assembly reverses its direction of motion. This value is also calculated and displayed as "Max Rev Err" (maximum reversal error). The two horizontal dotted lines are the high and low points of the average error values for, in this case, the four runs. This value is calculated and displayed as "Max Avg Error" (maximum average error). These are the two key values that give an indication of the mechanical condition of the machine. Compensation routines discussed elsewhere are directed at reducing these values to acceptable levels.

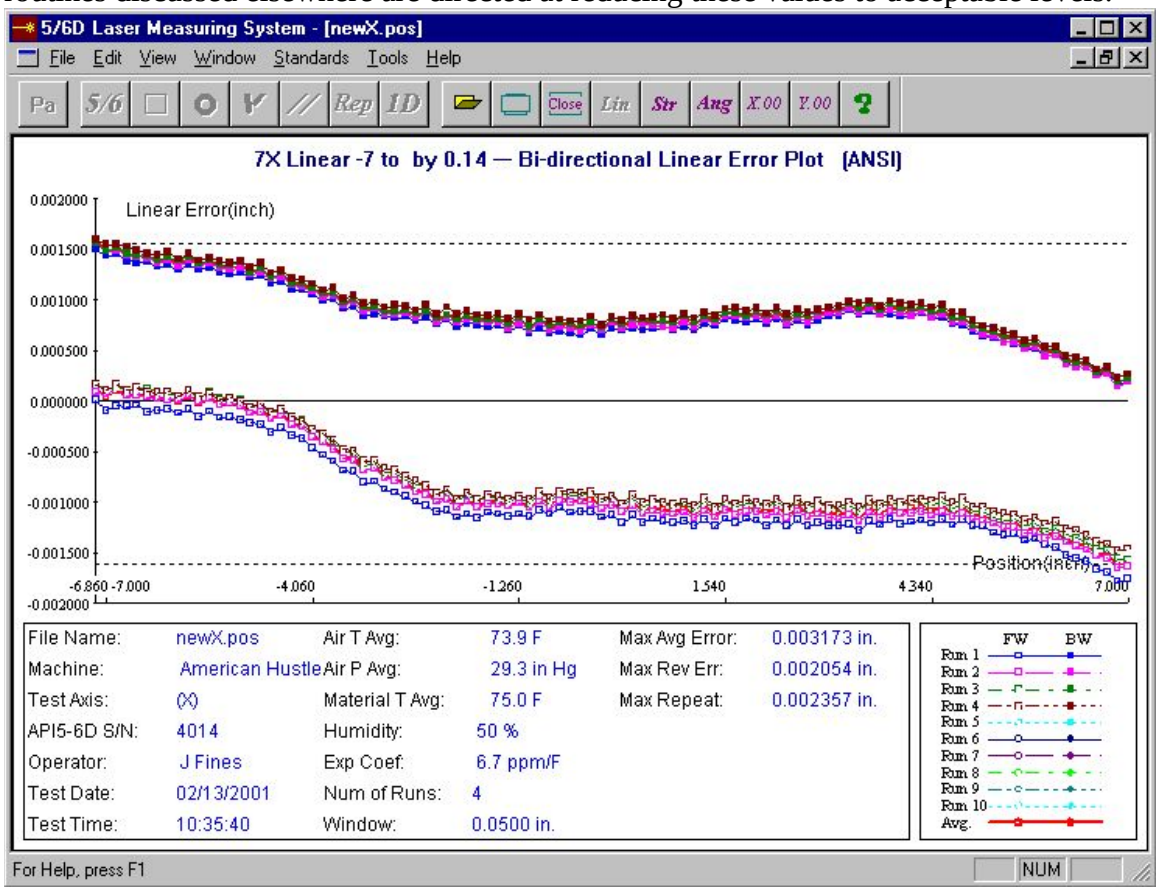


Figure 2.7 Graphical display of 5/6-D laser system data.

2.4 Artificial Neural Networks

As one of the forms of artificial intelligence, artificial neural networks are an attempt to mimic the behavior and capabilities of biological neural networks. In particular, neural networks are designed to model the behavior of the human brain (Russell and Norvig). For this thesis, the key characteristics of interest are the ability of a

network to be trained on a problem, and its ability to learn or to improve its performance as it is presented with more data.

The fundamental structure of a neural network is that of a massively interconnected collection of simple computing elements (neurons), operating in parallel to produce an output (or set of outputs) in response to a set of inputs. Figure 2.8 illustrates the interconnections of a small, fully connected neural network. In this figure, each of the input neurons feeds information to each of the neurons in the next layer, and each of those neurons feed information to the output neuron.

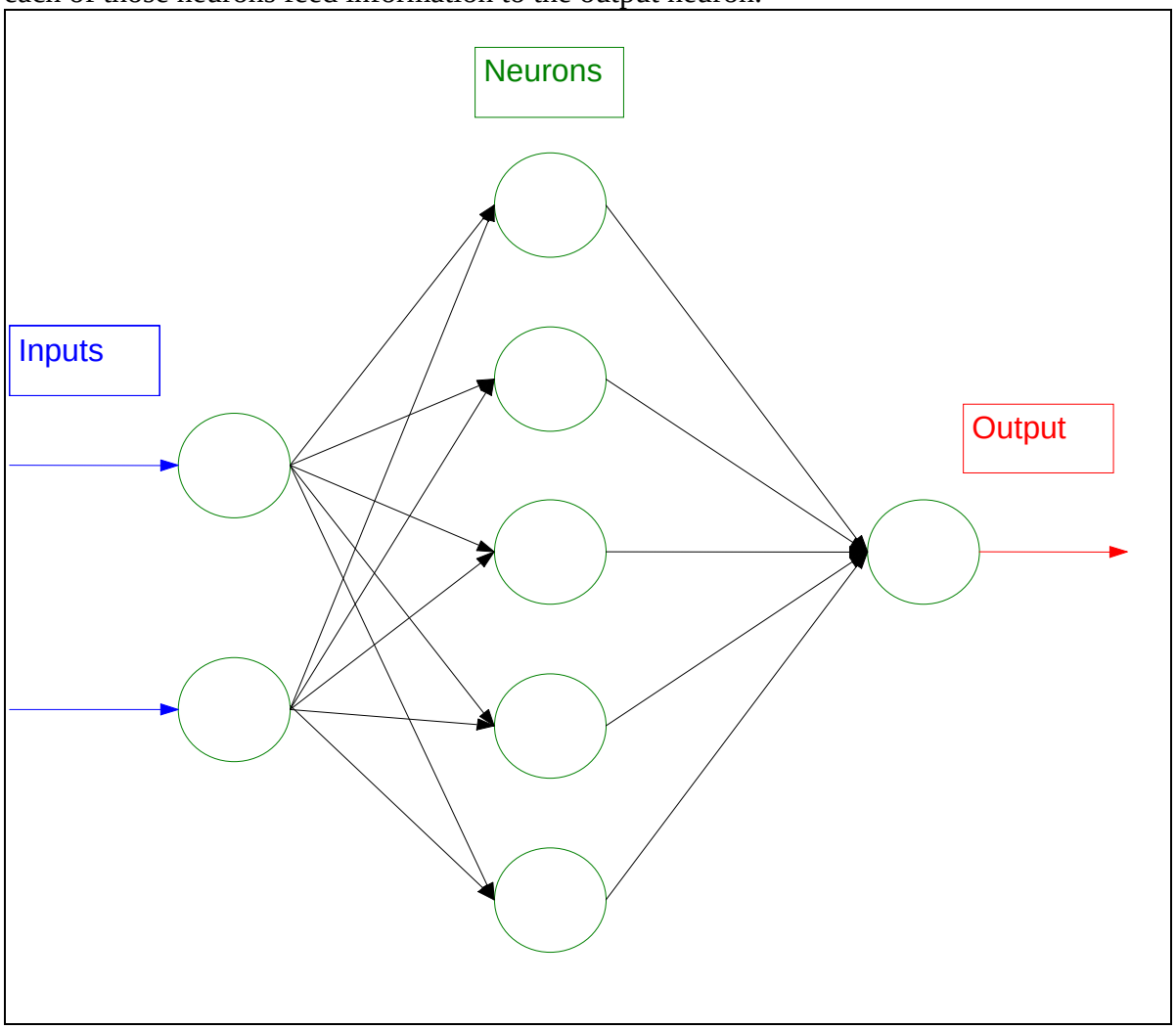


Figure 2.8 Neural network interconnections.

It is this characteristic of having every neuron in each layer connected to every neuron in the next layer that defines this network as being fully connected. If some of the connections were missing, the network would be defined as partially connected (Haykin).

The computational elements of the neural network are in two areas. The first is in the interconnections themselves. Each of these interconnections has a weight assigned to it. The weight assigned to the interconnection becomes part of the information passed to the connected neuron. The second computational element of the network is the transfer

function of each neuron. The transfer function is used to generate the output of each neuron based on the complete set of inputs to that individual neuron.

In its simplest form, a neural network would consist of a single input to a single neuron, which outputs a single output. A more useful form of this single neuron network is diagrammed in Figure 2.9. In this case, a single neuron with several inputs generates a single output. The figure illustrates the network consisting of a set of inputs **I**, weighted connections **W**, a single neuron with transfer function **T**, and output **O**. It also adds the concept of an applied bias **B**. Where **B** is used to offset or bias the output of the neuron. **B** is also sometimes viewed as another weighted connection to a fixed input value of 1. A mathematical description of the single-neuron network is:

$$O = T\left(I_1 * W_1 + I_2 * W_2 + ... + B\right)$$

The transfer function of the neuron is determined when the network is initially designed and created. Determination of the transfer function is one of the basic issues the network designer has to address.

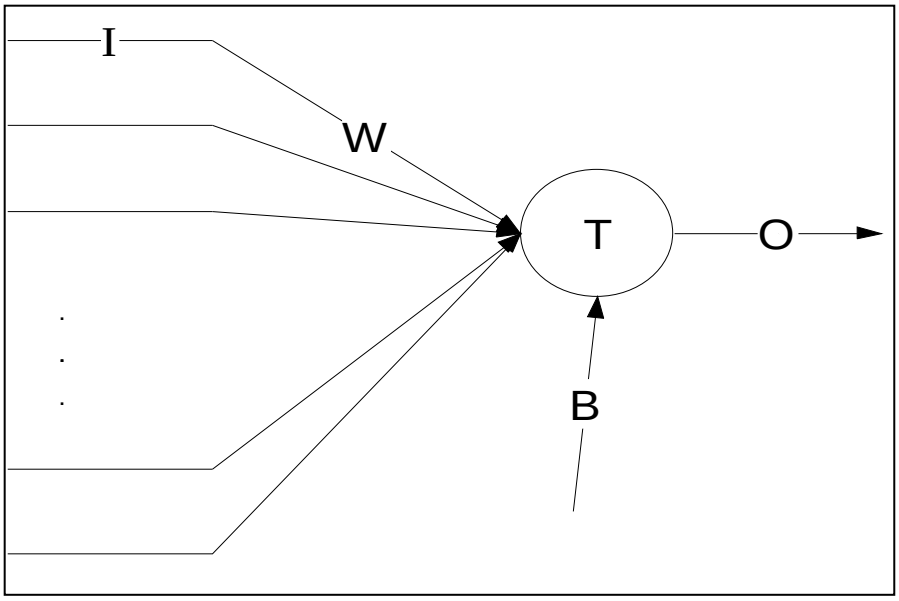
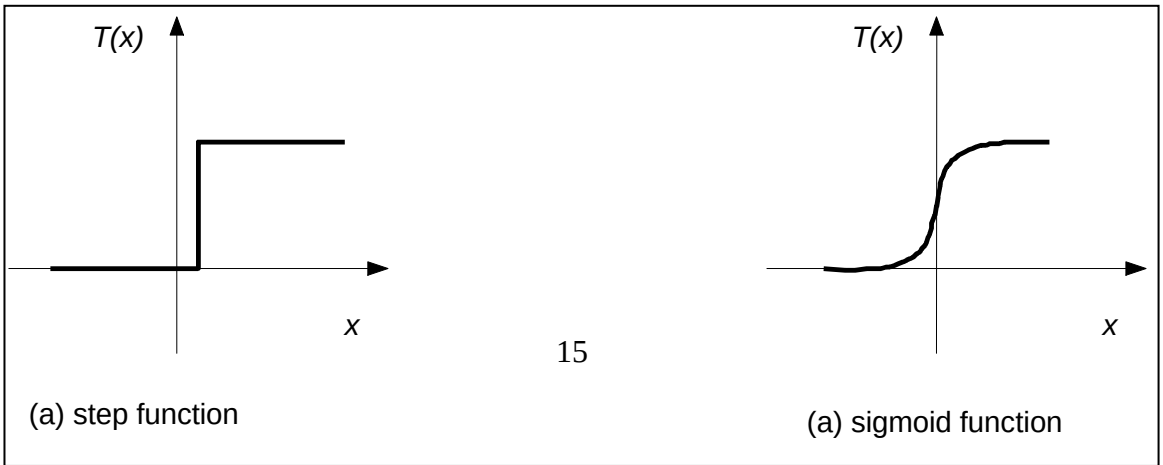


Figure 2.9 Single-neuron network.

There are many different types of transfer functions used in neural networks but a few examples are commonly presented in the literature, including the step (or threshold) function and the sigmoid function. Simple graphical illustrations of these two functions are shown in Figure 2.10. Figure 2.10 (a) shows the step function.

In this function, the output is held at zero until the input reaches a threshold value. At that point, the output immediately steps to a fixed value (normally one). In Figure 2.10



(a) step function

(a) sigmoid function

(b) the sigmoid function is illustrated. While the sigmoid has the same general shape as the step function, there are some key differences. First, it can take on values anywhere between zero and one, where the step function is either zero or one. Second, the sigmoid function is differentiable. This becomes an important issue in later discussions on training algorithms. Once the transfer function of the neuron(s) has been defined, determining the weights and bias values is the other portion of defining the functionality of the network.

Figure 2.10 Step and sigmoid function graphs.

Determining the values for the weights and bias for the neurons in a network is accomplished by the training process. Haykin generalizes the training (or learning) process to three steps of stimulating the network, changing the network parameters based on the stimulation, and the network responding in a different manner due to the changes (Haykin). For example, in error-correction learning, the network is stimulated by a set of training data in the form of input-output pairs. Each pair is a set of inputs combined with the desired output for this input. The actual output of the network is compared to the desired, and an error value is produced. This error value is then used to adjust the network parameters such that it will produce an output that is closer to the desired.

Mathematically, referring to the single neuron model in Figure 2.9 where *I* is the input vector, *W* is the weight vector, and *O* is the output of the neuron, and adding *D* as the desired output from the training data, and the factor η, the learning rule can be stated as:

$$W^{new} = W^{old} + \eta \bullet I \bullet (D - O)$$

From this equation, the following points can be made:

1. The new weight values are not totally new values, but rather modifications of the previous values after some incremental adjustment.
2. Since *D* is the desired output, and *O* is the actual, if *D*-*O* is positive the output *O* is increased in the next iteration, and if *D*-*O* is negative, the output is decreased in the next iteration. Since each individual input contributes $W \bullet I$ to the total input, the error, *D*-*O*, is multiplied by each individual input *I* in the adjustment factor. This way, if this input is positive, its associated weight is greater, giving it more influence on the total input to increase it. If the input is negative, the associated weight is decreased, reducing this input's influence on the total input - also increasing it.
3. The factor η is known as the learning rate. It is used to adjust the learning rule for a balance between stability and speed of convergence. A large learning rate will make the system converge more quickly to an acceptable state (reducing the error to acceptable limits), but too large of a learning rate may make the system unstable by causing the learning rule to over-correct for small errors causing it to oscillate around the desired weight values.

The rules stated above are often presented in neural network texts as the initial starting point for a study of learning rules or training algorithms, and complete coverage can be found in Russell, Hagan, and Haykin.

2.5 MATLAB and the Neural Network Toolbox

There are two primary methods of implementing a neural network system. One is in dedicated hardware, and the other is to simulate the network on a digital computer. Because of the obvious cost and flexibility concerns, the latter is the most common method. Also, because of the programming complexities of implementing the various types of networks, it is often advantageous to use a commercially available packaged set of routines to design and develop a neural network system. One such package is the Neural Network Toolbox from The Mathworks, Inc (Demuth and Beale). This package runs under The Mathworks' MATLAB program and extends its capability to include many of the functions necessary to design and implement a neural network system.

MATLAB and the Neural Network Toolbox provide the capability to design many different types of neural network systems for a variety of applications. The Neural Network Toolbox User's Guide includes chapters on applications in control systems, linear and adaptive filters, and others (Demuth and Beale). There is a Graphical User Interface (GUI) for interacting with the routines in the toolbox, or command-line access for use in MATLAB's programming and scripting capability. The Toolbox also includes an extensive set of built-in transfer functions to use in defining the neurons of the network, and many different training routines are included for use. All of the neural

network development and implementation discussed in this thesis was done using MATLAB and the Neural Network Toolbox.

Utilizing the functionality of the Neural Network Toolbox (referred to here as simply the toolbox), defining a new neural network becomes a fairly simple series of one-line commands with a few parameters to create, train, and simulate the specific characteristics and behavior of the network. The toolbox has commands to create perceptrons, radial basis networks, competitive, feed-forward networks, and many others (Demuth and Beale). For example, creating a two-input, single-neuron perceptron network where one input ranges from 0 to 2, and the second from –1 to 1 is accomplished by using the command:

```
net = newp([0 2: -1 1],1);
```

A more complex (and useful) feed-forward network with two inputs, ranging from 0 to 1 and 1 to 40, with 30 neurons in the hidden layer, with the *logsig* transfer function, and a single, linear neuron in the output layer can be created using the command:

```
net = newff([0 1: 1 40],[30 1],
'logsig','purelin');
```

Where logsig is defined as: $\frac{1}{1+e^{-x}}$

This network has a structure similar to the network shown in Figure 2.8.

Training a newly created neural network can also be accomplished using a single command such as:

```
net = train(net,p,t);
```

where `p` is a set of input vectors and `t` is the set of associated target outputs for `p`. This command would train the network for the toolbox default of 100 epochs (one epoch is a single pass through all the inputs and targets) utilizing the other toolbox default parameters for training function, performance function and other parameters. If desired, or necessary, all of the default parameters can be changed to meet the specific needs of the application under study.

Simulating network behavior when presented with a set of inputs is accomplished using the command:

```
sim(net,p);
```

which presents the network the inputs `p` and calculates the output of the network based on its current definition. For example, in the case of a single-neuron perceptron with two inputs, with the input weights defined as [1 2], a bias value of 0.5, inputs of -1 and 1, and using the "hardlim" transfer function in the neuron, the function sim() performs the following calcuation:

$$a = hard\lim\left(\begin{bmatrix}1 & 2\end{bmatrix}\begin{bmatrix}-1\\1\end{bmatrix} + 0.5\right)$$

where *a* is the network output.

By utilizing the functions of the toolbox, a network designer can quickly create, train, and test a neural network for the application of interest. With some knowledge of neural networks and some study of the toolbox User's Guide, the designer can rapidly get into very complex networks and customize the details of the network structure as desired, while taking for granted the lowest-level details of network implementation in a software simulation environment.

Chapter 3 – Related Works

Since the primary goal of this thesis was to implement a neural network-based error-compensation capability for a machine tool, the related works falls into three areas. The first area is the traditional methods of machine tool error compensation. JThe second is a brief survey of other applications of neural networks. The third area is a review of available material on attempts to use a neural network-based error compensation routine for machine positioning errors.

3.1 Traditional Lead-Screw Compensation

Any modern machine tool control will contain some capability to compensate for linear axis positioning errors. This is often referred to as lead-screw error compensation – so named because the major cause of positioning errors is often the lead-screw mechanism that drives the motion (Figure A.3 and Figure A.4 illustrate the detailed views). The errors occur when, usually due to mechanical variations or wear, the machine does not exactly reach the commanded position. For example, the X-axis of a lathe may be commanded to move to the +4 inch position, but if measured with a very precise measurement system it will actually be at +3.9980 inch – thus an error of -0.002 inches in axis positioning. The normal method of compensation is accomplished by creating a table of these positioning errors for a set of points at fixed intervals. For a large machine this might be at every even inch for the entire range of travel. Other machines may have other intervals, or the end-user might be able to choose an interval that best meets the requirements. Giddings & Lewis and MDSI (*Variable Dictionary*) include the details for methods of implementing lead-screw compensation. There are two basic types of lead-screw error compensation, the first is uni-directional with backlash compensation, and the second is bi-directional compensation.

3.1.1 Uni-Directional Compensation with Backlash

In uni-directional compensation (with backlash), the error measurements are all taken with the machine moving in only one direction, with no direction reversals while taking the error measurements. Measurements taken with the machine moving in the other direction are only used to determine an average backlash value. Backlash is the error induced by mechanical looseness or other mechanical slop that occurs when the machine reverses direction of motion. In other words, it is the amount that the motor will move to drive the machine in the opposite direction – before the motion is observed at the point of interest. In a machine tool, this is usually at the cutting tool tip. This type of compensation is generally quicker to measure and apply than bi-directional, but it is limited in some ways. It does work well for cases where the errors in motion are consistent in each direction, and the amount of backlash is consistent throughout the range of motion. Figure 3.1 illustrates an error-motion profile that would be suited for uni-directional with backlash compensation. Note the consistent error profile – the reverse motion plot looks like the forward motion plot shifted down a fixed amount. There is no variation in the backlash across the range of motion.

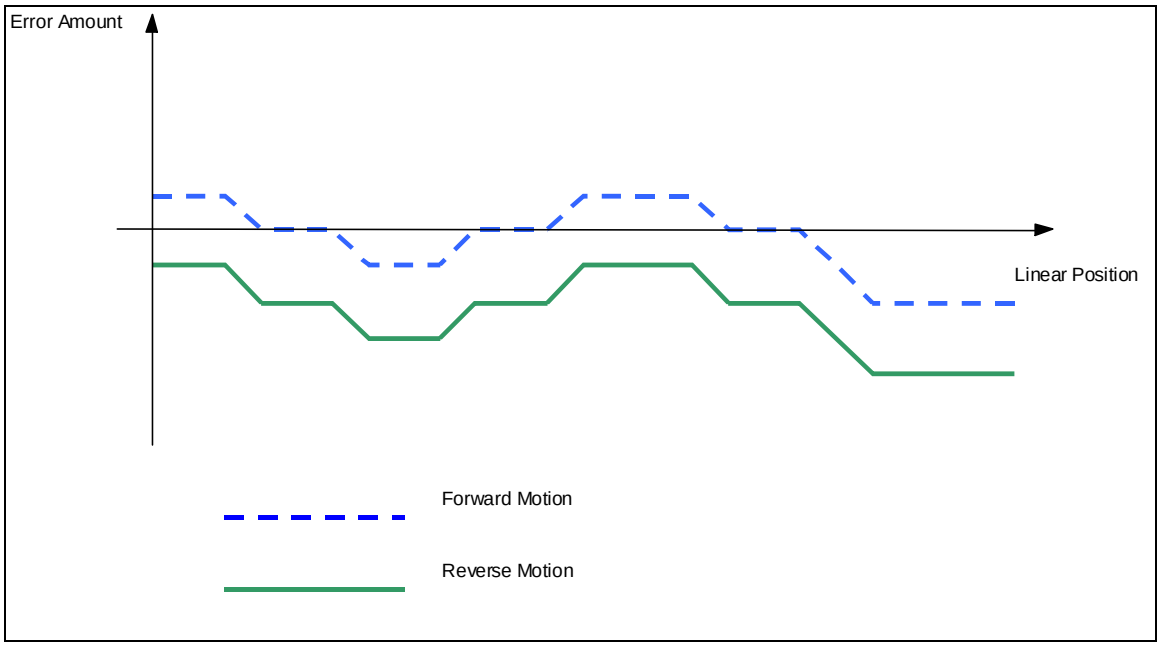


Figure 3.1 Error versus motion profile for uni-directional compensation.

Implementing uni-directional compensation from this data is simply a matter of creating a table that provides the amount of error at each fixed point of measurement. Each control has its own manner of entering the table, but the basic contents are listed in Table 3-1. Note that in this table, each error amount will be negated when entered as a compensation value. For example, if at position 1.0 inch there is an positioning error of +0.001 inches, the compensation amount will be –0.001 inches. The control will take this information and uses it any time an axis motion is programmed. The compensation values for any points between the listed fixed intervals are normally calculated by linear interpolation between the two adjacent fixed points. The compensation table should cover the entire range of motion for the axis, but if it does not, the compensation values for points outside the table are often held stable as the value at the nearest fixed point.

Position	Error
0.0”	-0.001”
1.0”	-0.001”
2.0”	-0.001”
3.0”	0.0”
4.0”	+0.001”
5.0”	+0.001”

Table 3-1 Error versus position compensation.

Implementing the backlash amount is a matter of calculating the average backlash over the range of motion and entering this amount into the control system. In Figure 3.1, the backlash is the difference between the two plots of forward and reverse motion. This number would be determined from the data and entered as specified by the control. The control would then use this value each time it reverses the direction of motion.

3.1.2 Bi-Directional Lead-Screw Compensation

In some situations, uni-directional compensation with backlash will not be capable of correctly compensating for the mechanical errors in the machine. If the error associated with the motion profile were similar to Figure 3.2, the variations in the overall error profile and the variations in the amount of backlash over the range of motion would make this difficult to compensate using that method. This would be a case for using bi-directional lead-screw compensation.

In this method, positioning errors are measured in both directions of motion and these values are independently recorded. These values would then be entered into the control as separate compensation values and the control would consider direction of motion when determining which value to use. In this method of compensation, there is no direct backlash value because that error is included in the bi-directional measurements. Once again, the exact method of entering the compensation values is control-specific, but the contents would appear similar to Table 3-3

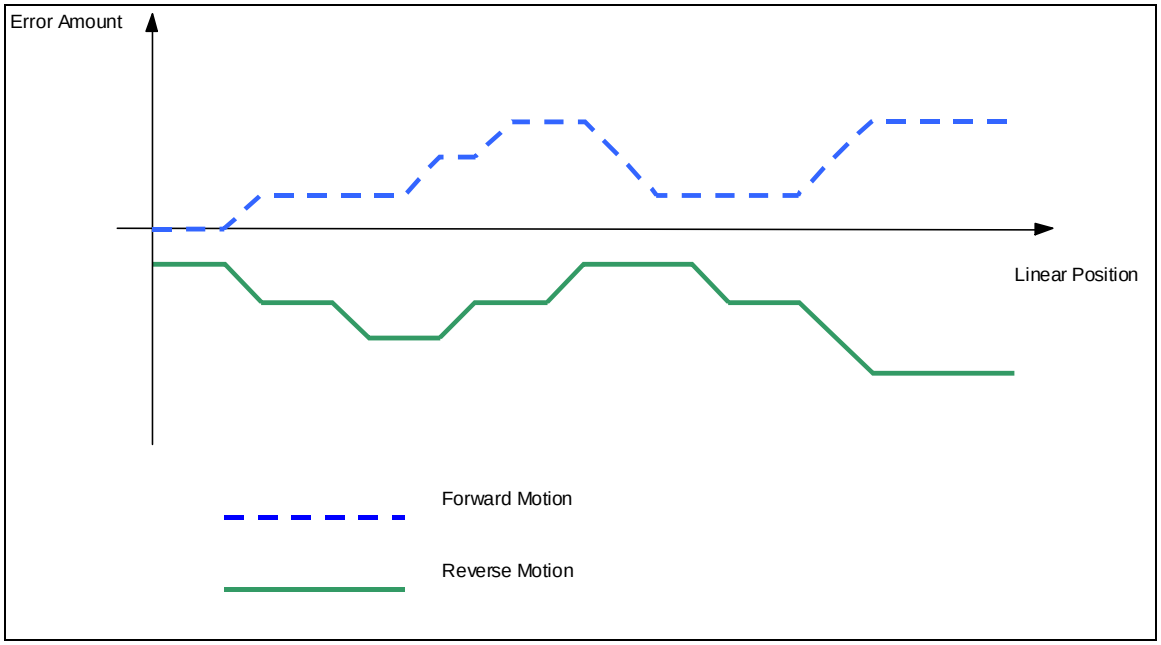


Figure 3.2 Error versus motion profile for bi-directional compensation.

Forward Position (inches)	Error (inches)
0.0	+0.001
1.0	+0.001
2.0	0.0
3.0	-0.001
4.0	-0.002

Table 3-2 Bi-directional lead-screw compensation values (forward).

Reverse Position	Error
4.0	-0.004
3.0	-0.003
2.0	-0.002
1.0	-0.001
0.0	-0.001

Table 3-3 Bi-directional lead-screw compensation values (reverse).

3.2 Applications of Neural Networks

The applications of neural networks are many and varied. The Matlab Neural Network Toolbox User's Guide (Demth & Beale) lists a variety of applications in Finance, Control, Manufacturing, and many others. In Artificial Intelligence: A Modern Approach, Russell and Norvig briefly discuss three specific applications of neural networks; pronunciation of written English text by a computer, recognition of handwritten characters, and driving a vehicle along a single lane highway.

3.2.1 Intelligent Control

One common area of application for neural networks is in intelligent control. Bauman, *et al.* present an intelligent control for load balancing the internal combustion engine (ICE) and the electric machine (EM) of a hybrid vehicle. In their problem, the ICE and the EM were mechanically connected in parallel to drive the vehicle. The goal of the controller was to distribute the load across the ICE and the EM to achieve maximum efficiency for the entire system. The authors present a discussion of the controller that uses a combination of neural networks and fuzzy logic to achieve the goals. In their discussion of the controller approach, they list five properties of the controller which make it appropriate for their application: a model-free approach, adaptability, fault tolerance, nonlinearity, and real-time operation. These properties were highly advantageous in their application which had the characteristics of being highly complex and difficult to mathematically model, experienced a high degree of variability and system noise in operation, was fundamentally nonlinear in its behavior, and required the ability to handle constant transient operations.

3.2.2 Analysis of Vibration Signatures

Another area of application for neural networks is in data analysis and classification or grouping. Alguindigue, *et al.* present a method of classifying vibration signatures for rolling element bearings in analyzing the bearings for defects. The idea is based on the fact that a real mechanical system in operation generates vibration. The amplitude and frequency spectra of the vibration changes as the mechanical system changes due to wear or mechanical defects. By analyzing the resulting vibration signatures, likely defects can be determined.

In this application, neural networks were used for compression of the spectral signatures and for classification of the compressed signature. For the compression phase of the system the authors described using a recirculation network. Compressing the

vibration signature results in mapping the spectra into a smaller dimensional space. This results in the network acting as a feature extractor which highlights the notable features of the data, while reducing the noise present in the signal. For the classification phase of the system, a backpropagation network was used to classify the signal into one of eight possible patterns – one indicating no fault, and the rest indicating one or a combination of predefined fault signatures.

3.3 Positioning Error Correction with Neural Networks

One example of using a neural network to correct for positioning errors in a machine tool is described in U.S. Patent 5,523,953 (USPTO). In it, the inventors outline a method of correcting for errors induced by temperature changes in the machine during its operation.

The machine is equiped with thermal sensors, and accurate measurements of the thermal deformations of the machine are taken while the temperature at each sensor is recorded. This data is prepared and presented to the network as training data. Once the trained network is completed, it then becomes a part of the machine controller. During operation, the thermal sensors are constantly monitored by the neural network and its output is routed to the machine control as an error signal to be included in the control of the axes of motion of the machine.

Chapter 4 – Experimental Setup

The setup of the experiments consisted of five major steps:

1. Configuring the laser interferometer system and the lathe control to measure the positioning accuracy of the axis under examination
2. Running the lathe and laser system to collect the measurement data
3. Designing the neural network and training it with the measured data
4. Extracting the key network parameters and programming the real-time calculations for integration into the control
5. Repeating the measurement test to determine real-world performance of the system

4.1 Configuring Laser and Lathe Control

The configuration of the laser and the lathe control is determined by the desired test data. While these are two separate systems, their configurations must match for a successful test. The lathe control must be set up to produce the pattern of motions expected by the laser measurement system.

4.1.1 Laser Measurement System Setup

The first step in configuring the laser interferometer system and the lathe for data collection is the physical alignment of the laser interferometer. In Figure 4.1 the physical arrangement for collecting data on the Z (long) axis is shown. In the lower right corner, the laser head can be seen which houses the laser emitter, beam splitter, interferometer, stationary reflector, and measurement electronics. Circled in the upper left is the mobile reflector mounted on a magnetic base to the tool turret. The mobile reflector is mounted such that, as closely as possible, it accurately reflects the motion of the tool – which is the point of greatest interest in this experiment.

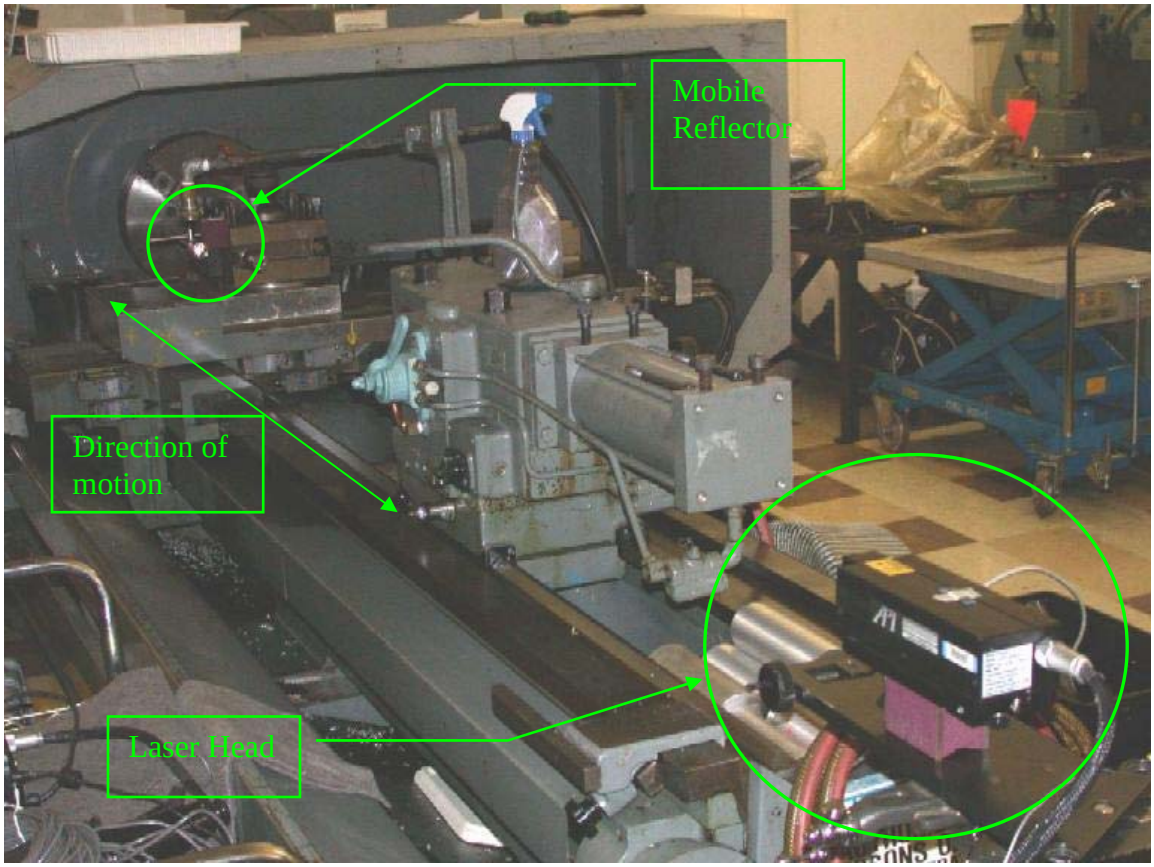


Figure 4.1 Laser alignment for Z axis.

Once the system components are arranged for the desired test, they are precisely aligned horizontally and vertically parallel with the Z axis so that linear motion in the Z axis is accurately measured by the laser interferometer system. This is readily determined by monitoring the beam-strength of the laser as the machine is moved through its entire range of motion – as long as the beam-strength remains in an acceptable range throughout this motion, the alignment is sufficient for correct measurement.

With the physical arrangement and alignment of the laser system complete, the final step in configuring the system is to set the software parameters for the test to be conducted. This is done on the Test Setup screen in the API 5/6-D laser system software. While many tests were conducted with a variety of parameters, the setup for the primary test for the Z axis is shown in Figure 4.2. The key parameter to note is that this was a bidirectional test, with readings taken every 0.025 inches (40 readings per inch).

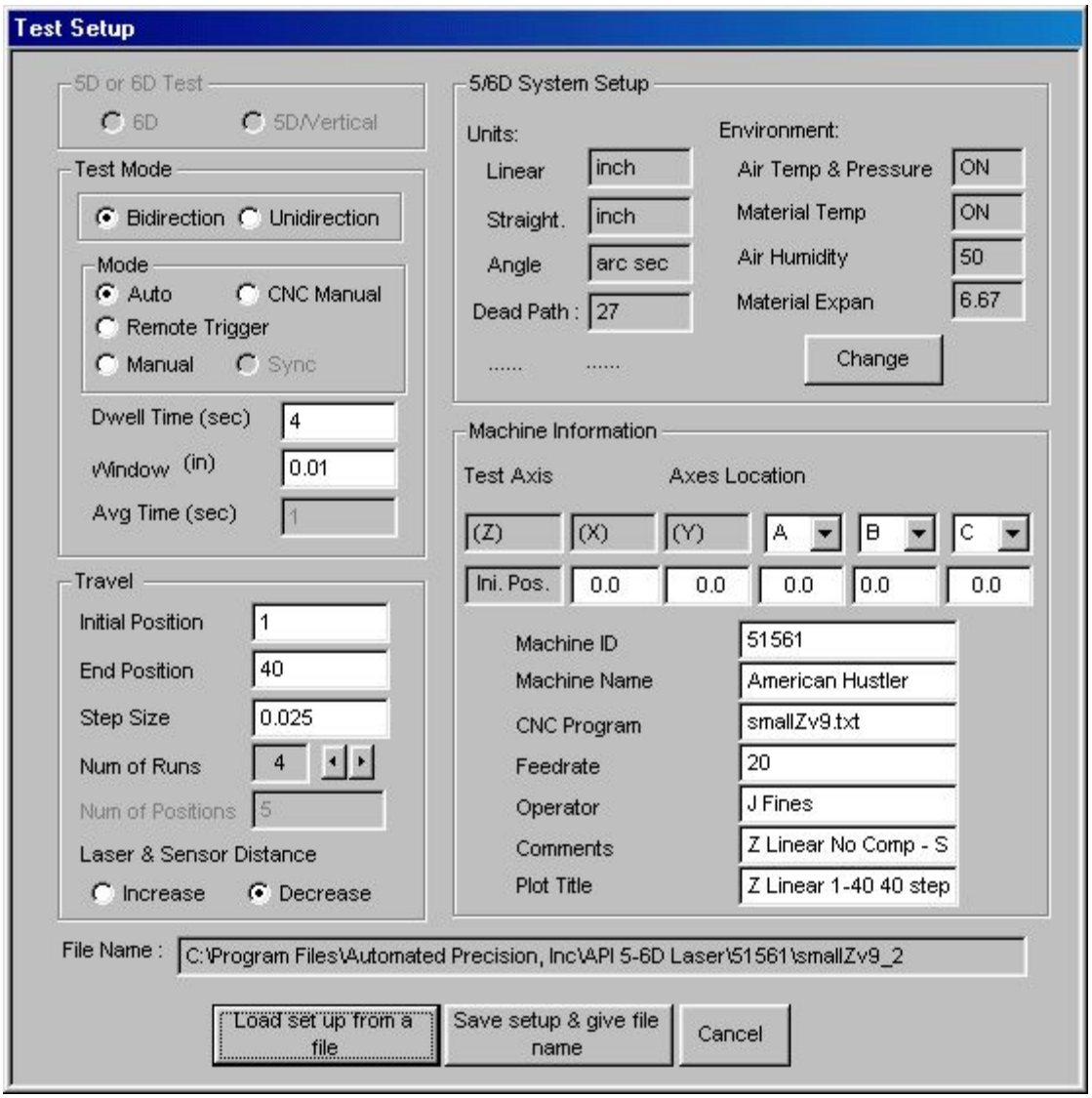


Figure 4.2 Test setup for primary Z axis test.

4.1.2 Lathe Control Setup

Configuring the lathe control is accomplished by writing a part program that produces the pattern of motion expected by the laser measurement system. For this experiment, OpenCNC's VB Macro feature was used for the required programming. In this setup, a small part program was written in which the key test parameters were defined. The program then called a VB macro to error check the input parameters and produce a subroutine that contained the G&M code required to produce the desired pattern of motion. Finally, the program called the subroutine to run. This allowed for major changes in test parameters with only quick, minor edits to the basic part program – the macro would then be recalled to regenerate all of the necessary code, instead of rewriting all of the code each time a test needed to be changed.

Figure 4.3 contains the part program used for control of the lathe under test. The first nine lines establish the parameters for the desired test. The "macrofile…" line calls the VB macro and passes in the test parameters for processing. The M200 is a part program syncing command in this control, and the "subfile…" line calls the subroutine produced by the VB macro. Finally, M30 is the end-of-program command.

Figure 4.3 The part program for lathe control.

```
vb #1 = 3       'Axis 1=X, 2=Y, 3=Z
vb #3 = 1       'Starting Position in Machine Coordinates
vb #4 = 7       'End Position in Machine Coordinates
vb #5 = 1       'Increment of motion between point
vb #6 = 2       'Dwell time at points
vb #7 = 3       'Number of complete runs
vb #8 = 50      'Feedrate for motion
vb #9 = .1      'Overshoot for backlash moves

macrofile "LaserMotion" A=#1 S=#3 E=#4 I=#5 T=#6 N=#7
F=#8 B=#9 D=0

M200
```

When the listed part program is run and the macro is called, the first function it performs is to error-check the input parameters. The macro looks for errors in requested axis of motion, requested limits of travel, and requested increment of motion – along with others. The first part of Appendix B includes a complete listing of the macro source code. After the input parameters have been checked, the G&M code is generated and is output to a specified filename and location. The second part of Appendix B contains a complete listing of the output from running the part program listed in Figure 4.3. After the macro has generated all of the necessary code, it exits and returns control back to the part program, which then calls the subroutine file to control machine motion.

4.2 Data Collection

The data collection step of the experiment is a fairly simple process. Once the laser and lathe controls are configured correctly, the lathe is put into an "AUTO" mode. This is where the control of the lathe is passed to the part program currently loaded. When the control's "CYCLE START" function is activated, the lathe proceeds along the programmed pattern of motion. The laser control observes this motion and, based on the parameters entered for the test, automatically takes measurements and collects the resultant data. At the conclusion of the test, the laser system writes out the data files that contain all of the data collected, along with some analytical results. Figure 4.4 is an extract of one laser data file with a few sample lines of data.

Figure 4.4 Laser data example.

The raw data collected by the laser measurement system include the commanded

Position	LasRead	LinearError	Status	AirTemp	AirPress	MatTemp
1.00000	0.000010	-0.000010	0	70.37	29.25	71.49
2.00000	-0.999691	-0.000309	0	70.36	29.25	71.49
3.00000	-1.999600	-0.000400	0	70.35	29.25	71.49
4.00000	-2.999554	-0.000446	0	70.35	29.25	71.49

position, laser reading, calculated linear error, status, air temperature, air pressure, and material temperature. This data can be taken off-line for analysis or reviewed in some detail on the laser measurement system.

4.3 Neural Network Design and Training

As all of the neural network design and training was done using Matlab and its associated Neural Network Toolbox, this portion of the experiment centers on the use of various Matlab functions to accomplish specific tasks.

After the data have been collected from the laser measurement system, they are taken off-line for processing. The data file would contain two sets of data. The first is the raw data collected during the entire test. The second (in the case of a multiple run test) would contain the average of the error value at each individual data point over all of the runs in the test. Using Matlab's internal *File:Import Data* functionality, all of the data is manually imported into two matrices – one of raw data, the other of averages. These matrices would become the basis for neural network training and testing. The average values were used to train the network, and the raw data were used to test the resultant network.

Generating and training a neural network, and displaying the results of test data was accomplished using Matlab's *m-file* scripting capability. As this process turned out to be highly repetitive in trying and testing different network structures (numbers of neurons, numbers of layers, etc.), using a script allowed for quickly editing a few key lines and rerunning the script to test an entirely different network. Appendix C includes the complete script. The key line in the script generates a new network and specifies the network architecture in terms of the number of layers, neurons in each layer, and the transfer function to be used in each layer. It also defines what training function is to be used and the evaluation function used in training and measuring performance. For example, the script line:

```
net = newff([0 1;1 40],[60 1],{'logsig','purelin'},'trainlm','learngdm','sse');
```

generates a new network with two layers. One layer will be made up of 60 neurons with the *logsig* transfer function, and the second layer will be a single neuron with a linear transfer function. Once edited, the script line:

```
net=newff([0 1;1 40],[40 20 1],{'logsig','logsig','purelin'},'trainlm','learngdm','sse');
```

will generate a three layer network, a 40 neuron *logsig*, a 20 neuron *logsig*, and a single linear neuron. The last few lines in the script plot the results of testing the new network on the same figure as the original test data. This provides a pictorial view of the results for evaluation of network performance.

After evaluating the network performance and finding a suitable network for the problem, the network structure is saved to allow for extracting the key parameters to use in programming the real-time calculations.

4.3.1 Matlab Implemented Training Algorithms

Training a neural network in the Matlab environment is normally accomplished using one of the built-in training functions that are included in the Neural Network Toolbox. There are sixteen different training functions implemented and included as part of the toolbox (Demuth and Beale). While each of the training algorithms could be used

on any problem, they are each more suited for certain classes of problems and network architectures. One of the issues was to determine, early on, which algorithm was best suited for the problem at hand and the network architectures under investigation.

For feed-forward, backpropagation networks, the default training algorithm is the Levenberg-Marquardt algorithm. According to Demuth and Beale this algorithm is well suited to function approximation problems with networks of moderate size and number of parameters. It is also well suited to problems that require the approximation to be very accurate. Other algorithms that were considered to work well in function approximation problems are the Scaled Conjugate Gradient and the BFGS Quasi-Newton algorithms (Demuth and Beale).

Demuth and Beale present a discussion on speed and memory comparisons of the various training algorithms for a set of sample problems. This discussion supports the conclusion that the Levenberg-Marquardt (LM) algorithm is the best available in this toolbox for the class of problems that are function approximation problems on small to moderate sized networks. As the problem under study fits that description, these recommendations were considered and the conclusions on the actual data for this problem were validated.

The work in this thesis included a function approximation problem that required the final error to be reduced to a very small value and, in general, the networks were of moderate size. A series of ten tests each were conducted on four of the training algorithms where a newly created network was trained against the data collected from the laser-interferometer measurements of the lathe. In each of the tests, the entire network was newly created and randomly initialized in the default manner of the toolbox. The training parameters were set to train for 1000 epochs, and the goal of reducing the error to 4×10^{-8} was set, as measured by the Sum Squared Error (SSE) function. In each of the tests, the time to complete 1000 epochs (or reach the training goal) was tracked, along with the final error reached. The results of the test are summarized in Table 4-1. While the LM algorithms was the slowest of the four tested, it was the only algorithm capable of reaching the training goal specified and it still completed in a reasonable amount of time. The accuracy reached in each trial is shown in Figure 4.5. In this figure, and noting that the accuracy level is plotted on a logarithmic scale, it can be seen how much better the LM algorithm is at reaching a very small error level as required by this application.

Training Algorithm	Average Time to 1000 Epochs (seconds)	Final Error Measure (inches)
TrainRP	23.4	4.67E-4
TrainSCG	44.9	3.67E-4
TrainBFG	54.3	1.23E-7
TrainLM	65.8	4.81E-8

Table 4-1 Comparison of training algorithms

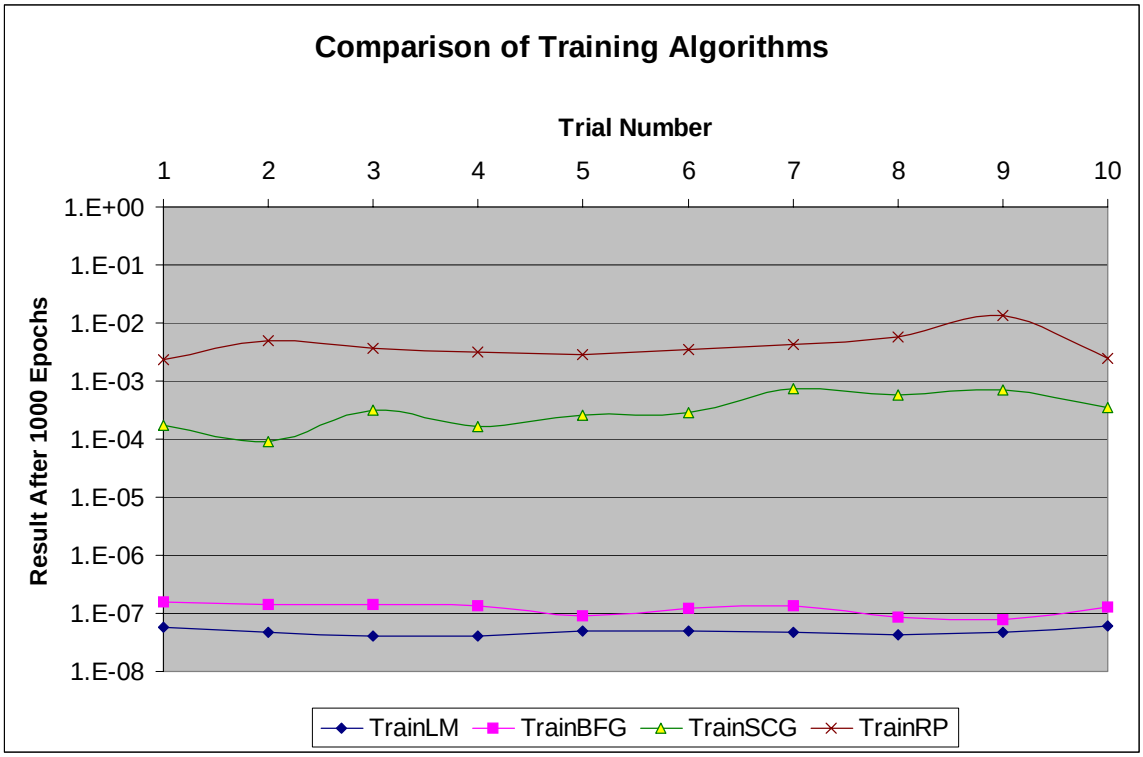


Figure 4.5 Comparison of training algorithm accuracy.

After reviewing the information on training algorithms, and conducting the test discussed above on actual data from this experiment, it was concluded that the Levenberg-Marquardt was the best training algorithm to use for this application. All further network development and training for this experiment were done using the LM algorithm.

4.4 Programming Real-Time Calculations

Programming the real-time calculations for integration into the lathe control system is accomplished using the capabilities provided by MDSI's Application Programming Interface (API) for hard, real-time program development (MDSI, *API Manual*). The programming was done in Microsoft Visual C++ and referenced libraries provided as part of VentureCom's Software Development Kit (VenturCom). Programming the real-time calculations first required the extraction of the key network

parameters from the network developed in Matlab described in the previous step. Then, those parameters had to be coded into the program that produced the actual calculations of compensation values to integrate into the control.

4.4.1 Extraction of Network Parameters

At the point where the real-time calculation is being programmed, the structure of the neural-network has been designed, and all of the values within the network have been determined by the training process. These key parameters are the weights applied to the inputs, the bias values applied to the input neurons, the weights applied to any values passed to neurons in the hidden layers of the network (these are usually known as layer weights), the bias values applied to hidden neurons, and the transfer functions of all the neurons in the network. Since the design and training of the network were accomplished within the Matlab environment these parameters are all stored as properties of the network object created in Matlab.

As is often the case, the easiest method of repeatedly accomplishing a task in Matlab is to write an m-file script to do the work. For example, within Matlab, if there exists a neural network object named "net", it will have properties referred to as "*net.IW{1}"* which contains the matrix of input weights for the network. Other properties are *net.b{1}*, *net.b{2}*, and *net.LW{2}* which are the bias values for layer one, bias values for layer two, and the layer weights applied to inputs to layer two, respectively. The script included as Appendix D was used to extract the parameters of the networks. Also included in Appendix D is the results of running the script which shows the output of the m-file. This output was formatted to be as close as possible to the text necessary to define constants in a C program – which is how the values were used in programming the calculations.

4.4.2 Calculation of Compensation Values

Determining the compensation values to be used in the system control is done by calculating the feed-forward action of the neural network defined previously. This calculation must be designed to meet the constraints of the tools used – it must use only functions supported by the VentureCom SDK (VCI), and must be simple and quick enough to meet the timing constraints established in the machine control. In the case of this system, the control was queried for inputs and new compensation values were calculated and returned every 10 milliseconds.

As an example, one network structure that was used in this system was made up of two simple networks. Figure 4.6 includes a diagram of the network structure with weights and bias points, where I1 and I2 are input neurons, I1W1…I1Wn and I2W1…I2Wn are input weights, H1…Hn are the hidden neurons, bH1…bHn are the bias values for these neurons, LW1…LWn are the layer weights, O1 is the output neuron, and bO1 is the bias value for that neuron. The first network had two input neurons, an eight-neuron hidden layer using the logsig transfer function, and a single output neuron with a linear transfer function. The second network also had two input neurons, a 30-neuron hidden layer using the logsig transfer function, and a single, linear, output neuron. The outputs from the two simple networks were added to make the final compensation value to be returned to the control.

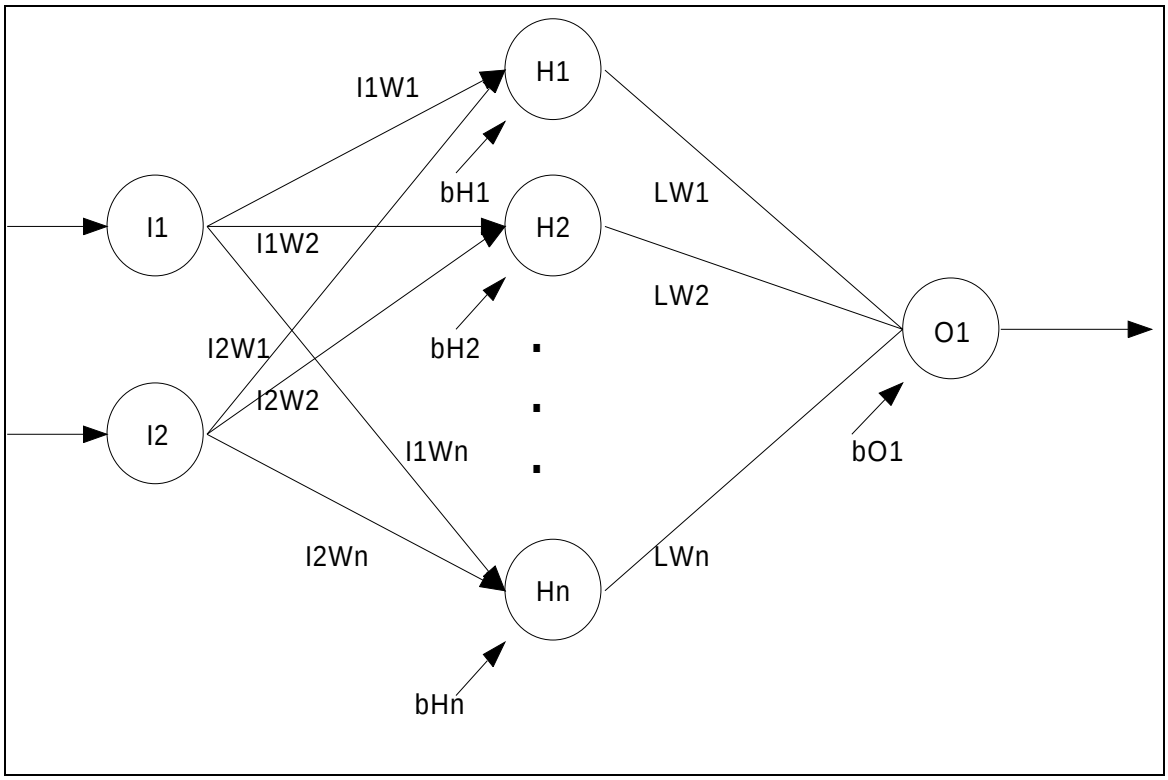


Figure 4.6 Neural network with weights and biases.

Following the diagram in Figure 4.6, the complete calculation of the feed-forward path of the network can be written as:

$$\begin{aligned}&\log sig\left(I1 \bullet I1W1 + I2 \bullet I2W1 + bH1\right) \bullet LW1\\ &+ \log sig\left(I1 \bullet I1W2 + I2 \bullet I2W2 + bH2\right) \bullet LW2\\ &\vdots\\ &+ \log sig\left(I1 \bullet I1Wn + I2 \bullet I2Wn + bHn\right) \bullet LWn\\ &+ bO1\end{aligned}$$

or more succinctly as:

$$\sum_{j=1}^{n} \log sig\left(I1 \bullet I1Wj + I2 \bullet I2Wj + bHj\right) \bullet LWj + bO1$$

where n is the number of neurons in the hidden layer, and the transfer function logsig(x) is defined as:

$$\frac{1}{1+e^{-x}}$$

However, by taking advantage of prior knowledge of the inputs, this can be simplified. In the case of this system, the first input, I1, is either 0 or 1 indicating forward or reverse motion respectively. In the case of forward motion where I1 is 0, the calculation can be reduced to:

$$\sum_{j=1}^{n} \log sig\left(I2 \bullet I2Wj + bHj\right) \bullet LWj + bO1$$

In the case of reverse motion where I1 is 1, and noting that the input weights and bias values are predetermined, the calculation can be reduced to:

$$\sum_{j=1}^{n} \log sig\left(bWHj + I2 \bullet I2Wj\right) \bullet LWj + bO1$$

where $bWHj$ is the predetermined value of $I1Wj + bHj$.

The complete C program code for this calculation is included as Appendix E. By reviewing that code, it can be seen how all of the calculations were accomplished using simple mathematical functions and basic looping structures. These were required to stay within the supported functions of the VentureCom SDK. It can also be seen how all required values from the network, and any additional constants that could be derived from those basic values, were predetermined and loaded into the program code as constants to be used in the calculations.

4.4.3 Control Integration and Measurement Testing

Integration into the control requires the installation of the compiled program onto the machine under test, and editing the control system's configuration files to include this program in the startup of the machine control. The exact details are specific to MDSI's OpenCNC product and can be found in the OpenCNC 6.0 API Manual and the sample programs provided with the OpenCNC API.

The program was configured to run on a strict timing cycle of 10 milliseconds. In each cycle the following actions take place:

1. Control variables are read to determine the current axis position.
2. Direction of motion is determined from previous position to current position.
3. Current position is recorded for motion comparison in the next cycle.
4. Input values are scaled and applied to the compensation calculations.
5. Compensation values are calculated based on the inputs presented.
6. Compensation value is returned to the control for immediate application to the system.
7. Wait for the next timing mark to repeat cycle.

The complete routine is programmed to run as an infinite loop. Once the loop is entered, the cycle will be repeated on each timing mark forever or until the process is killed by shutting down the control system.

Once the compensation routine is installed in the control system and the control is restarted, the measurement test outlined previously is repeated under the exact same setup conditions. The results from this test are then used to determine the effectiveness of the compensation system as applied to the real-world system.

Chapter 5 – Results

The results for this thesis are based on the laser interferometer data taken with the machine in three different states. In the first state, all positioning error compensation systems are turned off or removed from the control system. This represents the basic mechanical condition of the machine and the data taken in this state is the starting point for any compensation system applied to the machine. The second state, the controls standard bi-directional leadscrew compensation is applied as described in previously. This represents the "state-of-the-art" in machine tool positioning error compensation as normally seen on the shop floor today. While there are more advanced methods of positioning error compensation, they are normally not part of a commercially available control system and therefore are not often seen in a modern machine shop. In the third state, the results achieved by the neural network-based positioning error compensation system are developed. This includes a discussion of the various network architectures examined as part of this project.

5.1 Uncompensated Positioning Errors

Initial attempts at measuring the linear positioning error were made at the usual settings for a large machine tool. Three repeated runs were made with measurements taken at one inch intervals over the full range of motion of the Z-axis of the machine. However, after getting some inconsistent results and taking a closer look at the condition of the machine, the final measurements were made as a set of four repeated runs with the measurement interval set to 0.025 inches (40 readings per inch). The range of measurements was reduced to 39 inches (from the 1 inch point to the 40 inch point). This was done to reduce the time involved in taking the measurements. The results of these measurements are shown in Figure 5.1.

In Figure 5.1, each data point was measured in the forward and reverse directions. The individual data points were averaged and presented. From these values, the maximum average error is calculated to be 0.005582 inches. This is the difference between the highest average value and the lowest average value across the entire range of measured motion. The lower set of values are those taken while the machine was moving in the forward direction (stated position was increasing in value). The upper set of values are those measured while the machine was moving in the reverse direction. The data plot graphically illustrates the three major issues that any positioning error compensation system must address to improve the performance of the machine.

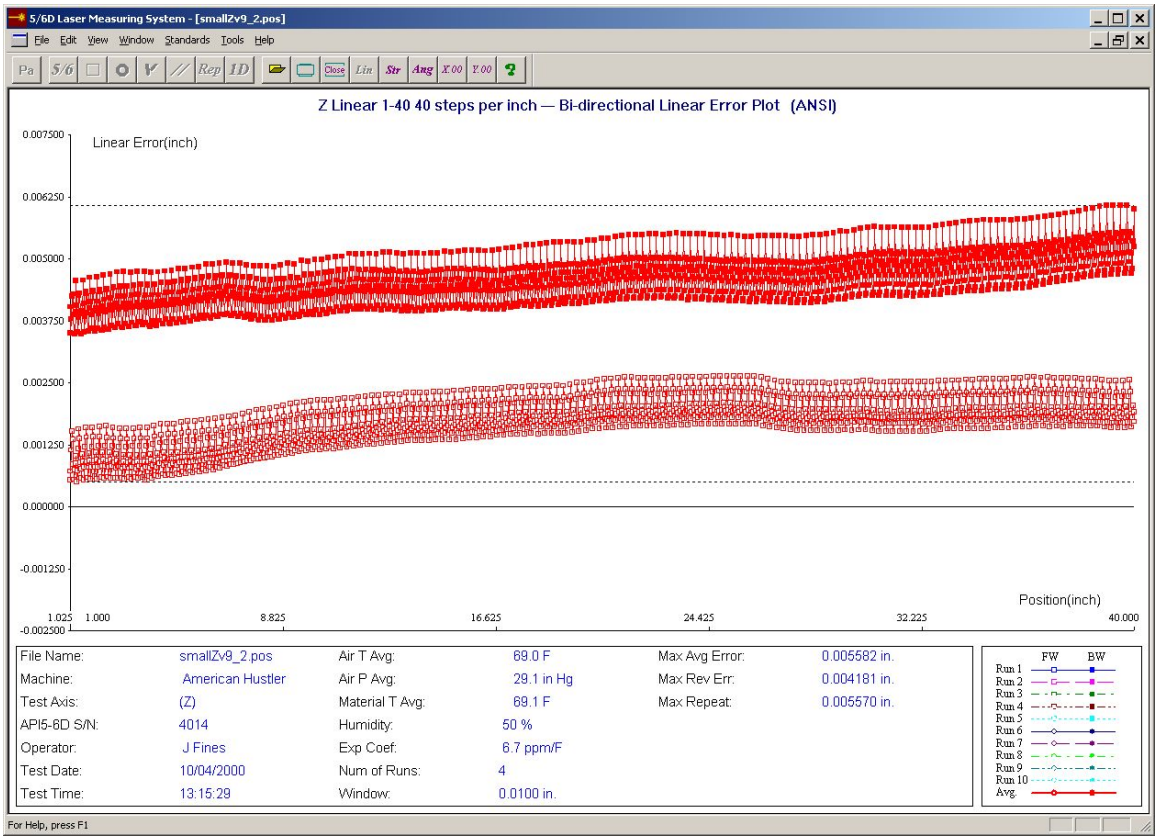


Figure 5.1 Uncompensated positioning error average results.

The most glaring problem on this machine is the difference in positioning from between the forward and reverse motions – commonly known as backlash. As listed in the lower portion of the plot, the maximum reversal error is 0.004181 inches. This is the maximum error at any one measurement point between the two directions of motion. The next problem is the general trend of the machine to position "long" as the desired position increases from 1 inch to 40 inches. In the left side of the graph, positioning errors are less than the errors on the right side of the graph. By analyzing the numerical measurements associated with this graph, this contributes approximately 0.0012 inches to the overall error. The last error is the most problematic and is very difficult to see in this plot. Figure 5.2 is an expansion of the plot around measurements from a single direction of motion over a space of 2.5 inches (from 1 inch to 3.5 inches). This highlights the cyclic error on this machine.

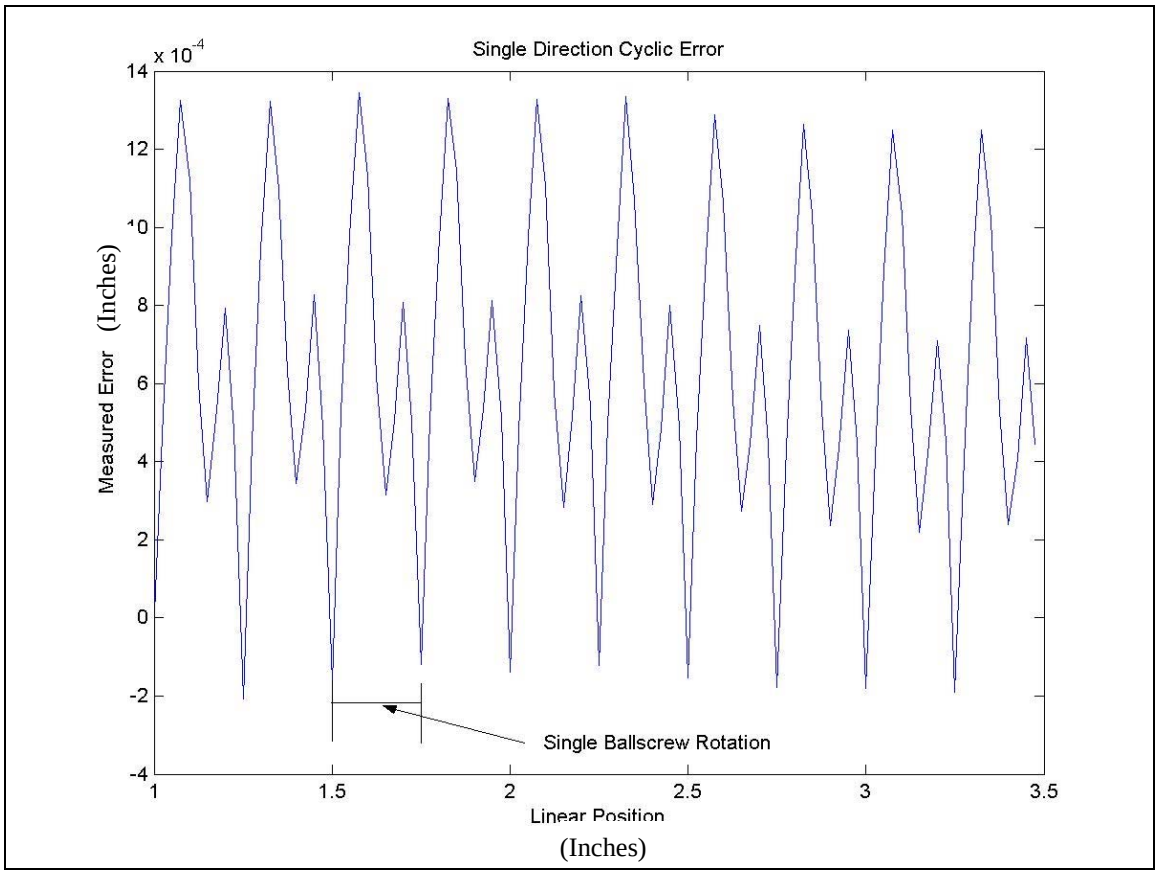


Figure 5.2 Single direction cyclic error.

The major mechanical contributor to linear positioning error on this machine is the leadscrew assembly (Figure A.3 and Figure A.4). In this case, the pitch of the leadscrew, or the amount of linear motion generated by a single revolution of the leadscrew, is 0.25 inches. Examining Figure 5.2 the rather severe cyclic error can be seen for each revolution of the leadscrew. It measures approximately 0.00145 inches peak-to-peak and has a double-humped pattern for each turn of the screw. This is what causes the forward and reverse runs in Figure 5.1 to appear as broad swipes across the graph instead of the expected single line of points. As previously mentioned, this contributes approximately 0.00145 inches to the overall positioning error, but more importantly is very difficult to compensate for in many control systems.

The data shown represents a machine with some significant mechanical flaws affecting its performance. Any effort truly directed at improving its performance (for example preparing it for use in a production environment) would have to begin with mechanical repairs of these flaws. However, in its current state it is a good test of compensation systems that may be applied.

5.2 Bi-Directional Leadscrew Compensation

Bi-directional leadscrew compensation was applied as described previously. The average error values recorded in the uncomped measurements were negated and input into the control system as leadscrew compensation values. Since this compensation system requires that the values be at a fixed interval over the entire compensated range of

motion, and in trying to make this work as well as possible, compensation values were applied just as measured – 0.025 inch interval from 1inch to 39 inches. The overall results are displayed in Figure 5.3.

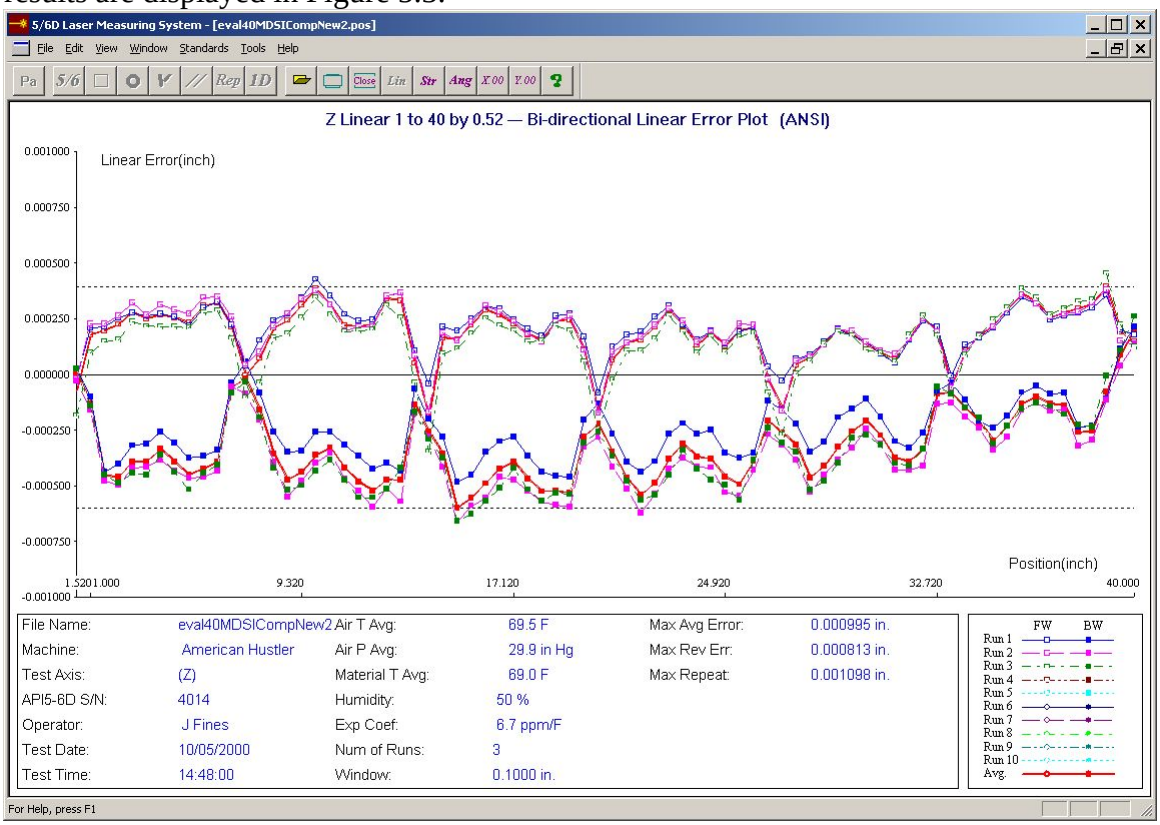


Figure 5.3 Bi-directional leadscrew compensation results.

The measurement runs that were used to test this compensation system were set to an interval of 0.52 for the range of 1 inch to 40 ionches. This made for a much quicker check (the umcompensated four passes took almost 20 hours to run), and by setting the increment to a value that was not evenly divisible by the pitch of the leadscrew it still brought out the cyclic errors as the leadscrew stopped in different rotational positions.

As shown in Figure 5.3, the bi-directional compensation system was able to correct for the majority of the errors. The overall maximum average error was reduced to 0.000995 inches – 17.8% of the uncompensated error. Backlash as measured by the maximum reversal error has been reduced to 0.000813 inches – 19.4% of the uncompensated value. And, while the cyclic error is clearly visible, it has been reduced to approximately 0.0005 inches or less – about 35% of the uncompensated value.

These results represent what is generally achievable as state-of-the-art in production machinery. This machine is compensated at a very close interval (typical for a machine this size is 1 inch intervals). But the control system in use here allows for this, and also allows for the large number of points that must be entered with this small interval.

5.3 Neural Network-Based Error Compensation

A neural network based positioning error compensation system was constructed as previously described, and the post-compensation test was rerun with the same parameters as the bi-directional leadscrew compensation was tested. This allowed for a direct comparison of the results of the two methods. The overall results can be seen in Figure 5.4.

Figure 5.4 shows that the neural network error compensation routine was able to correct for the majority of the positioning errors in the machine. The overall maximum average error was reduced to 0.000933 inches – 16.7% of the uncompensated error. The maximum reversal error was measured at 0.000614 inches – showing a reduction of the backlash error to 14.7% of the uncompensated error. Under close examination of the data plot, the cyclic error can still be seen as peaks in the error measurements, but it has been reduced to less than 0.0005 inches, or about 35% of the uncompensated measurement.

The laser interferometer tests results are for the three compensation systems (uncompensated errors, bi-directional leadscrew compensation and neural network-based compensation) are summarized in Table 5-1. Briefly, it can be seen that either compensation system made vast improvements in the accuracy of the machine, and the results of the two systems are comparable.

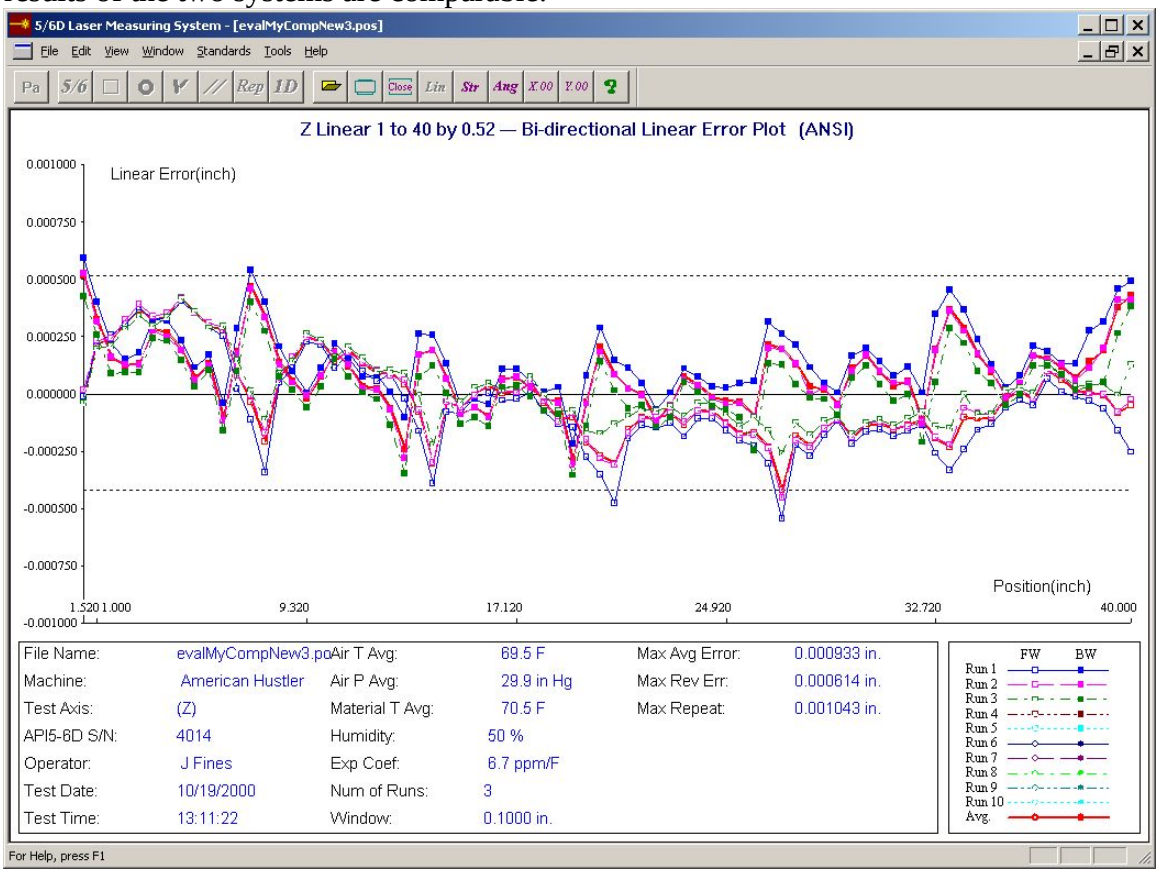


Figure 5.4 Neural network error compensation results.

Error Measured	Uncompensated	Bi-Directional	Neural Network
Max Average Error	0.005582	0.000995	0.000933

(inches)			
Max Reversal Error (inches)	0.004181	0.000813	0.000614
Max Repeat (inches)	0.005570	0.001098	0.001043
Cyclic Error (inches)	~0.00145	~0.0005	~0.0005

Table 5-1 Positioning error summary.

5.4 Network Training Results

In the course of this project, many different networks were tried and examined. This trial phase quickly focused on three network architectures that produced reasonable and consistent results for the problem under study. For all the networks, the inputs consisted of the linear position of the machine, a direction-of-motion indicator, and a calculated value indicating where the machine was currently located - within a single revolution of the leadscrew. Also for all the networks, the output was a single value of the currently required positioning error compensation value to be applied to the control system. The variation between the networks was in the hidden layers of the network and the connectivity between the neurons.

The first network consisted of a single, 40-neuron, hidden layer between the inputs and outputs as diagrammed in Figure 5.5 (the diagram is an extract from the view function in the Matlab Neural Network Toolbox graphical network editor *nntool*). All the hidden layer neurons have the logsig transfer function, with a single linear neuron in the output layer. The network is fully connected from inputs to the hidden layer to the output layer.

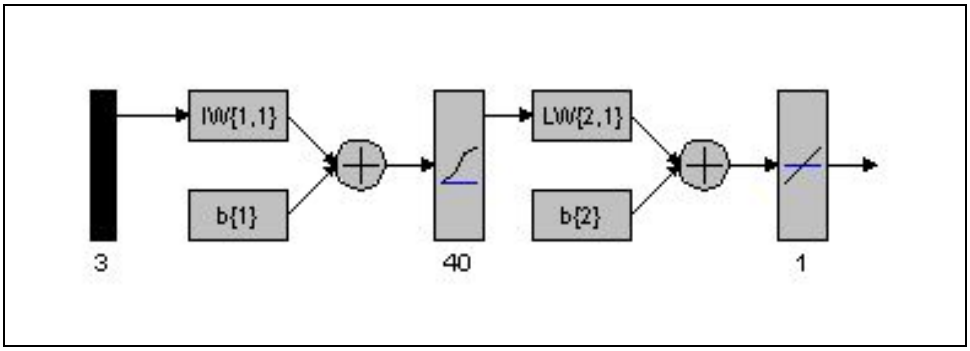


Figure 5.5 Single hidden-layer network.

Typical training results for the trials of this network are shown in Figure 5.6. After approximately 7500 epochs, which took just over two hours on the test computer, the network was normally able to get the error measure down to a value of approximately 7.00e-7 as measured by the Sum Squared Error performance function. With this level of performance, a maximum error of 1.3498e-4 was reached for any single point in the range of operation.

The second network architecture that was closely studied consisted of two hidden layers between the inputs and outputs. Both of these layers were made up of logsig

neurons, with 30 neurons in the first layer, and 10 neurons in the second layer. The diagram of this network is shown in Figure 5.7.

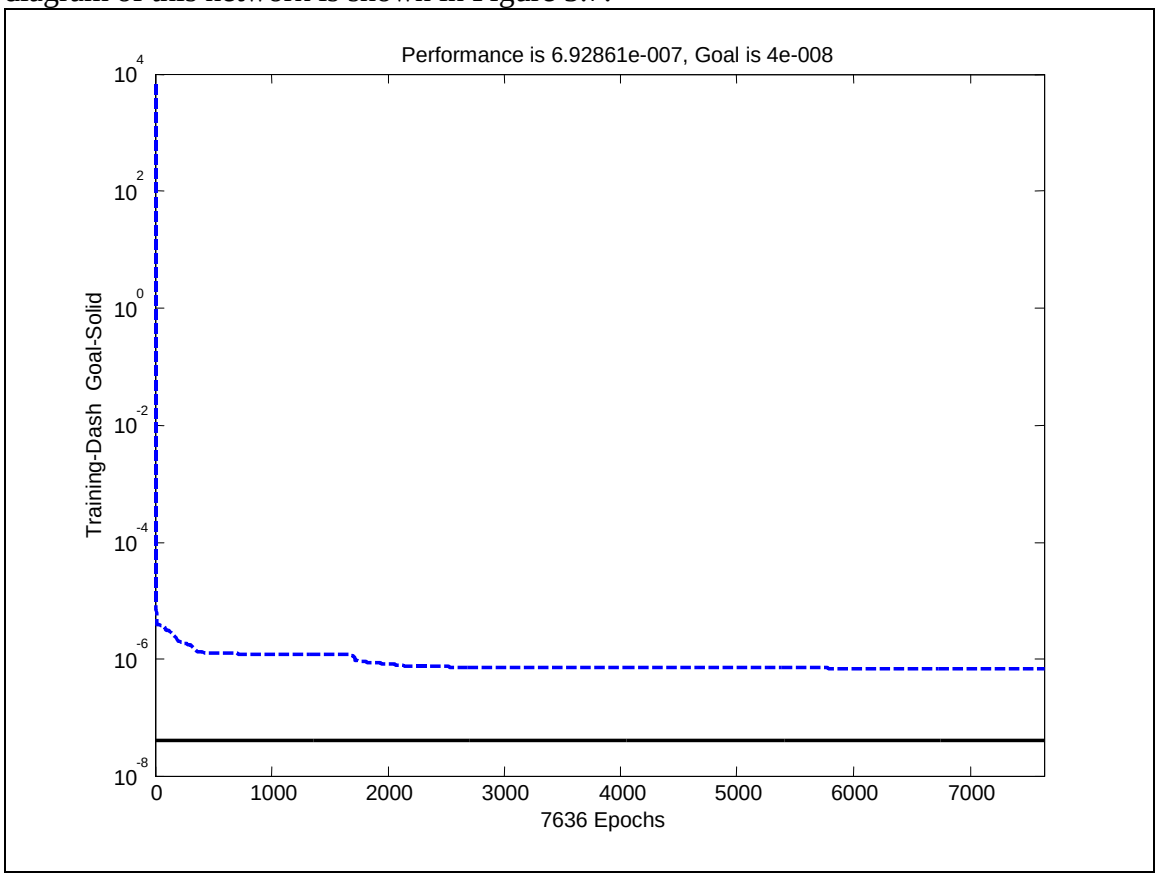


Figure 5.6 Training results for the single-layer network.

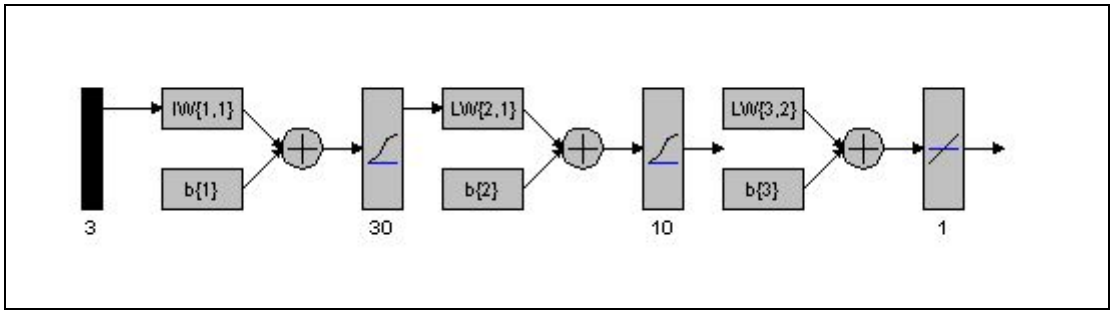


Figure 5.7 Multi-layer network diagram.

The training results for this network were typically in the same range as the 40-neuron single-layer network. The training results are shown in Figure 5.8. As shown, the network was able to achieve a comparable level of performance, with the training time for this network taking about 2.5 hours (about 25% longer than the single-layer network), but only 3200 epochs – or less than half the single-layer network.

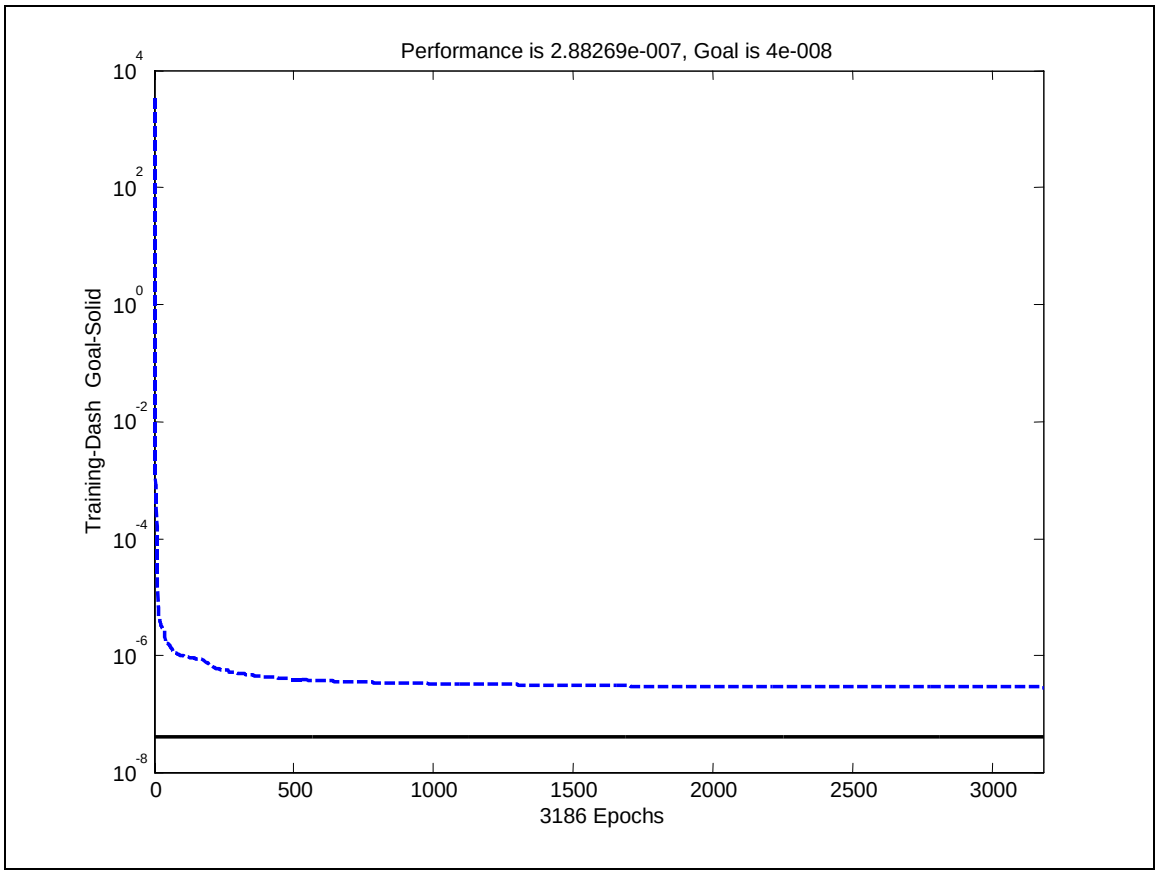


Figure 5.8 Training results for the multi-layer network.

The last network architecture closely examined was a combination of two networks. This architecture was arrived at by considering the nature of the problem. From the uncompensated positioning error results (Figure 5.1 and Figure 5.2) the errors can be classified into two categories – large or gross errors that occur over the range of motion and the cyclic error that occurs within the space of a single revolution of the leadscrew. The large errors would include the general trend of the positioning error as the machine moved from 1inch to 40 inches and the backlash error as the machine changes direction of motion. By considering, and compensating for, these problems separately, two networks can be created. One would be designed to address the large error issues, and the other would focus solely on the cyclic error problem. The system would be constructed so that, just prior to applying the compensation to the control, the network outputs would be added to make a single compensation value to be used.

The resulting networks were two single-layer networks, one with 30 neurons in the hidden layer, and one with eight neurons in the hidden layer. In both cases, the network is as diagramed in Figure 5.5, only the number of hidden-layer neurons would change. The training results for the 30-layer network are shown in Figure 5.9. For the data presented, this network was able to reduce the error measure (as before, the performance function was the SSE) to under 4e-8 in less than 600 epochs. The training time for this network was just under 42 seconds.

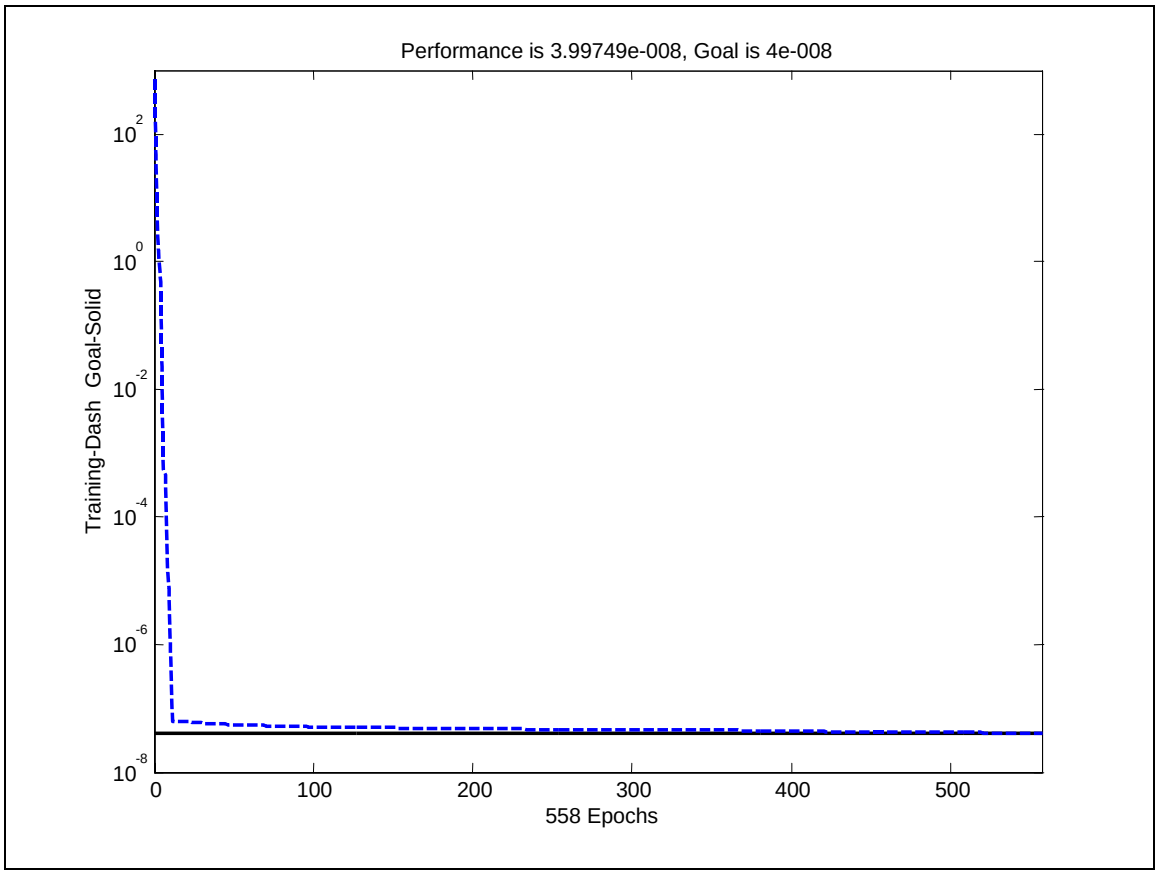


Figure 5.9 Training result for large-error network.

The training result for the eight-neuron network is shown in Figure 5.10. This network was designed to address the cyclic errors seen in the data. For this, the network was able to reduce the SSE error measure to under 9e-9 in less than 300 epochs. The training time for this network was under 15 seconds on the test computer.

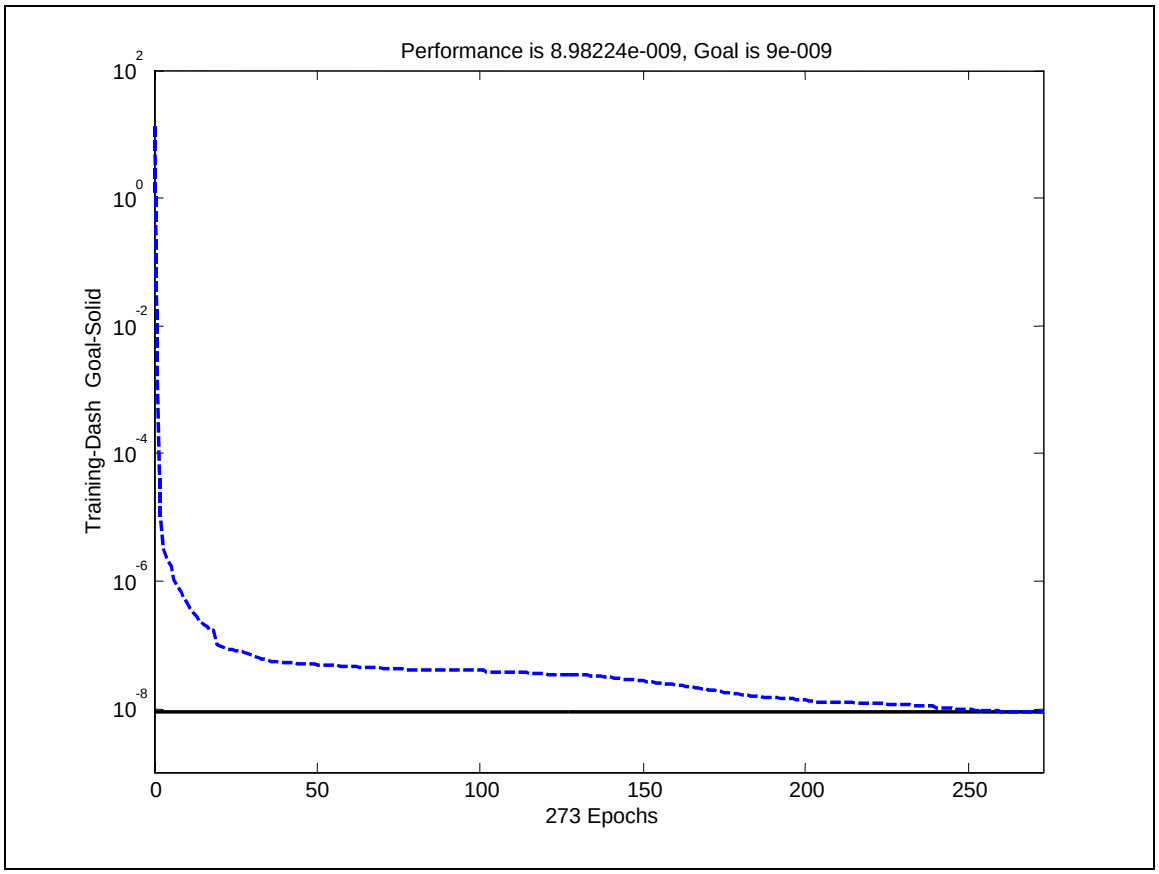


Figure 5.10 Training result for cyclic error network.

By taking advantage of some pre-training analysis, it was possible to reduce the training time from over two hours to just a few minutes for the problem under study. This was achieved while maintaining (or improving) the ability of the system to approximate the function as necessary to generate the compensation values. This multi-network architecture was the network architecture that was implemented and tested.

Chapter 6 - Conclusions

6.1 Contributions

In the course of this project, it was shown that a neural network is a viable technology for calculating compensation values for positioning errors in a machine tool. By treating this as a function approximation problem, and designing and training a network to approximate the function that describes the mechanical errors in the machine, a system was constructed that successfully corrected for the majority of the positioning errors in the machine.

The system designed was not simulated, but was actually implemented on a real machine tool in an industrial environment. Uncompensated errors were measured, the neural network system was designed and integrated into the real-time, deterministic, level of the machine control system. The results were measured using industry-standard test equipment and procedures. The final results achieved were as good as, or slightly better, than the current state-of-the-art in machine tool positioning error compensation capabilities in commercially available control systems.

Three different network architectures were closely studied, and it was demonstrated that any of the three were able to achieve final results sufficient to meet the machine requirements. However, it was shown that with some analysis of the problem prior to network design, an architecture could be created that was well tailored to the application. This architecture was able to meet system requirements with much less training time.

6.2 Limitations

The limitations of this work are in two areas. The first is in the highly manual nature of the work required to implement the system. Data is manually moved from the laser interferometer system to the network design and training system. Results are then manually moved to the real-time calculation programming system and imbedded in the code. The end result is then manually moved to the control system and installed. In working on this project, the ability of each of these systems to take the information and data from other systems was proven, but no real effort was made to truly integrate these systems.

The second limiting area was in designing the neural network to perform the function approximation. As it was implemented, there was no generic or universal solution that could be used to solve a wide range of problems. For example, applying the neural network-based positioning error compensation system to another machine tool would likely require the re-examining of the network architecture and redesigning it to solve the new function approximation problem (a new set of error data).

6.3 Future Work

While this project was successful in demonstrating the application of neural networks to the positioning error problem, there is certainly much more work to be done. Addressing the limitations discussed would be a high priority. This would involve several areas of integration. One would be the integration of the control of the laser

interferometer system and the machine tool motion. This would eliminate the dual set up required, and it would facilitate development of more advanced motion routines. For example, variable increments of motion over the range of motion versus the current requirement for a fixed interval across the entire range of motion can be examined.

Development of a widely applicable network architecture would greatly improve the usability of this technology in the industrial environment. Development of a universal architecture would also facilitate development of a more universal real-time calculation routine that would not have to be independently implemented for each individual machine.

Other areas of future work would focus on expansion of the capabilities of the system. For example, including more variables into the compensation calculation. This would allow the system to account for errors induced by thermal changes or by errors caused by the interaction between machine axes or by fundamental geometric errors in the machine. A common geometric error is an out-of-square condition, where the axes of motion are not exactly perpendicular. In this case, the motion of one axis would induce a positioning error in the other axes.

References

Alguindigue, Israel E., Anna Loskiewicz-Buczak, and Robert E. Uhrig. "Clustering and Classification Techniques for the Analysis of Vibration Signatures." *Applications of Artificial Neural Networks III* SPIE Vol. 1709 (1992).

American Tool, Inc. *Operator's Instruction, Service and Repair Parts Manual for the American Tool 'The Hustler' N/C Turning Center*, ([Cincinnati]: n.p. 1975).

Automated Precision, Inc. *User Manual for the 5/6-D Laser Measuring System V3.0*, ([Gaithersburg]: n.p. n.d.).

Baumann, Bernd, Giorgio Rizzoni, and Gregory Washington, *Intelligent Control of Hybrid Vehicles Using Neural Networks and Fuzzy Logic*. SAE Technical Paper Series 981061 (1998).

Demuth, Howard, and Mark Beale *Neural Network Toolbox User's Guide*. ([Natick]: [The Mathworks, Inc.], 2000).

Giddings & Lewis, Inc. *Orion 2200/2300 Machining Centers Electrical Service Manual With the NumeriPath 8000B Computer Numerical Control*, ([Fond du Lac]: n.p., 1999).

Hagan, Martin T., Howard B. Demuth, and Mark H. Beale, *Neural Network Design* (Boston, MA, PWS Publishing Company, 1996).

Haykin, Simon, *Neural Networks, A Comprehensive Foundation, 2nd ed.* (Upper Saddle River; Prentice-Hall, 1999).

Jenkins, Francis A., and Harvey E. White, *Fundamentals of Optics, 4th ed*. (New York: McGraw-Hill, 1976).

Manufacturing Data Systems, Inc. (MDSI) *OpenCNC Application Programming Interface DataSheet* n.d.. 13 June 2002. <http://www.mdsi2.com/images/DataS.5.0%20API.pdf>.

Manufacturing Data Systems, Inc. (MDSI) *OpenCNC Plus/Pro V6.2 Datasheet* 2002. 13 June. 2002 <http://www.mdsi2.com/images/DataS.6.2%20Plus&Pro.pdf>.

Manufacturing Data Systems, Inc. (MDSI). *OpenCNC 6.0 API Manual*, ([Ann Arbor]: n.p., 2000).

Manufacturing Data Systems, Inc. (MDSI), *OpenCNC 6.0 Variable Dictionary*, ([Ann Arbor]: n.p., 2000).

Microsoft Corporation, *Product Information* 2003. 12 March, 2003 <http://www.microsoft.com/ntworkstation/ProductInformation/default.asp>.

Open Modular Architecture Controls (OMAC) User's Group. *Business Justification of Open Architecture Control White Paper Version 1.0*, 8 Apr. 1999. 13 June 2002 <http://www.arcweb.com/omac/Deliverables/default.htm>.

Russell, Stuart, and Peter Norvig, *Artificial Intelligence, A Modern Approach* (Upper Saddle River; Prentice-Hall, 1995).

United States. Patent and Trademark Office (USPTO). United States Patent 5,523,953. *Method and apparatus for correcting positioning errors on a machine tool*. Araie, Ichiro, and Osamu Akemura (inventors). June 1996. 9 July 2002 <http://patft.uspto.gov/netahtml/srchnum.htm>.

VenturCom, Inc. (VCI) *RTX 5.0 Reference Guide*, ([Cambridge]: n.p. 2000).

Appendix A – Detail Photographs

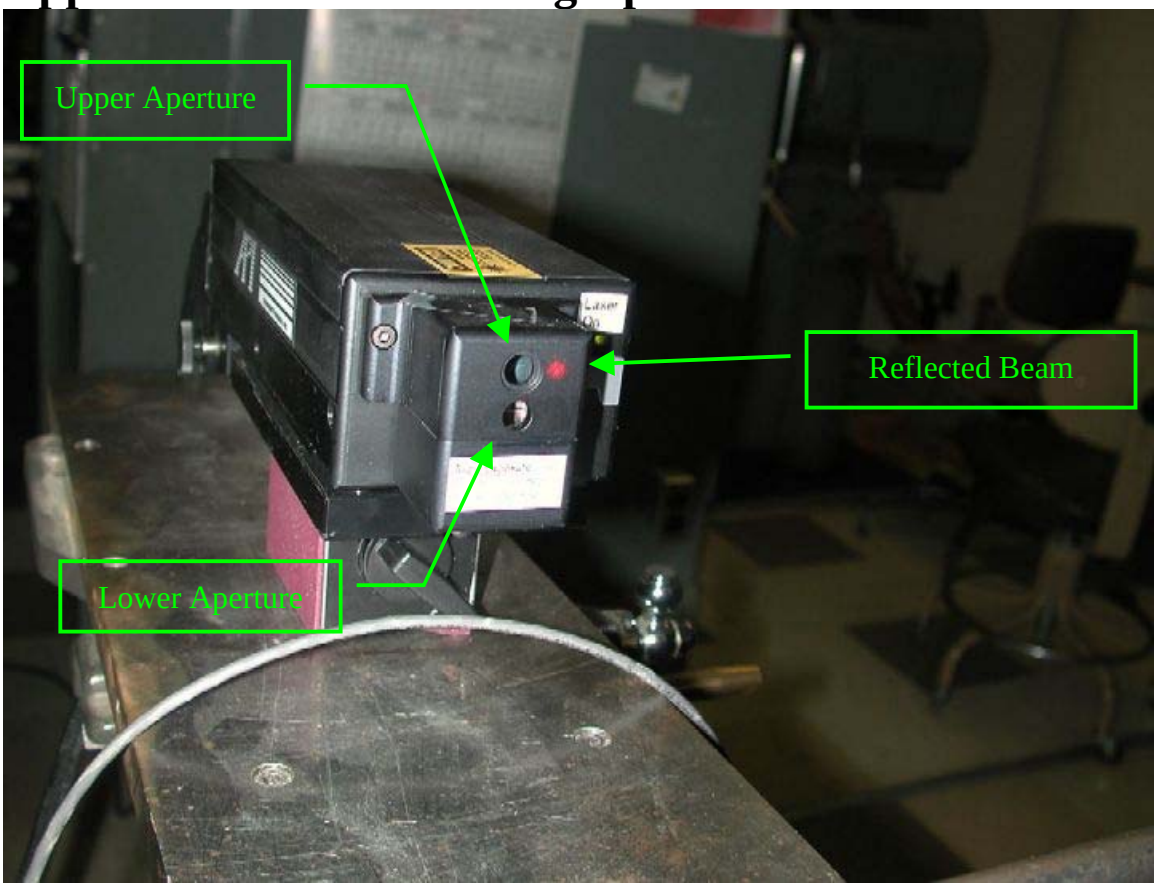


Figure A.1 Laser interferometer upper and lower apertures.

Figure A.1 shows the upper and lower apertures of the interferometer. The laser beam is emitted from the lower aperture and is reflected back to the upper aperture from the mobile reflector. In this photo, the beam is intentionally misaligned to be visible to the side of the upper aperture.

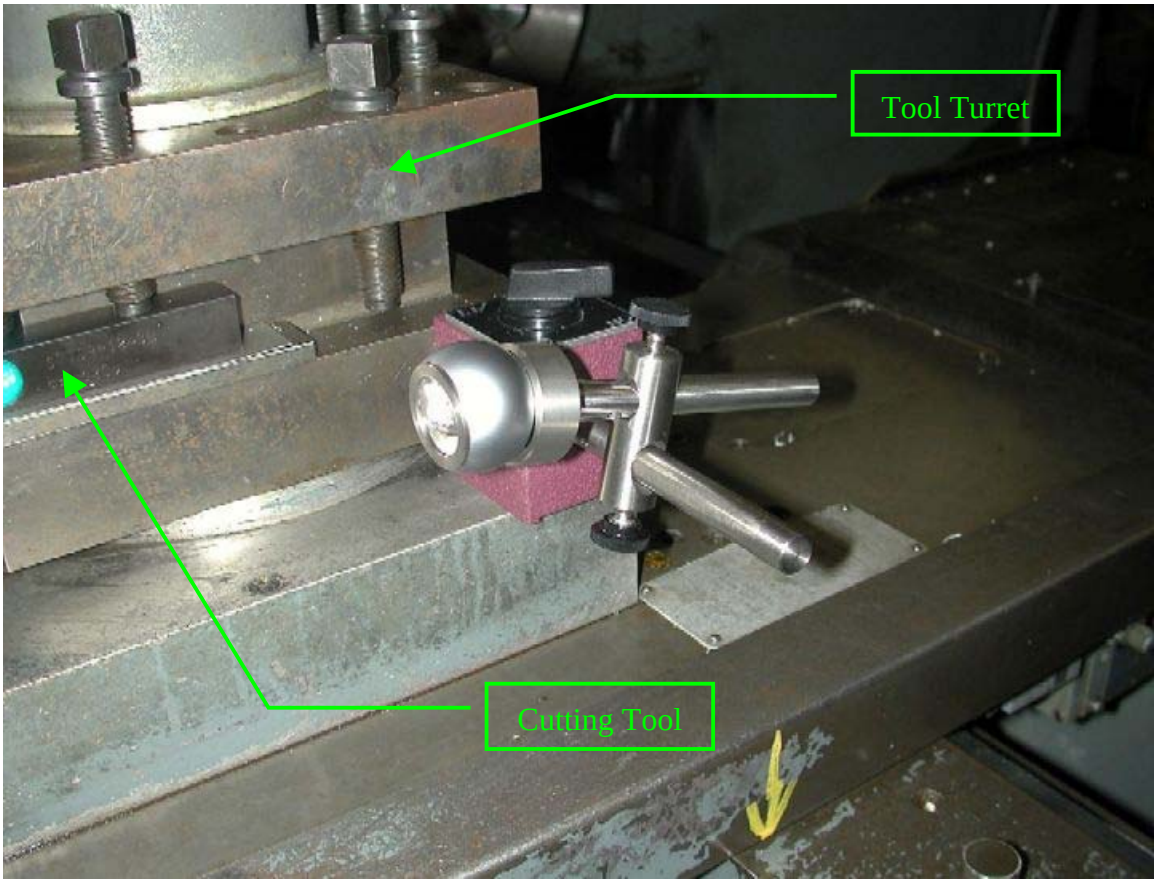


Figure A.2 Close-up of mobile reflector.

Figure A.2 shows a close-up view of the mobile reflector. It is mounted on a magnetic base and its position is adjusted by sliding or rotating the shafts used to mount it. As shown, it is positioned as close as possible to the tool turret and its height matches the height of cutting tools mounted in the turret. In this position, the motion of the mobile reflector closely matches the motion of the cutting tool.

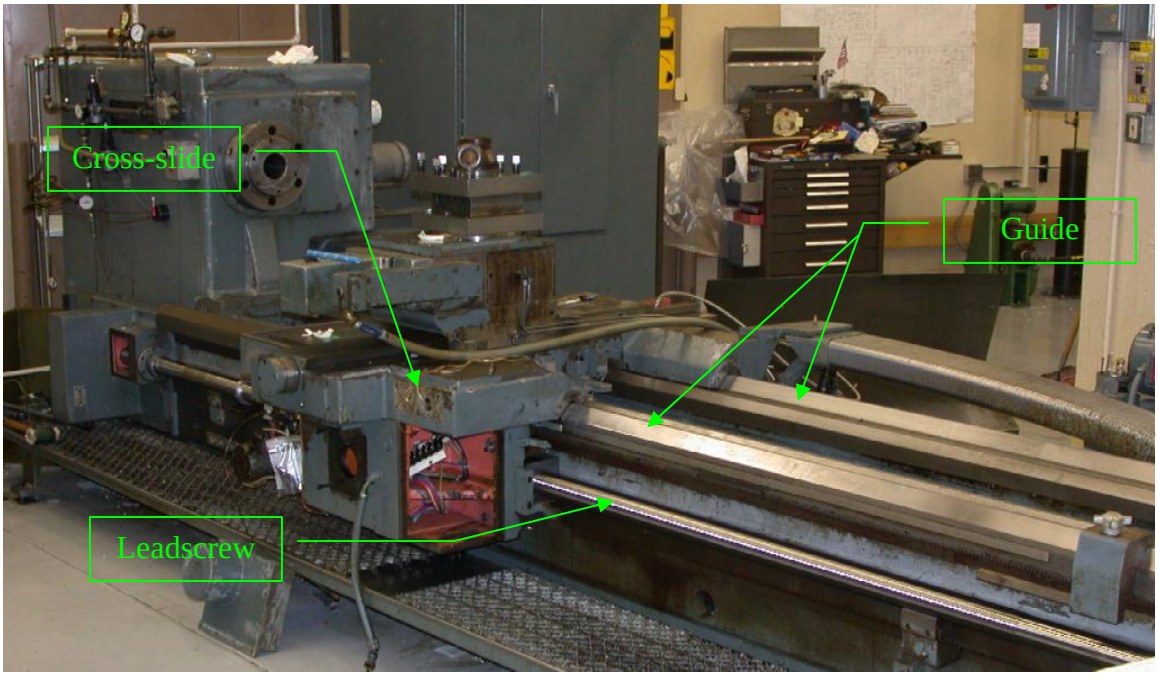


Figure A.3 Long axis leadscrew assembly.

The machine depicted in Figure A.3 similar to the American Hustler lathe used in the experiments. As shown, it has been partially disassembled for major maintenance – leaving the leadscrew exposed and viewable. The long axis leadscrew is visible in the lower portion of the figure, running the entire length of the machine. The leadscrew (and its corresponding ballnut) is used to convert the rotary motion of the servo-motor to the linear motion required by the machine. It is fixed at both ends with bearings, and rotating it causes the cross-slide to move along the guide ways for the long axis. This can be seen in greater detail in Figure A.4.

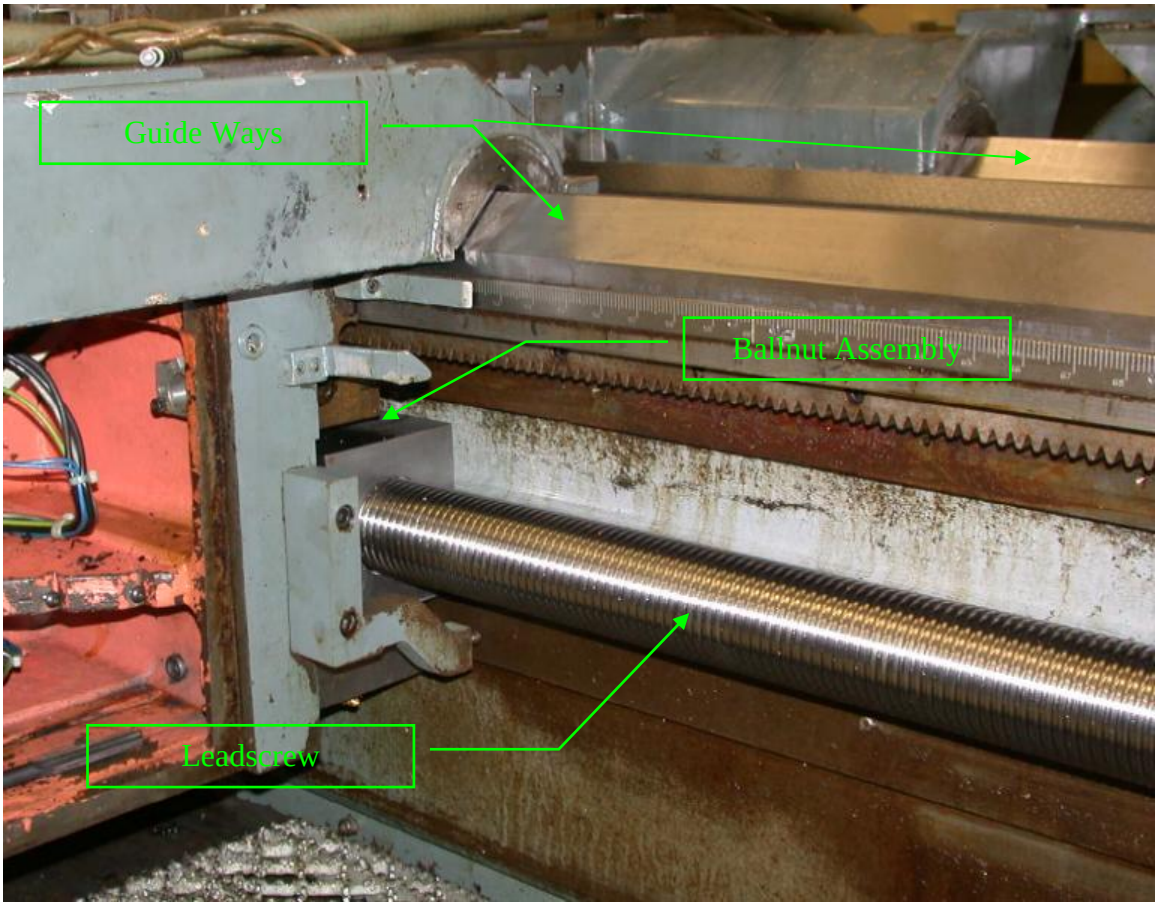


Figure A.4 Close up of ballnut assembly.

Figure A.4 shows the leadscrew-ballnut assembly for the long axis on a lathe. Motion along this axis is generated by a servo-motor rotating the leadscrew. Since the leadscrew is fixed in position by bearings at each end, rotating it causes the ballnut assembly to move linearly along its length. The ballnut is, of course, fixed to the cross-slide which then slides along the long axis of the machine, guided by the guide ways seen at the top of the figure.

Appendix B – Visual Basic Macro Code and Subroutine Result

The following text is the main module from the Visual Basic (VB) macro that error checks the input parameters and generates the G&M code subroutine to run the machine under test.

```
Attribute VB_Name = "Module1"

'**************************************************************************
'*
'* LaserMotion
'* VB Macro to make motion for laser checks
'* By John M. Fines
'* This assumes that we have either a XZ lathe or XYZ mill
'* and only does bidirectional runs
'**************************************************************************

'Ensure that all variables must be declared
Option Explicit

'macro-specific constants
'ms.... => index to argument name
'mv.... => index to argument value
Public Enum macroArgs
    msAxis = 0
    mvAxis        'Axis of motion X=1, Y=2, Z=3
    msStartPos
    mvStartPos    'Starting point of motion
    msEndPos
    mvEndPos      'Ending point of motion
    msIncrement
    mvIncrement   'Motion increment
    msDwell
    mvDwell       'Dwell time at each point
    msNumRuns
    mvNumRuns     'Number of complete runs
    msFeedrate
    mvFeedrate    'Feedrate for moves
    msBacklash
    mvBacklash    'distant for backlash overshoot moves
    msDebugMode
    mvDebugMode       'Debug mode for program 0 = No messages NonZero = messages
    maxArguments
End Enum

'Declare any global constants for use
Const NearZero As Double = 0.0001


'Declare the win32 API function that let's us return an error code
Private Declare Function ExitProcess Lib "kernel32" (ByVal exitCode As Long) As Long

'this needs to be global so we can clean it up on a fatal error.
Dim mdsiMacroObj As IMdsiMacroSupport

'declare argument variables
    Dim Axis As Double
    Dim StartPos As Double
    Dim EndPos As Double
    Dim Increment As Double
    Dim Dwell As Double
    Dim NumRuns As Double
    Dim Feedrate As Double
    Dim Backlash As Double
    Dim DebugMode As Double


Public Sub Main()
    'generic macro variables
    Dim ret As mdsiMacroReturnTypes
    Dim block As String
    Dim nValues As Long

    'macro application-specific variables
    '  for example:
```

```
    'Dim x As Double, y As Double
    'Dim index As Integer
    Dim defAxisLetters As Variant, defGInchMode As Variant, defDecimalShift As Variant
    Dim AxisLetter As String
    Dim AxisNumber As Integer
    Dim AxMin As Double, AxMax As Double
    Dim v As Variant
    Dim CurrentPos As Double
    Dim LaserZero As Boolean
    Dim RunCount As Integer

    'open and clear the subroutine file
    Open "\OpenCNC\Subroutines\laserSub.txt" For Output As #1
    Print #1, "( Error in processing input - see message window )"
    Close #1


    'Initialize and parse the command line arguments
    Initialize

    'first check the axis selection - if bad bail out
    'this allows me to set up initial values based on axis selection
    'it might not be the right axis - but at least it's not total garbage
    If (Axis < 1) Or (Axis > 3) Then
        ret = mdsiMacroObj.mdsiMsgWindowWriteVB("ERROR: Invalid Axis selected must be between 1 and
3")
        fatal ret
    End If

    'Set up initial values
    'go get the axis letters for this machine
    ret = mdsiMacroObj.mdsiVariableReadByNameVB("defAxisLetters", defAxisLetters, nValues)
    If ret <> mdsiMacro_Succeeded Then
        fatal ret  'quit; return the error code to OpenCNC
    End If
    If DebugMode <> 0 Then
        ret = mdsiMacroObj.mdsiMsgWindowWriteVB("DEBUG: Axis letters: " + CStr(defAxisLetters))
    End If
    AxisLetter = Choose(Axis, "X", "Y", "Z")        'select letter based on numeric input from user
    If Axis < 3 Then        'if X or Y, then axis number is 0 or 1 respectively
        AxisNumber = Int(Axis) - 1
    ElseIf ((Axis = 3) And (InStr(CStr(defAxisLetters), "Y") > 0)) Then
        AxisNumber = 2      'if selected Z and Y exists, then axis number is 2
    Else
        AxisNumber = 1      'if selected Z and no Y, then axis number is 1
    End If

    ' get the decimal shift value for location calculations
    ret = mdsiMacroObj.mdsiVariableReadByNameVB("defDecimalShift", defDecimalShift, nValues)
    If ret <> mdsiMacro_Succeeded Then
        fatal ret  'quit; return the error code to OpenCNC
    End If

    'go get the absolute minimum coord for selected axis
    ret = mdsiMacroObj.mdsiVariableReadByNameVB("axLocAbsMin", v, nValues)
    If ret <> mdsiMacro_Succeeded Then
        fatal ret  'quit; return the error code to OpenCNC
    Else
        AxMin = v(AxisNumber) / (2540000 * (10 ^ -(defDecimalShift)))
    End If
    If DebugMode <> 0 Then
        ret = mdsiMacroObj.mdsiMsgWindowWriteVB("DEBUG: Axis Minimum: " + CStr(AxMin))
    End If
    'go get the absolute max coordinate for selected axis
    ret = mdsiMacroObj.mdsiVariableReadByNameVB("axLocAbsMax", v, nValues)
    If ret <> mdsiMacro_Succeeded Then
        fatal ret  'quit; return the error code to OpenCNC
    Else
        AxMax = v(AxisNumber) / (2540000 * (10 ^ -(defDecimalShift)))
    End If
    If DebugMode <> 0 Then
        ret = mdsiMacroObj.mdsiMsgWindowWriteVB("DEBUG: Axis Maximum: " + CStr(AxMax))
    End If

    'go get the inch mode G code for this machine
    ret = mdsiMacroObj.mdsiVariableReadByNameVB("defGInchMode", defGInchMode, nValues)
    If ret <> mdsiMacro_Succeeded Then
        fatal ret  'quit; return the error code to OpenCNC
    End If
    If DebugMode <> 0 Then
        ret = mdsiMacroObj.mdsiMsgWindowWriteVB("DEBUG: Inch Mode G-Code: " + CStr(defGInchMode))
```

```
    End If

    'Error checks - see imbedded messages for individual purpose
    If (InStr(CStr(defAxisLetters), AxisLetter) < 1) Then
        ret = mdsiMacroObj.mdsiMsgWindowWriteVB("ERROR: Invalid Axis specified")
    ElseIf (StartPos >= EndPos) Then
        ret = mdsiMacroObj.mdsiMsgWindowWriteVB("ERROR: Start Position must be less than End
Position")
    ElseIf (StartPos < AxMin) Or (StartPos > AxMax) Then
        ret = mdsiMacroObj.mdsiMsgWindowWriteVB("ERROR: Invalid Start Position must be between " +
CStr(AxMin) + " and End Position")
    ElseIf (EndPos < AxMin) Or (EndPos > AxMax) Then
        ret = mdsiMacroObj.mdsiMsgWindowWriteVB("ERROR: Invalid End Position must be between Start
Position and " + CStr(AxMax))
    ElseIf (Increment < NearZero) Or (Increment > (EndPos - StartPos)) Then
        ret = mdsiMacroObj.mdsiMsgWindowWriteVB("ERROR: Invalid Increment spec'd - too small or too
large")
    ElseIf Not CheckIncrement(EndPos, StartPos, Increment) Then
        ret = mdsiMacroObj.mdsiMsgWindowWriteVB("ERROR: Increment does not divide evenly")
    ElseIf (Dwell < 1) Or (Dwell > 99) Then
        ret = mdsiMacroObj.mdsiMsgWindowWriteVB("ERROR: Invalid Dwell spec'd must be between 1 and
99")
    ElseIf (NumRuns < 1) Or (NumRuns > 99) Then
        ret = mdsiMacroObj.mdsiMsgWindowWriteVB("ERROR: Invalid Number of Runs spec'd must be between
1 and 99")
    ElseIf (Feedrate < 0.1) Or (Feedrate > 50) Then
        ret = mdsiMacroObj.mdsiMsgWindowWriteVB("ERROR: Invalid Feedrate spec'd must be between 0.1
and 50 IPM")
    ElseIf (Backlash <= 0) Or (Backlash > 1) Then
        ret = mdsiMacroObj.mdsiMsgWindowWriteVB("ERROR: Invalid Backlash move spec'd must be between 0
and 1")
    ElseIf ((StartPos - Backlash) < AxMin) Then
        ret = mdsiMacroObj.mdsiMsgWindowWriteVB("ERROR: Not enough travel to make backlash move from
starting position")
    ElseIf ((EndPos + Backlash) > AxMax) Then
        ret = mdsiMacroObj.mdsiMsgWindowWriteVB("ERROR: Not enough travel to make backlash move from
ending position")
    Else

        'made it past all the error checks so run the machine
        LaserZero = False

        'open the subroutine file for writing
        Open "\OpenCNC\Subroutines\laserSub.txt" For Output Shared As #1

        'set machine initial conditions
        'if there is a diameter type axis (i.e. a lathe) do the G07
        ret = mdsiMacroObj.mdsiVariableReadByNameVB("axDiameterType", v, nValues)
        If ret <> mdsiMacro_Succeeded Then
            fatal ret  'quit; return the error code to OpenCNC
        End If
        If v(AxisNumber) <> 0 Then
            block = "G07"
            Print #1, block
        End If

        'set to inch mode
        block = "G" & CStr(defGInchMode)
        Print #1, block

        'Feedrate moves, Inch mode, work coord offsets off, absolute mode, IPM
        block = "G01G59G90G94  (Setting Machine Modals)"
        Print #1, block

        'move to starting position
        'I'm using G53 to make sure offsets are not a factor - necessary???
        block = "G53" & AxisLetter & Format(StartPos, "##0.0###") & "F" & Format(Feedrate, "##.#")
        Print #1, block

        'start to loop through motion
        For RunCount = 1 To NumRuns
            'make the backlash move
            block = "G53" & AxisLetter & Format(StartPos - Backlash, "##0.0###") & "F" &
Format(Feedrate, "##.#")
            Print #1, block

            block = "G04F" & Format(Dwell, "##")
            Print #1, block
            'return to start position
            block = "G53" & AxisLetter & Format(StartPos, "##0.0###") & "F" & Format(Feedrate, "##.#")
            Print #1, block
```

```
            If LaserZero Then
                block = "G04F" & Format(Dwell, "##")
                Print #1, block
            Else
                block = "M00  (Program Stop for Zero Laser)"
                Print #1, block
                LaserZero = True
            End If
            'clear the comment line
            block = "(Measurement run: " & CStr(RunCount) & ")"
            Print #1, block

            'loop for motion in positive direction
            CurrentPos = StartPos + Increment
            Do
                block = "G53" & AxisLetter & Format(CurrentPos, "##0.0###")
                Print #1, block
                block = "G04F" & Format(Dwell, "##")
                Print #1, block
                CurrentPos = CurrentPos + Increment
            Loop Until (CurrentPos > (EndPos + NearZero))
            'make the end backlash move
            block = "G53" & AxisLetter & Format(EndPos + Backlash, "##0.0###")
            Print #1, block
            block = "G04F" & Format(Dwell, "##")
            Print #1, block

            'loop for return pass
            CurrentPos = EndPos
            Do
                block = "G53" & AxisLetter & Format(CurrentPos, "##0.0###")
                Print #1, block
                block = "G04F" & Format(Dwell, "##")
                Print #1, block
                CurrentPos = CurrentPos - Increment
            Loop Until (CurrentPos < (StartPos - NearZero))
            block = "M01 (End of Run: " & CStr(RunCount) & ")"
            Print #1, block
        Next
'        ret = mdsiMacroObj.mdsiMsgWindowWriteVB("End of requested runs - exiting macro")
    End If

    Set mdsiMacroObj = Nothing
    Close #1

End Sub   'main()

'This function checks for (EndPos - StartPos) / Increment evenly (within value of NearZero)
'By looping through - there are numerical problems with int()
Public Function CheckIncrement(E As Double, S As Double, I As Double) As Boolean
    'local variables
    Dim x As Double

    CheckIncrement = True       'default to success
    x = S

    Do
        x = x + I
    Loop Until (Abs(E - x) < NearZero) Or (x > E)
    If (x > E) And (Abs(E - x) > NearZero) Then
        CheckIncrement = False
    Else
        x = E
        Do
            x = x - I
        Loop Until (Abs(S - x) < NearZero) Or (x < S)
        If (x < S) And (Abs(S - x) > NearZero) Then
            CheckIncrement = False
        End If
    End If

End Function 'CheckIncrement()
'This subroutine does the initialization and command line parsing.
'
Public Function Initialize()
    'generic macro variables
    Dim arguments(maxArguments + 10) As Variant
    Dim retArgs As Variant
    Dim ret As mdsiMacroReturnTypes
    Dim argCount As mdsiMacroReturnTypes
    Dim errorPosition As Long
```

```
    'MDSI initialization
    '  this section of code must not be deleted
    Set mdsiMacroObj = CreateObject("mdsiMacroSupport.clsMdsiMacroSupport")
    ret = mdsiMacroObj.Initialize(Command())
    If ret <> mdsiMacro_Succeeded Then
        fatal ret  'quit; return the error code to OpenCNC
    End If
    'setup argument names
    arguments(msAxis) = "A"
    arguments(msStartPos) = "S"
    arguments(msEndPos) = "E"
    arguments(msIncrement) = "I"
    arguments(msDwell) = "T"
    arguments(msNumRuns) = "N"
    arguments(msFeedrate) = "F"
    arguments(msBacklash) = "B"
    arguments(msDebugMode) = "D"

    'setup default values

    'parse the incoming command-line arguments
    argCount = mdsiMacroObj.mdsiParseMacroArgsVB(arguments, retArgs, errorPosition)
    If argCount < mdsiMacro_Succeeded Then
        fatal argCount  'quit; return the parse error code to OpenCNC
    End If
    'assign argument variables
    Axis = retArgs(mvAxis)
    StartPos = retArgs(mvStartPos)
    EndPos = retArgs(mvEndPos)
    Increment = retArgs(mvIncrement)
    Dwell = retArgs(mvDwell)
    NumRuns = retArgs(mvNumRuns)
    Feedrate = retArgs(mvFeedrate)
    Backlash = retArgs(mvBacklash)
    DebugMode = retArgs(mvDebugMode)

End Function

' Call fatal when you want to quit with an error code.
'
Public Sub fatal(exitCode As Long)
  Dim o As Object

  'cleanup all objects we know about
  mdsiMacroObj.unInitialize
  Set mdsiMacroObj = Nothing
  For Each o In Forms
    Unload o
  Next o

  'close all open files
  Reset

  ExitProcess exitCode
End Sub


' This is a wrapper function for mdsiBlockWrite.
' It checks the return code and calls 'fatal' on an error other
' than 'mdsiMacro_InJogMode'.
' Note:  A caveat about using this function:  mdsiBlockWriteVB returns
'        the error code 'mdsiMacro_InJogMode' if OpenCNC is in Jog mode.
'        This is not considered a fatal error here.  The property
'        'runMode' may be checked to determine if OpenCNC is in Jog mode.
'
Public Sub SendBlock(block As String)
  Dim ret As mdsiMacroReturnTypes

  'send the block to the parser
  ret = mdsiMacroObj.mdsiBlockWriteVB(block)
  If (ret <> mdsiMacro_Succeeded) Then      'And (ret <> mdsiMacro_InJogMode) Then
    'failed to send block; do error processing here.
    fatal ret  'quit; return the error code to OpenCNC
  Else
    ret = mdsiMacroObj.mdsiMsgWindowWriteVB("Sendblock OK:" & block)
  End If
End Sub
```

The following text is the resultant subroutine code after running the part program listed in Figure 4.3.

```
G70
G01G59G90G94  (Setting Machine Modals)
G53Z1.0F50.
G53Z0.9F50.
G04F2
G53Z1.0F50.
M00  (Program Stop for Zero Laser)
(Measurement run: 1)
G53Z2.0
G04F2
G53Z3.0
G04F2
G53Z4.0
G04F2
G53Z5.0
G04F2
G53Z6.0
G04F2
G53Z7.0
G04F2
G53Z7.1
G04F2
G53Z7.0
G04F2
G53Z6.0
G04F2
G53Z5.0
G04F2
G53Z4.0
G04F2
G53Z3.0
G04F2
G53Z2.0
G04F2
G53Z1.0
G04F2
M01 (End of Run: 1)
G53Z0.9F50.
G04F2
G53Z1.0F50.
G04F2
(Measurement run: 2)
G53Z2.0
G04F2
G53Z3.0
```

```
G04F2
G53Z4.0
G04F2
G53Z5.0
G04F2
G53Z6.0
G04F2
G53Z7.0
G04F2
G53Z7.1
G04F2
G53Z7.0
G04F2
G53Z6.0
G04F2
G53Z5.0
G04F2
G53Z4.0
G04F2
G53Z3.0
G04F2
G53Z2.0
G04F2
G53Z1.0
G04F2
M01 (End of Run: 2)
G53Z0.9F50.
G04F2
G53Z1.0F50.
G04F2
(Measurement run: 3)
G53Z2.0
G04F2
G53Z3.0
G04F2
G53Z4.0
G04F2
G53Z5.0
G04F2
G53Z6.0
G04F2
G53Z7.0
G04F2
G53Z7.1
G04F2
G53Z7.0
G04F2
```

```
G53Z6.0
G04F2
G53Z5.0
G04F2
G53Z4.0
G04F2
G53Z3.0
G04F2
G53Z2.0
G04F2
G53Z1.0
G04F2
M01 (End of Run: 3)
```

Appendix C - M-File for Neural Network Generation

The following Matlab m-file script is the basic tool for generating a new neural network, training it with test data, and viewing the results of the new network.

```
% preprocess.m
% basic processing of the data to prepare for nn tasks

% clear the workspace and load the data file
clear;
load z;

% plot the runs just cause I like to look at pictures
plot (smallzv9_2(1:3122,1),smallzv9_2(1:3122,3),'-r');
hold on
plot (smallzv9_2(3123:6244,1),smallzv9_2(3123:6244,3),'-
g');
plot (smallzv9_2(6245:9366,1),smallzv9_2(6245:9366,3),'-
b');
plot (smallzv9_2(9367:end,1),smallzv9_2(9367:end,3),'-c');
title('Four Runs As Recorded By Laser');

%plot the averages for the same reason as above
figure
plot (smallzv9_2ave(:,1),smallzv9_2ave(:,2),'-b');
title('Averages');

% vector indicating direction of motion - 0 => positive, 1
=> negative
dir = [zeros(1561,1); ones(1561,1)];

% generate the network
net = newff([0 1;1 40],[60
1],{'logsig','purelin'},'trainlm','learngdm','sse');

% set constants
% for the network construction
net.trainParam.show = 5;
net.trainParam.epochs = 1000;
net.trainParam.goal = 4e-8;

% for simplicity, collect the inputs (p) and targets (t)
p = [dir smallzv9_2ave(:,1)]';
t = smallzv9_2ave(:,2)';

% train the network
net = train(net,p,t);
```

```
% test the network on the first run
Xm = [zeros(1,1561); smallzv9_2(1:1561,1)'];
am = sim(net,Xm);

Ym = [ones(1,1561); smallzv9_2(1562:3122,1)'];
bm = sim(net,Ym);

% plot the result
figure(1)
hold on
plot(smallzv9_2ave(1:1561,1)', am, '-b');
plot(smallzv9_2ave(1562:3122,1), bm, '-b');

% save the resultant network for later use
save net30 net
```

Appendix D – M-File for Extraction of Network Values

The following Matlab m-file script is a compete listing of the script used to extract and save the key parameters. These values are written out to a text file in a form intended for ease of use as constant definitions in a C program.

```
% netvalues.m
% gets the network values and outputs them in as useful
% a format as I can determine

clear
load netmean;
load netdet3;

% for each network, I need:
% inputWeights(:,2) - multiplied by position input
% inputWeights(:,1) + bias1 - equivalent to dir input = 1 +
bias: added % to pos input * IW(:,2)
% bias1 - used with dir input = 0: either this or above is
added to pos % input * IW(:,2)
% layerWeights - multiplied by result of layer 1
% bias2 - added to result of above
fid = fopen('netvalues.txt','w');

% begin processing for netmean
inputWeights = netmean.IW{1};
bias1 = netmean.b{1};
bias2 = netmean.b{2};
layerWeights = netmean.LW{2};

% output input weights column 2
fprintf(fid,'const double FullIW2[%d] =
{\r\n',length(inputWeights));
for i = 1:(length(inputWeights)-1)
     fprintf(fid,'  %+2.16e,\r\n',inputWeights(i,2));
end
fprintf(fid,'  %+2.16e\t};\r\n\r\n',inputWeights(i+1,2));

% output input weights column 1 + bias 1
fprintf(fid,'const double FullIW_B1[%d] =
{\r\n',length(inputWeights));
for i = 1:(length(inputWeights)-1)
     fprintf(fid,'
%+2.16e,\r\n',inputWeights(i,1)+bias1(i));
end
fprintf(fid,'
%+2.16e\t};\r\n\r\n',inputWeights(i+1,1)+bias1(i+1));

% output bias values for layer 1
```

```
fprintf(fid,'const double FullBias1[%d] =
{\r\n',length(bias1));
for i = 1:(length(bias1)-1)
     fprintf(fid,'  %+2.16e,\r\n',bias1(i));
end
fprintf(fid,'  %+2.16e\t};\r\n\r\n',bias1(i+1));

% output layer weights
fprintf(fid,'const double FullLW[%d] =
{\r\n',length(layerWeights));
for i = 1:(length(layerWeights)-1)
     fprintf(fid,'  %+2.16e,\r\n',layerWeights(i));
end
fprintf(fid,'  %+2.16e\t};\r\n\r\n',layerWeights(i+1));

% output the bias value for layer 2
fprintf(fid,'const double FullBias2 = ');
fprintf(fid,'%+2.16e;\r\n\r\n',bias2);

% begin processing for netdet
inputWeights = netdet.IW{1};
bias1 = netdet.b{1};
bias2 = netdet.b{2};
layerWeights = netdet.LW{2};

% output input weights column 2
fprintf(fid,'const double cyclicIW2[%d] =
{\r\n',length(inputWeights));
for i = 1:(length(inputWeights)-1)
     fprintf(fid,'  %+2.16e,\r\n',inputWeights(i,2));
end
fprintf(fid,'  %+2.16e\t};\r\n\r\n',inputWeights(i+1,2));

% output input weights column 1 + bias 1
fprintf(fid,'const double cyclicIW_B1[%d] =
{\r\n',length(inputWeights));
for i = 1:(length(inputWeights)-1)
     fprintf(fid,'
%+2.16e,\r\n',inputWeights(i,1)+bias1(i));
end
fprintf(fid,'
%+2.16e\t};\r\n\r\n',inputWeights(i+1,1)+bias1(i+1));

% output bias values for layer 1
fprintf(fid,'const double cyclicBias1[%d] =
{\r\n',length(bias1));
for i = 1:(length(bias1)-1)
```

```
     fprintf(fid,'  %+2.16e,\r\n',bias1(i));
end
fprintf(fid,'  %+2.16e\t};\r\n\r\n',bias1(i+1));

% output layer weights
fprintf(fid,'const double cyclicLW[%d] =
{\r\n',length(layerWeights));
for i = 1:(length(layerWeights)-1)
     fprintf(fid,'  %+2.16e,\r\n',layerWeights(i));
end
fprintf(fid,'  %+2.16e\t};\r\n\r\n',layerWeights(i+1));

% output the bias value for layer 2
fprintf(fid,'const double cyclicBias2 = ');
fprintf(fid,'%+2.16e;\r\n\r\n',bias2);
fclose(fid);
```

The following text file is the result of running the previous m-file on one of the neural networks developed. It is the output of netvalues.m.

```
const double FullIW2[30] = {
  -3.5299591828609689e-001,
  +2.9772889998702345e-001,
  -6.4793260461973534e-001,
  +5.4195020390193360e-001,
  +9.7106115496384826e-001,
  +6.1905741094569089e-001,
  +7.7323968706067736e-001,
  -1.9677546910613436e-001,
  -7.5903727601389914e-003,
  -1.0715500354505123e+000,
  +7.4182768355941786e-001,
  -8.3897565665597562e-001,
  +6.2476996136658969e-001,
  +6.2602614925206490e-001,
  -2.4024212111154136e-001,
  +8.1848186292315062e-001,
  -4.0822195177470089e-001,
  -7.1626070157100863e-001,
  +8.9921491040980839e-001,
  +1.2928512211754312e-001,
  -8.2383751470809030e-001,
  +8.3874484365466773e-001,
  +3.7197208002594834e-001,
  -7.0063070389361926e-001,
  -7.4830770954720938e-001,
  +2.5576441880935413e-001,
  +5.7884559271706870e-001,
  -5.1772547956387849e-001,
  +2.8298441045876910e-001,
  -3.9568791198859837e-001    };

const double FullIW_B1[30] = {
  +7.0470049377939930e+000,
  -1.4097100862194786e+001,
  +1.8789253140934623e+001,
  -9.7455579099105591e+000,
  -2.1396950984946812e+001,
  -1.0504336719678323e+001,
  -8.4603634074333058e+000,
  -4.9485358741831220e+000,
  +8.5169577723346919e+000,
  +1.3935073610259401e+001,
  -1.6541715543130792e+001,
  +1.3470121353454136e+001,
```

```
  -1.7694116756426421e+001,
  -1.8332232907721902e+001,
  -9.1405546309460455e+000,
  -1.0092226745629825e+001,
  +2.3863482551199322e+001,
  +2.3776316084580969e+001,
  -6.8667949688335881e+000,
  -2.6486017651552007e+001,
  +3.1025011814584467e+000,
  +4.3567770884020387e+000,
  +1.1123058872867919e+001,
  +5.5031336101198693e+000,
  +2.9086499981718013e+001,
  +1.9976889065608717e+001,
  -3.1096013722954432e+001,
  +3.5389976478144561e+001,
  -3.1986270148644632e+001,
  -2.0684497872413800e+001    };

const double FullBias1[30] = {
  +3.5545079968154958e+001,
  -3.6416031669458668e+001,
  +3.4983425512452655e+001,
  +8.5797397367591479e+000,
  -3.1331556021774304e+001,
  +1.6358650232444849e+000,
  -5.4085567369344272e+000,
  +2.5396281073556299e+001,
  -2.2155285650187714e+001,
  +5.4701322032968340e+000,
  -2.1295955495436054e+000,
  +1.1081061271037377e+001,
  -7.2995753942845081e+000,
  -9.5543556064397652e+000,
  +2.0061702533473493e+001,
  -1.9567626679904070e+001,
  +9.7023646305029010e-001,
  +1.0185475520917636e+001,
  -1.6336552516527902e+001,
  -5.3423886361010664e-001,
  +1.4957998772309816e+001,
  -1.5311024980801433e+001,
  -1.3861709831682187e+001,
  +8.6954486655705221e+000,
  +2.2303031682519762e+001,
  -8.8421753766822473e+000,
  -1.7125562961409566e+000,
```

```
  +1.6313926034840971e+001,
  -1.7332421739865340e+000,
  +5.9170950459425411e+000    };

const double FullLW[30] = {
  -1.9722090856230893e-002,
  -3.3381213233866988e-002,
  -3.5392149785856523e-002,
  -2.2485102542564316e-002,
  +2.1203505113612767e-003,
  +1.6435067023459145e-002,
  -2.8571235938803881e-003,
  -1.7985204049942757e-001,
  -2.8522718298553684e-001,
  +1.6739422819069079e-003,
  -6.6813157353764335e-003,
  +5.1686224863966524e-003,
  -1.2998730486961044e-002,
  -2.5279135841123679e-002,
  +5.5089960293835630e-002,
  +2.0197341240874665e-003,
  -3.0999957862638091e-002,
  -1.5265730944106306e-003,
  -5.8489218591469359e-003,
  -1.9922873247891157e-001,
  +1.0679026108750460e-003,
  +4.2333059590505904e-003,
  -2.1126715470884939e-002,
  -7.3627218857791965e-003,
  -3.7719891563069265e-003,
  +4.6331692055082856e-002,
  +6.3970870956342772e-005,
  +1.6432927867145796e-002,
  +7.2777120317224900e-002,
  -7.2225148283068585e-002    };

const double FullBias2 = +3.3526357495979814e-001;

const double cyclicIW2[8] = {
  +4.0199735339014765e+001,
  +1.1412266461437305e+001,
  -2.6173165967393352e+001,
  -3.5485150577054590e+001,
  +4.7523812303695863e+001,
  -2.6388036946666858e+001,
  -6.4902852076295659e+001,
  -3.7128378964014928e+001    };
```

```
const double cyclicIW_B1[8] = {
  -7.3902984827427503e+000,
  -4.8286945653885969e+000,
  +7.7240626438899760e+000,
  +9.6338193323527328e+000,
  +3.7826837129765529e+000,
  -1.1199979350411143e+001,
  +3.8645326454024267e+000,
  -8.5064102092927918e+000    };

const double cyclicBias1[8] = {
  -1.9372202168215299e+001,
  +1.1473068913108277e+001,
  -7.1779003755671003e+000,
  -5.9534315151373125e+000,
  -1.0972641799667036e+001,
  +3.2932649248391974e+000,
  -1.5021445926326495e+000,
  +4.4508828942124801e+000    };

const double cyclicLW[8] = {
  -1.2845598766206338e-002,
  -3.7583290668195674e-001,
  -7.9272704047420051e-001,
  +4.2340609441792598e-001,
  -3.3566978204302399e-003,
  -1.9669801168724809e-002,
  -3.5317654044065912e-003,
  +1.6403797843778867e-002    };

const double cyclicBias2 = +3.7874518228170045e-001;
```

Appendix E – C Program for Network Calculation

The following C program is the code for the calculation of the feed-forward path of the neural network used in determining error-compensation values. This is the central part of the program that runs in the hard real-time environment to interact with the lathe control system.

```
/* calcZ.c */

#include <math.h>

/* input weights column 2 - for mulitply by position input
*/
const double FullIW2[30] = {
  -3.5299591828609689e-001,
  +2.9772889998702345e-001,
  -6.4793260461973534e-001,
  +5.4195020390193360e-001,
  +9.7106115496384826e-001,
  +6.1905741094569089e-001,
  +7.7323968706067736e-001,
  -1.9677546910613436e-001,
  -7.5903727601389914e-003,
  -1.0715500354505123e+000,
  +7.4182768355941786e-001,
  -8.3897565665597562e-001,
  +6.2476996136658969e-001,
  +6.2602614925206490e-001,
  -2.4024212111154136e-001,
  +8.1848186292315062e-001,
  -4.0822195177470089e-001,
  -7.1626070157100863e-001,
  +8.9921491040980839e-001,
  +1.2928512211754312e-001,
  -8.2383751470809030e-001,
  +8.3874484365466773e-001,
  +3.7197208002594834e-001,
  -7.0063070389361926e-001,
  -7.4830770954720938e-001,
  +2.5576441880935413e-001,
  +5.7884559271706870e-001,
  -5.1772547956387849e-001,
  +2.8298441045876910e-001,
  -3.9568791198859837e-001    };

/* input weights column 1 added to input bias - used for
addtion when direction input = 1 */
const double FullIW_B1[30] = {
  +7.0470049377939930e+000,
```

```
  -1.4097100862194786e+001,
  +1.8789253140934623e+001,
  -9.7455579099105591e+000,
  -2.1396950984946812e+001,
  -1.0504336719678323e+001,
  -8.4603634074333058e+000,
  -4.9485358741831220e+000,
  +8.5169577723346919e+000,
  +1.3935073610259401e+001,
  -1.6541715543130792e+001,
  +1.3470121353454136e+001,
  -1.7694116756426421e+001,
  -1.8332232907721902e+001,
  -9.1405546309460455e+000,
  -1.0092226745629825e+001,
  +2.3863482551199322e+001,
  +2.3776316084580969e+001,
  -6.8667949688335881e+000,
  -2.6486017651552007e+001,
  +3.1025011814584467e+000,
  +4.3567770884020387e+000,
  +1.1123058872867919e+001,
  +5.5031336101198693e+000,
  +2.9086499981718013e+001,
  +1.9976889065608717e+001,
  -3.1096013722954432e+001,
  +3.5389976478144561e+001,
  -3.1986270148644632e+001,
  -2.0684497872413800e+001    };

/* input bias - used for addition when direction input = 0
*/
const double FullBias1[30] = {
  +3.5545079968154958e+001,
  -3.6416031669458668e+001,
  +3.4983425512452655e+001,
  +8.5797397367591479e+000,
  -3.1331556021774304e+001,
  +1.6358650232444849e+000,
  -5.4085567369344272e+000,
  +2.5396281073556299e+001,
  -2.2155285650187714e+001,
  +5.4701322032968340e+000,
  -2.1295955495436054e+000,
  +1.1081061271037377e+001,
  -7.2995753942845081e+000,
  -9.5543556064397652e+000,
```

```
  +2.0061702533473493e+001,
  -1.9567626679904070e+001,
  +9.7023646305029010e-001,
  +1.0185475520917636e+001,
  -1.6336552516527902e+001,
  -5.3423886361010664e-001,
  +1.4957998772309816e+001,
  -1.5311024980801433e+001,
  -1.3861709831682187e+001,
  +8.6954486655705221e+000,
  +2.2303031682519762e+001,
  -8.8421753766822473e+000,
  -1.7125562961409566e+000,
  +1.6313926034840971e+001,
  -1.7332421739865340e+000,
  +5.9170950459425411e+000    };

/* layer weights - multiplied by result of layer 1 */
const double FullLW[30] = {
  -1.9722090856230893e-002,
  -3.3381213233866988e-002,
  -3.5392149785856523e-002,
  -2.2485102542564316e-002,
  +2.1203505113612767e-003,
  +1.6435067023459145e-002,
  -2.8571235938803881e-003,
  -1.7985204049942757e-001,
  -2.8522718298553684e-001,
  +1.6739422819069079e-003,
  -6.6813157353764335e-003,
  +5.1686224863966524e-003,
  -1.2998730486961044e-002,
  -2.5279135841123679e-002,
  +5.5089960293835630e-002,
  +2.0197341240874665e-003,
  -3.0999957862638091e-002,
  -1.5265730944106306e-003,
  -5.8489218591469359e-003,
  -1.9922873247891157e-001,
  +1.0679026108750460e-003,
  +4.2333059590505904e-003,
  -2.1126715470884939e-002,
  -7.3627218857791965e-003,
  -3.7719891563069265e-003,
  +4.6331692055082856e-002,
  +6.3970870956342772e-005,
  +1.6432927867145796e-002,
```

```
  +7.2777120317224900e-002,
  -7.2225148283068585e-002    };

/* bias value layer 2 */
const double FullBias2 = +3.3526357495979814e-001;

/* input weights column 2 for cyclic error network */
const double cyclicIW2[8] = {
  +4.0199735339014765e+001,
  +1.1412266461437305e+001,
  -2.6173165967393352e+001,
  -3.5485150577054590e+001,
  +4.7523812303695863e+001,
  -2.6388036946666858e+001,
  -6.4902852076295659e+001,
  -3.7128378964014928e+001    };

/* input weights column 1 added to bias layer 1 for cyclic
error net */
const double cyclicIW_B1[8] = {
  -7.3902984827427503e+000,
  -4.8286945653885969e+000,
  +7.7240626438899760e+000,
  +9.6338193323527328e+000,
  +3.7826837129765529e+000,
  -1.1199979350411143e+001,
  +3.8645326454024267e+000,
  -8.5064102092927918e+000    };

/* bias layer 1 for cyclic error net */
const double cyclicBias1[8] = {
  -1.9372202168215299e+001,
  +1.1473068913108277e+001,
  -7.1779003755671003e+000,
  -5.9534315151373125e+000,
  -1.0972641799667036e+001,
  +3.2932649248391974e+000,
  -1.5021445926326495e+000,
  +4.4508828942124801e+000    };

/* layer weights for cyclic error net */
const double cyclicLW[8] = {
  -1.2845598766206338e-002,
  -3.7583290668195674e-001,
  -7.9272704047420051e-001,
  +4.2340609441792598e-001,
  -3.5669678204302399e-003,
```

```
  -1.9669801168724809e-002,
  -3.5317654044065912e-003,
  +1.6403797843778867e-002    };

/* bias layer 2 for cyclic error net */
const double cyclicBias2 = +3.7874518228170045e-001;


/*************************************************************
***************************
 * function calcZ
 * inputs:
 *        dir: direction of motion 0 => forward, 1 =>
reverse
 *        pos: current position in inches
 * ouputs:
 *        returns comp value for position and direction in
Z axis
 * processing:
 *        This does the feed forward portion of the neural
networks.It calculates the
 *        transfer function of logsig for the hidden layer
and a pure linear for the output
 *        neuron.  There are two networks, one for a full
length error function and one
 *        that handles the details of the cyclic error in
the screw.

*************************************************************
***************************/
double calcZ(int dir, double pos) {

     register int i;
     double temp[30];
     double cyclicPos;
     double fullResult = 0;
     double cyclicResult = 0;

/* init the temp array dependent on direction of travel */
     if (dir == 0)  /* direction input = 0 => zeros out the
input weights */
     {
          for (i = 0; i < 30; i++)
               temp[i] = FullBias1[i];
     }
     else           /* direction input = 1 => includes dir
input weights */
```

```
     {
          for (i = 0; i < 30; i++)
               temp[i] = FullIW_B1[i];
     }

/* calculate position input*input weights and add to temp
*/
     for (i = 0; i < 30; i++)
          temp[i]  += (pos * FullIW2[i]);

/* calculate the logsig function for each element */
     for (i = 0; i < 30; i++)
          temp[i] = 1 / (1 + exp(-(temp[i])));

/* calculate the vector multiplication of layerweights *
result of hidden layer */
     for (i = 0; i < 30; i++)      /* multiplication */
          temp[i] *= FullLW[i];
     for (i = 0; i < 30; i++)      /* summing results */
          fullResult += temp[i];

/* add the bias value for output neuron */
     fullResult += FullBias2;

/* now calculate the value for the cyclic error
compensation */
     cyclicPos = fmod(pos, 0.25);

/* init the temp array dependent on direction of travel */
     if (dir == 0)  /* direction input = 0 => zeros out the
input weights */
     {
          for (i = 0; i < 8; i++)
               temp[i] = cyclicBias1[i];
     }
     else           /* direction input = 1 => includes dir
input weights */
     {
          for (i = 0; i < 8; i++)
               temp[i] = cyclicIW_B1[i];
     }

/* calculate position input*input weights and add to temp
*/
     for (i = 0; i < 8; i++)
          temp[i]  += (cyclicPos * cyclicIW2[i]);
```

```
/* calculate the logsig function for each element */
     for (i = 0; i < 8; i++)
          temp[i] = 1 / (1 + exp(-(temp[i])));

/* calculate the vector multiplication of layerweights *
result of hidden layer */
     for (i = 0; i < 8; i++)       /* multiplication */
          temp[i] *= cyclicLW[i];
     for (i = 0; i < 8; i++)       /* summing results */
          cyclicResult += temp[i];

/* add the bias value for output neuron */
     cyclicResult += cyclicBias2;

     return (- (fullResult + cyclicResult));
}    /* end of calcZ() */
```