Abstract

The PTA Controller interface was given to the students by Dr. Yusef Altintas and Andrew Woronko as a Mech 457 design project. Due to certain insufficiencies of the original interface, several improvements must be made to render the precision turning actuator more user-friendly.

The PTA Controller consists of two parts: a dialog-based graphic user interface and a controller program (modified from the original interface, FCExec). The GUI sends any user input to the controller program through an external data file, which avoids the need for recompilation, an inherent program of the original interface. Several other features such as plotting and default value save/load functions are also incorporated in the Controller.

A number of tests were performed for the PTA. These included open loop and close loop analyses, cutting tests, and frequency response tests. The three former tests were done with the new interface, all of which yielded satisfactory results and showed improvements over the original. The latter test was conducted to find the relationship between the actuator power, frequency, and capacitance using an oscilloscope and an amplifier.

Table of Contents

List of Figures

List of Tables

1.0 Introduction

The precision turning actuator is a device designed to replace a conventional grinding machine in a manufacturing process that requires higher precision and efficiency. It has recently been realized that the current user interface for the device is not sufficient for potential end users due to some of its inherent problems. A new interface should be created to improve upon it.

The objective of this project is to write a user-friendly graphic user interface (GUI) for the precision turning actuator. The following report entails some background information about the actuator itself and all progress made during the past several months, which includes the design process during the development of the GUI and any test results made with the finished interface. A number of appendices are attached to the end of the report as supplements.

2.0 The Precision Turning Actuator

The Precision Turning Actuator (PTA) is currently under development at UBC by Andrew Woronko. The main purpose of the actuator is to replace the final grinding process while turning a mechanical part such as the shaft of an airplane engine, although its precise characteristics can be useful for a number of other occasions. There are several potential customers in the industry for the actuator, the most significant ones being Pratt & Whittney Canada and Okuma Manufacturing.

2.1 Conventional Turning VS Precision Turning

A conventional turning process typically has four major steps, namely rough cutting, semi-rough cutting, finish cutting, and surface grinding. While the first three processes can be done on the same machine, the work piece must be dismantled and put onto another machine for grinding. This is not only time consuming, but neither does it produce a very precise surface finish.

With the precision turning actuator installed, cutting and surface grinding can be combined into one process: grinding will be replaced by performing a precise cut controlled by the PTA. The machined parts would then have a very fine surface finish with an error of plus or minus five nanometers. By performing precise cutting, a technician would be able to complete the whole manufacturing process on one CNC machine instead of relocating the parts for grinding. This would not only reduce the time and money that a company spends on the production line, but it would also allow the employees to work in a more environmentally friendly situation.

2.2 Background Information for the PTA

The PTA is surrounded by a solid housing with the cutting tool head extended out from one side and the power inputs from the other. Inside the actuator, a piezoelectric translator is mounted onto the flexure housing which would expand and provide motion to the tool head when the translator receives a voltage input. The piezoelectric translator in the PTA can accept a voltage input that ranges from 0 to 1000 volts with a maximum expansion of 40 micrometers. Additional clamping devices are installed inside the actuator to hold the piezoelectric translator in place while the tool head is cutting against a work piece.

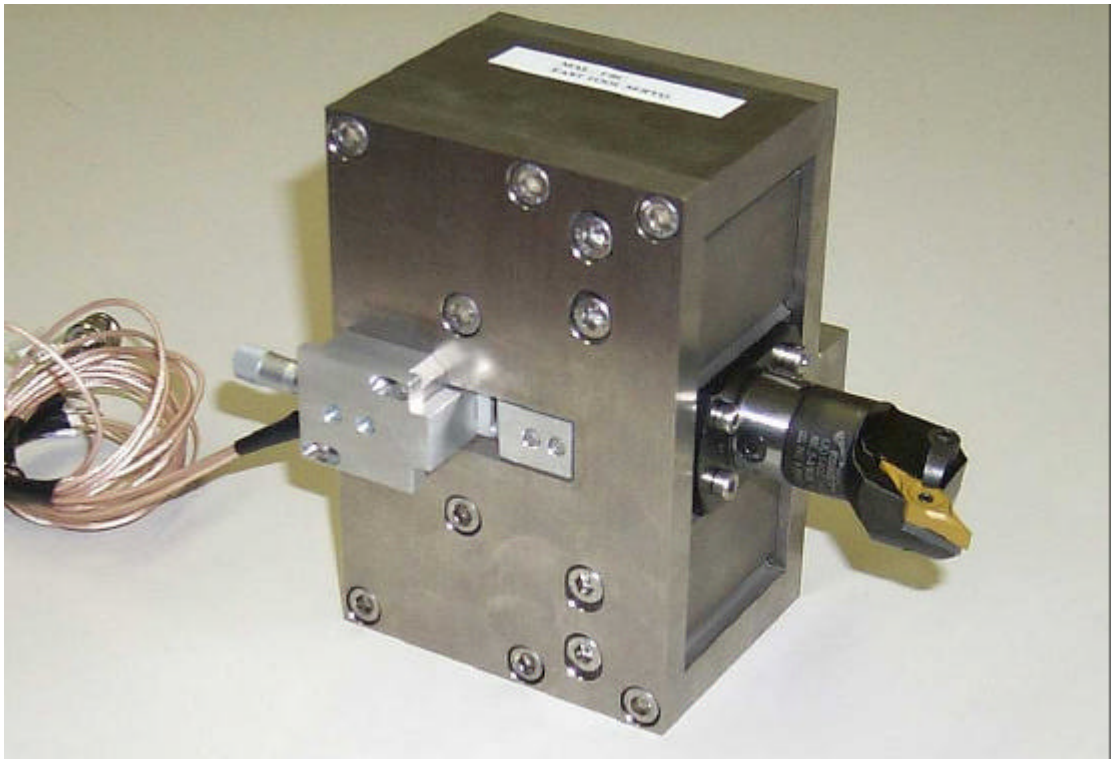

Figure 2.1: Close-up of the precision turning actuator

The actuator is carefully controlled by sending feedback signals from a capacitive positioning sensor to the computer. This high density sensor has a resolution of 10 nanometers and detects the actual position of the tool head in every sampling period. When the actual position of the tool head changes, different voltage signals will be generated by the sensor accordingly. The computer would then compare the voltage signals with the values calculated from the desired position. This closed loop process

would allow the PTA itself to adjust the tool head consequently, and as a result, increase the finishing surface quality of a machined part.

The flowchart below provides a general overview of how the PTA is controlled and how it is implemented with a closed loop system design.

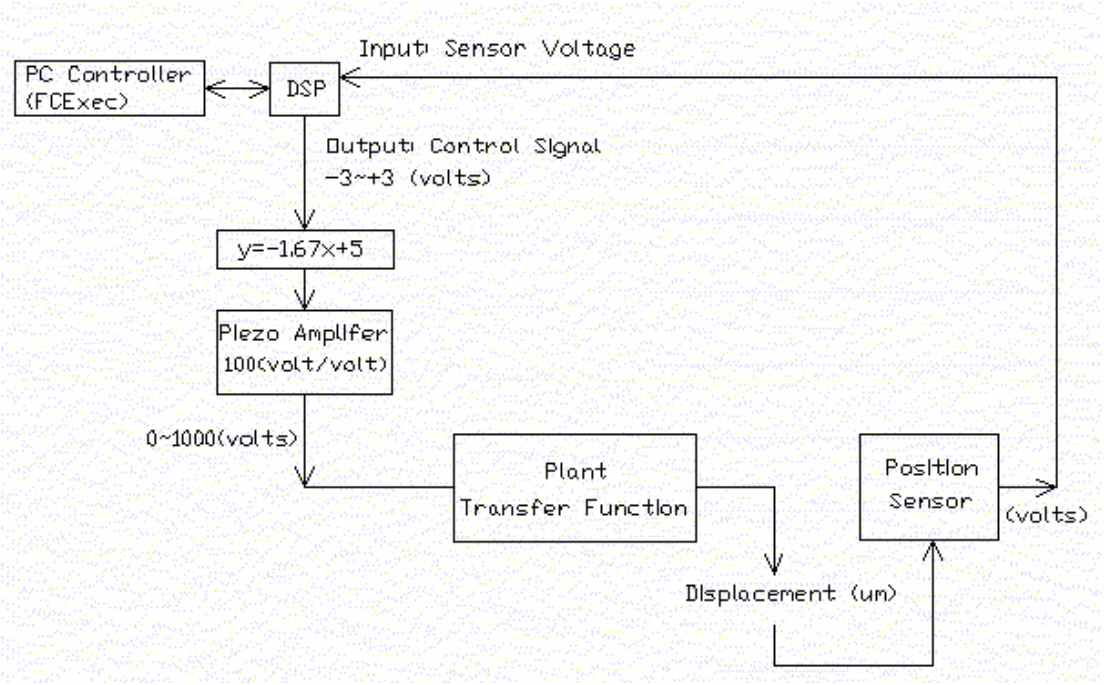

Figure 2.2: Simple schematic of the precision turning actuator

First, the voltage signals from the computer controller must be processed through a digital signal processing (DSP) board in order to make the signals discrete. These signals are then sent to two system amplifiers. The first amplifier offsets the computer generated signals to vary from 0 to 10 volts. The second amplifier with a gain of 100 volts per volt would then increase the signal magnitude by a factor of one hundred. With voltage signals range from 0 to 1000 volts, the input voltage requirements for the translator are satisfied. These amplified signals are then sent to the plant transfer function to output a relative displacement in micrometers so the piezoelectric translator can translate proper amount of displacement to the flexure housing and to the tool head.

At the same time, the capacitive sensor detects the actual position change of the tool head and generates feedback voltage signals. These output readings are sent back to the computer controller to complete a closed-loop process. With the conversion factor

of 5 micrometer per volt which is pre-determined for the control system, the computer would be able to compare the feedback actual position with the user desired position and adjust the next commending signal accordingly.

3.0 Design Process

The purpose of this project is to improve upon the existing user interface for the PTA. The project undergoes a similar procedure as most other design processes. The problem must be identified; considerations must be made for solving it. Only then can the product be created.

3.1 Identifying the Problem

The original user interface for the PTA is the Fast Cyclic Executive, or FCExec for short. It is a concise version of the ORTS (a real-time operating system for DSP boards), optimized for the PTA to decrease execution time and delays. The FCExec is written in a Visual C++ single document format, with a text area to display messages and a menu bar to issue start/stop commands. Control parameters (tool position, gains, etc.) are specified in the program code as constants. Changes made to these parameters will not take effect until the program is recompiled. A number of problems arise due to this limitation:

1. **Inconvenience** – the extra time spent due to recompilation for every use of the PTA stacks quickly and reduces efficiency significantly over time.
2. **Adequate programming knowledge required** – the user must be familiar with computer programming in order to make changes in the code manually; not all potential users of the PTA are trained in such.
3. **Compilers must be installed** – the Visual C++ and Texas Instrument C3x compilers must be installed on the PC in use, which is an unnecessary expense for the customer.
4. **Source code must accompany the product** – software developers usually withhold source codes from the public as a means to protect their work – something that cannot be done for the FCExec.

The compiled interface of the FCExec itself has very little functionalities apart from starting and stopping the PTA. A user-friendly interface in this case should allow the user to change any control parameter at will with the least amount of work involved.

Therefore, the graphic user interface should be in a self-explanatory, dialog-based format, which allows the user to issue commands and modify input values at a click of the mouse. A problem statement can be defined for this project:

> *To design a Windows based multi-functional user interface that allows the user to adjust the control parameters directly from the interface without changing and recompiling the program source code each time before activating the Precision Turning Actuator controller.*

3.2 The Fast Cyclic Executive

The compiled FCExec is composed of three files: *fcexec.exe*, *f3.out*, and *fcexec.ini*. *Fcexec.exe*, figuratively termed the "PC side", is compiled using Visual C++. It deals with the user interface, namely file input/output, visualization, start/stop command, and the like. It is what the user sees and interacts with when the program is running. *F3.out*, otherwise termed the "DSP side", is compiled using the TI C3X compiler. The file contains the control functions and directly communicates with the PTA through the DSP. It is the controller and can be seen as the "heart" of the system. All control parameters are specified in this file. *Fcexec.ini* merely stores initialization parameters such as sampling frequency and number of input/output channels in a plain text format.

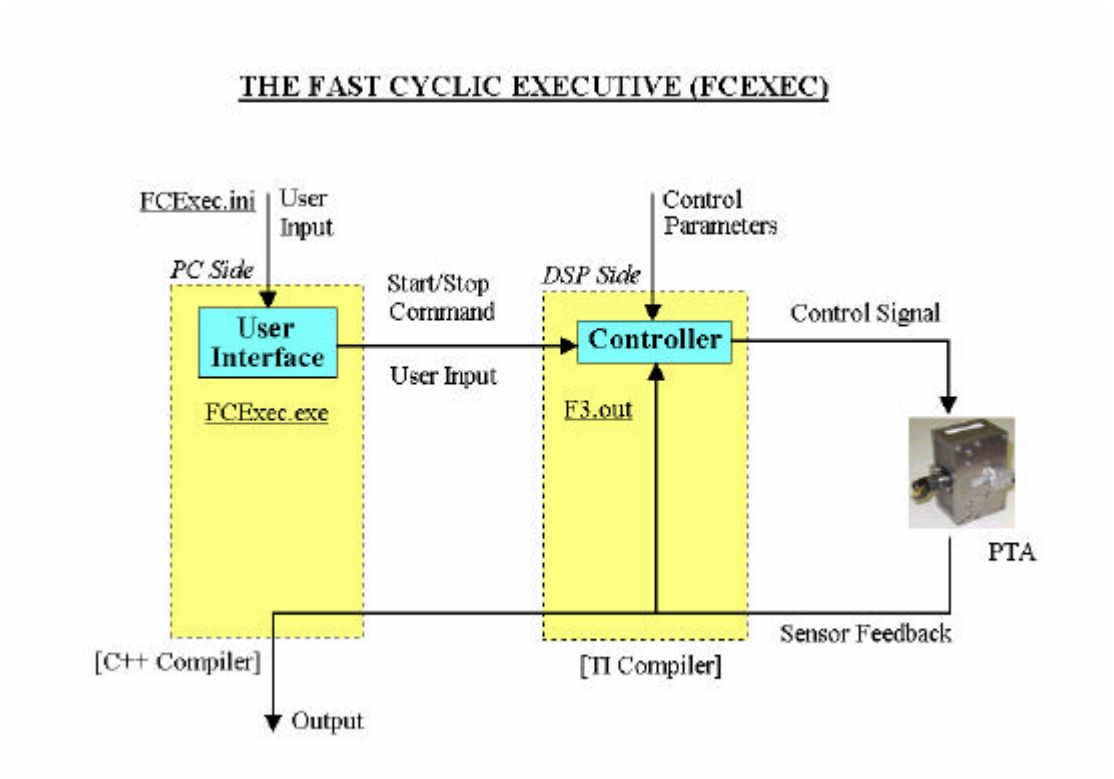

Figure 3.1: Simple schematic of FCExec

Figure 3.1 is a simple schematic that shows the program flow of FCExec. As *fcexec.exe* is run, initialization parameters are read from *fcexec.ini* automatically. As soon as the user issues the start command on the PC side, the parameters and the

command itself are sent to the DSP side, which contains the predefined control parameters. Calculations are performed by the controller and the resulting control signal (in volts) are sent to the PTA to position the tool. During the process, the position sensors within the actuator send error signals (also in volts) to the controller as feedback to close the loop. The process is repeated between the controller and the actuator until a stop command is issued from the PC side. As a reference, both control and feedback signals can be sent to the PC side to be printed in a text file should the user deem necessary (specified in *fcexec.ini*).

3.3 Design Considerations

As soon as it has been made clear how FCExec functions, a programming tool must be selected to make the necessary modifications to the interface. Two languages were put into consideration: Visual C++ and Visual Basic. To help the decision making process, a simple conceptual design is created for each of the two language models.

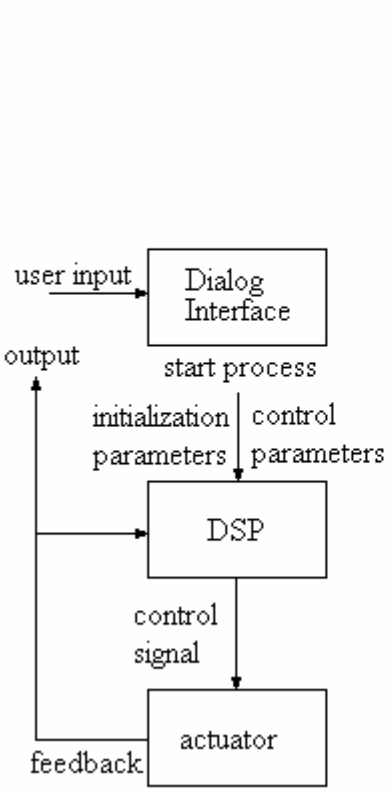

Figure 3.2a: The VC++ model

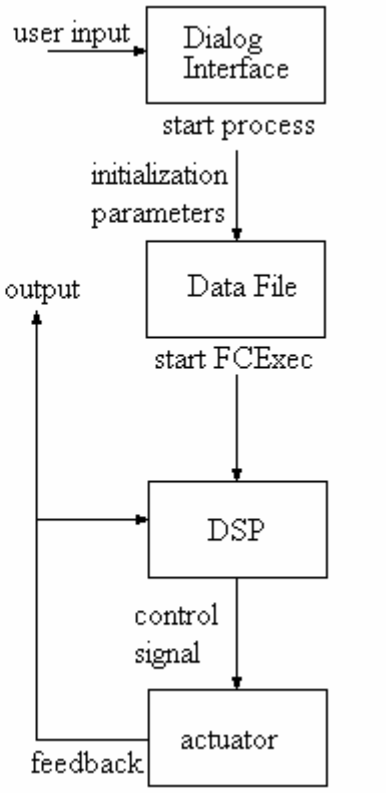

Figure 3.2b: The VB model

As seen in Figures 3.2a and 3.2b, the Visual C++ model has a distinct advantage of sending user input directly to the DSP through PC and DSP memory without an extra external file involved as in the Visual Basic model. However, using Visual C++ here has a major drawback. Since FCExec is built around an automatically generated single-document interface, converting it to a dialog-based format can essentially suggest a near-complete rewrite of the program. A design choice would have to be made for whether the omission of the external file is worth the extra time spent for rewriting the interface from scratch during the development cycle. Further pros and cons of each model are tabulated below.

	Pro	Con
Visual Basic	- Fast to assemble - Event-oriented: simple	- Possible reduction in processing speed - Additional step to create a link file
Visual C++	- Fast processing time - Consistent programming - No link file necessary	- Complex structure - Complete rewrite of the existing code

Table 3.1: Pros and cons of each language model

The main concern for the VB model is the time delay between the instant when the user clicks "Start" (or "Stop") and when the PTA control cycle actually begins (or ends). A test program was quickly assembled to assess the severity of such delay.

A dummy VB interface with only a "Start" and a "Stop" button was created to write twenty random values (more than eighteen in the real case) onto a file, followed by the execution of the original FCExec. It was observed that the delay was less than 0.5 seconds on the test machine. Compared to a hypothetical Visual C++ interface, which would have a near-instantaneous execution, the difference is found to quite small. Also, since the relative execution delay in VB when compared to VC++ is only noticeable for large numbers of computations, such factor can be neglected for the purpose of this project. This is due to the fact that the VB interface only performs minimal amount of computations *once* at each of the beginning and the end of a process. Therefore, the VB model was chosen to save time spent on writing the program in exchange for more actual tests done with the PTA later.

3.4 Core Modifications

Visual Basic has now been selected as the language for the graphic user interface. The next step is to create the GUI and make sure that the program works at least as well as the original. The dialog interface itself is, in fact, easily assembled; it is the modifications made to the original FCExec for the purpose of integrating the two parts seamlessly together that requires the most care.

3.4.1 Data-Passing Technique

The single most important step to solving the problem is to understand how variables are transferred between the PC and DSP sides. Since FCExec already reads and uses raw data from the file *fcexec.ini*, it is logical to follow the flow of those data through the code. One item in the *ini* file is the number of input channels, which undergoes a similar procedure as intended for the new data.

The procedure begins with a data reading function found in *PCThread.cpp*. Once the data have been converted into floating point form, they are sent to *genericdsp.cpp* using the *SendCommandToDsp* function with a particular command in the form of:

dspObject -> SendCommandToDsp(<command>, <parameters>)

where <command> is the command name and <parameters> are the values to be passed. For example, in the case of the number of channels, the line becomes:

dspObject -> SendCommandToDsp(CMD_NofChannels, &nofchannels)

The command mentioned above is defined on each of the PC (in *genericdsp.cpp*) and the DSP (in *dsp_main.cc*) sides since each file act as gateways for it respective side. When the command is given by *PCThread.cpp* using the *SendCommandToDsp* function, the tasks listed under the command definitions are performed sequentially on the PC side then on the DSP side.

For the purpose of passing values between the two sides, again using the number of channels as an example, the command line is *CMD_NofChannels*. In *genericdsp.cpp*

it is given the task to store the value in the parameter *nofchannels* to a predefined memory address, named *NOF_CHANNELS_MEMORY_LOCATION* in the same file. Once completed, the *CMD_NofChannels* defined in *dsp_main.cc* on the DSP side then performs its own task, which is to read from the same memory location. To make use of the newly acquired value for the DSP, it simply uses basic function calls to pass it as a parameter to either of *userprcs.cc* and *piezoprcs.cc.*

The modifications made to accommodate the new VB GUI essentially follow the same procedure. The raw data in the file written by the Visual Basic interface is read in PCThread.cpp with the following standard I/O functions:

```
FILE *stream = fopen( "piezoprcs.dat", "r" );
fscanf( stream, "%f", &KD_read );
fscanf( stream, "%f", &KP_read );
…
fclose( stream );
```

Then, with the line below:

```
dspObject -> SendCommandToDsp( CMD_Gain, &KD, &KP, &KI,... )
```

the recently read variables are passed using a newly defined command, *CMD_Gain.* Under the command definition on the PC side, in *genericdsp.cpp*:

```
case CMD_Gain:

        float KD_read, KP_read, KI_read;
        va_start( argList, command );        /* Initialize variable arguments. */
        KD_read = *(float *)va_arg( argList, float *);
        KP_read = *(float *)va_arg( argList, float *);
…
        va_end( argList );                   /* Reset variable arguments.      */

        WriteFloat(kd_address, KD_read);
        WriteFloat(kp_address, KP_read);
  …
        WriteLong(command_address, CMD_Gain);
        WriteLong(synchronization_address, SYN_CommandReady);

        reply = 0;
```

```
        while(reply != SYN_Success) reply =
ReadLong(synchronization_address);
        break;
```

is used to fill in the newly created memory locations (declared in gerericdsp.cpp) with corresponding values. And on the DSP side, in *dsp_main.cc*:

```
case CMD_Gain:
        PROCESS_KD = *KD_MEMORY_LOCATION;
        PROCESS_KP = *KP_MEMORY_LOCATION;
  …
        *SYNCHRONIZATION_MEMORY_LOCATION = SYN_Success;
        AssignUserKD(KD);
        AssignUserKP(KP);
…
        break;
```

reads from memory and passes them to the control functions in *piezoprocs.cc* and *userprcs.cc*. A complete list of changes made and accompanying notes is presented in Appendix A for clarification.

3.4.2 Interface Linking

Once the FCExec successfully reads data from a file and makes use of them, it is time to link the new Visual Basic interface with the modified FCExec. As the user starts the PTA (clicks the Start button), all values that appear on the GUI are saved to an external file (named *piezoprcs.dat*) using Visual Basic standard file I/O methods. The file can be opened and read by FCExec as the source of data.

For running FCExec and beginning the cycle after *piezoprcs.dat* has been saved, the method used is a simulation of key pressing. Similar to most document-based programs, FCExec has a menu bar with a hotkey wired to each item on the menu. After *fcexec.exe* is called by the GUI, a series of signals simulating key press are given to the FCExec interface, as if the user is typing those keys manually. For example, the Start command on FCExec is under the File menu. To execute that command, the keys "Alt-F" are pressed to show the menu, then "S" is pressed to start the cycle. Visual Basic can simulate these by using the line *SendKeys "%FS"* to run the cycle automatically. To end the process, a similar procedure is done for stopping (*SendKeys "%FP"*) and exiting (*SendKeys "%FX")* FCExec.

3.4.3 Controller Selection

An additional controller type, the sliding mode controller, has been in use since the end of last year. Unlike the PID controller, it uses three different gains designated lambda (λ), row (ρ), and K_s. Since either controller can be useful for a given situation, the user should be given the freedom to switch between them.

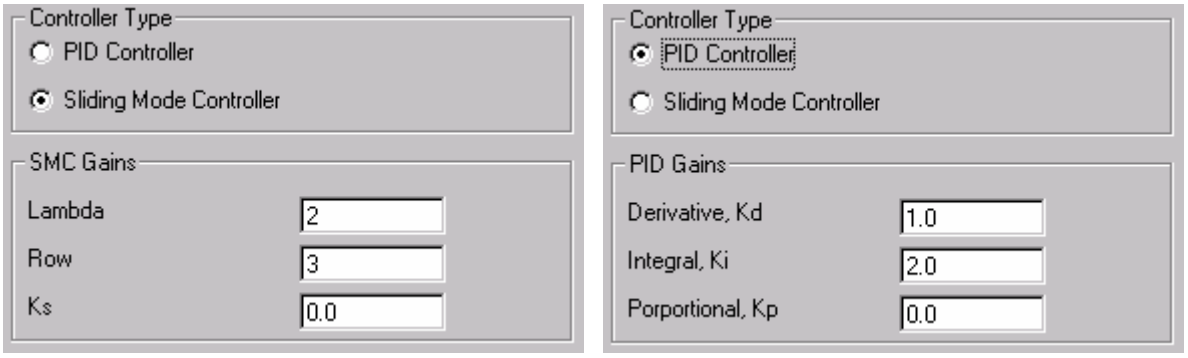

Figure 3.3: Switching between the PID controller and the SMC

For the VB GUI, two additional frames are added: *SMC Gains* and *Controller Type*. *SMC Gains* is essentially a replica of the *PID Gains* only with its own gains, whereas the *Controller Type* frame contains two buttons, labeled with each of the corresponding controller types. On the interface, the controller frames (*SMC Gains* and *PID Gains*) overlap one another, depending on which of the two buttons is selected. The code follows the form:

```
Private Sub OptionControlType_Click(Index As Integer)
Select Case Index
  Case 1:
  FrameSMC.Visible = False
  FramePID.Visible = True

  Case 0:
  FramePID.Visible = False
  FrameSMC.Visible = True
End Select
End Sub
```

As indicated, the two buttons (or "options") are members of a same array. The variable *Index* denote whether the PID button is ON: if it is, *Index*=1; otherwise,

Index=0. Since only one of the buttons can be ON at any given time, it is convenient to use *Index* as an indicator. When *Index*=1 (PID is ON), the *PID Controller* frame is on the top (visible) and the other is hidden, and vice versa. When the user clicks on the "Start" button, a Boolean value indicating whether the PID controller is selected along with the SMC gains are written on *piezoprcs.dat*, and the same data reading/passing procedure is undergone as all previously mentioned values (see the Jan 2nd report for details).

On the FCExec front, new changes are made for *userprcs.cc* and *piezoprcs.cc*. In the former, where control functions are called, the following lines are added:

```
if (bController == 1)
        PID_Control(AdcIn, …, alpha, 1, Disp, KD, KP, KI);
else
        SMC_Control(AdcIn, …, alpha, 1, Disp, Lambda, Row, KS);
```

The variable *bController* contains the same controller type value read from *piezoprcs.dat* as explained earlier. Namely, *PID_Control()* is called when *bController* is 1, and *SMC_Control()* is called when it is 0. The two control functions are defined in *piezoprcs.cc*, with few changes made other than the addition of the SMC controller.

3.4.4 Override Control

The override function gives the user the choice to maintain the output to a specific output signal as an open loop system. In the original FCExec, this is achieved by using

DacOut[<#>] = <constant>

Where # indicates the member of an array (0~3, corresponding to the number of output channels used) and *value* is the desired output voltage (±0~3 volts).

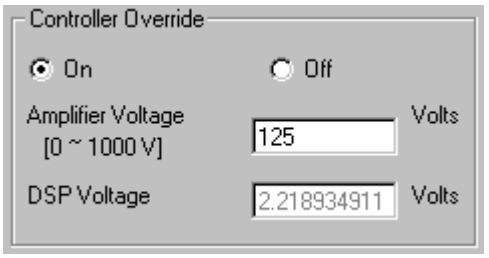

Figure 3.4: Controller Override in the ON state

The override function in the interface has 2 main features. It allows the user to turn ON/OFF override at will, as well as converting an amplified input voltage to an amplitude acceptable for the DSP. To perform these tasks, the VB interface writes two values to *piezoprcs.dat* from which FCExec can read and implement them. These values are defined as *bOverride* and *Override* in FCExec, which provide the program with the information of whether to use an override or not and the actual override value itself, respectively.

If the user wishes to use a manual override, he needs to first click on the ON button in the "Controller Override" frame on the GUI (default is OFF). This enables the first text box on the same frame in which he can input the amplified override voltage as desired. The next text box contains DSP voltage and is always disabled (grayed out). FCE_GUI automatically calculates and converts the user override to a voltage tolerable by the DSP, and then it places the result into the DSP voltage box. The calculated value is the one written in *piezoprcs.dat* instead of the amplified voltage. The conversion is done with the using the formula

DSP Voltage = (1 / 1.69) * (5 - <Amplified Voltage> / 100)

Once the override data values have been written in *piezoprcs.dat* and read by FCExec, they undergo the same memory passing procedure as explained earlier. On the DSP side of FCExec, a simple statement is written to inform the program of whether to implement user override or not:

```
if (bOverride == 1)
        DacOut[0] = Override;
```

3.4.5 File Output Control

A file that includes the complete history of actuator position and voltage from the beginning to the end of an operation can be helpful for research purposes. The data contained within can later be plotted should the user finds the need to do so. However, such feature is not always necessary. If the user merely wishes to see the physical product of the machine, data output becomes relatively useless. In fact, it can be a nuisance as mapping a large number of data points onto a file cost valuable time and computer resources. Consequently, a function that allows for a control of the output is included for the GUI. Figure 3.5 shows the difference between the ON and OFF states of the output frame.

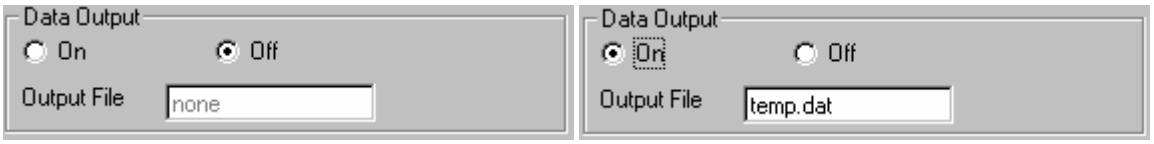

Figure 3.5: The Data Output frame

The procedure for adding this function is quite simple. In FCExec, two files are responsible for the data writing process: *PCThread.cpp* (PC side) reads the output file name from the last line "output=<filename>" in *FCExec.ini*, and sends the acquired filename to the DSP side of the program; *Userprcs.cc* (DSP side) calls the function *SendFloatToHost()* afterwards to the open a file with that name for writing. If the line reads "output=none", no output file is created. However, the function call *SendFloatToHost()* must also be remarked at the same time lest the program hangs on stopping.

To avoid this error, the VB interface can be edited to write a Boolean value to *prezoprcs.dat*. Namely, if FCExec reads 1 from the file, an if-statement on the DSP side executes *SendFloatToHost()*; otherwise, no output is written. On the GUI, the user is able to switch between YES and NO for the output. If the NO button is selected, the text box below is disabled and displays "none", and "output=none" will be written to *FCExec.ini* when the start button is clicked. If the YES is selected, the user can then type a desired name for the output file or use the default name *temp.txt*.

3.5 Secondary Features

Several additional features are given to the GUI to aid the user in several different ways. These include graphic capabilities, saving/loading default values, and safety features during input.

3.5.1 Graphing

Once an output file is created, the user can have the data points plotted into graphs to inspect the process visually. Two sets of data are presented in the output file: tool position (in volts) and control signal (also in volts). These are both functions of sampling time, given by the inverse of the frequency. The latter is to be plotted as it is, while the former must be converted into microns using the formula:

$$Position_{microns} = Voltage_{DSP} * 5$$

Since a graphing utility written in Visual Basic by a former graduate student was already present, it was convenient to incorporate it into the FCE_GUI with some modifications. This utility consists of two parts: one contains all the class definitions and function operations; the other is a single sub whose main function is to feed the former with data points. The modifications made are in the latter part only.

The first step to plotting a graph is to acquire data points from the output file. However, since a new output file with different lengths (number of points) is generated every time the program is executed, it is necessary for the graphing tool to know the length of the file beforehand. Therefore, a *while*-loop is written to perform the task:

```
NbrSteps = 0
While Not EOF(1)
  Input #1, column1, column2
  NbrSteps = NbrSteps + 1
Wend

NbrSteps = NbrSteps – 1
```

Where *NbrSteps* is the number of data points, and *column1* and *column2* are tool position and DSP voltage respectively. It is sometimes the case when FCExec is cut off while in the middle of writing a line to the output file. Such an incomplete file, when plotting is attempted, can result from plotting meaningless data to an execution error. To prevent this, the line *NbrSteps = NbrSteps – 1* is added to the code to neglect the bottommost line altogether. Since all processes are stopped after steady state has been reached, neglecting the last line will not have significant, if any, impact on the plotted result.

After the number of data points has been acquired, each point will be read individually and be drawn on the graphs with the following block:

```
Dim data_top() As Double
ReDim data_top(1 To 2, 1 To NbrSteps)

Dim data_bottom() As Double
ReDim data_bottom(1 To 2, 1 To NbrSteps)

For i = 1 To NbrSteps
  data_top(1, i) = i * 1 / Val(TextFreq.Text)
  Input #1, temp
  data_top(2, i) = temp * 5
  data_bottom(1, i) = data_top(1, i)
    Input #1, data_bottom(2, i)
Next i
```

Data_top() and *data_bottom()* are arrays of variables that refer to data points for tool position and DSP voltage respectively. *Data_top(1, i)* defines the x-axis (sampling time) for the tool position graph, and the two lines immediately below read data for line i and assigns it to the y-value (tool position); d*ata_bottom()* works in a similar way. This loop continues until the second-last line of the output file has been reached.

Finally, all these points are sent to the core of the graphing tool using the line

```
Call FCEGraph.Set_data("Plot", data_top, data_bottom)
```

A new window will appear on the foreground with two completed graphs.

A problem exists when the number of data points is too large: the drastic decrease in speed can be rather disappointing. A mere three-second process can take up to thirty seconds of loading time when it comes to graphing – a most undesirable phenomenon. A proposed solution is to rewrite the code in a way such that only one in an adjustable number of points is read depending on the total number of rows in the output file. This can be achieved by skipping a number of *for*-loops, although its drawback is that many data points will not show on the graphs.

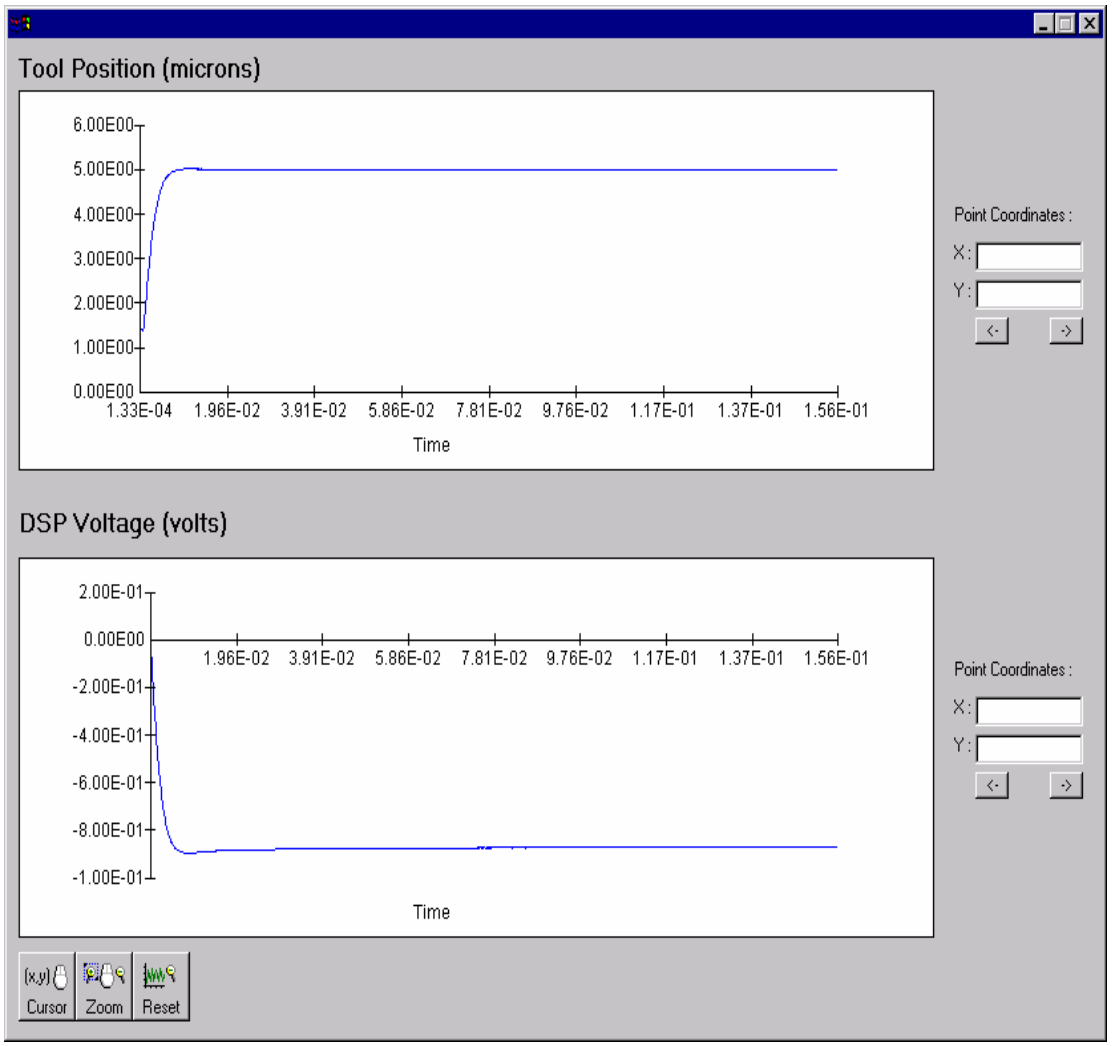

Figure 3.6: Plotted output graph

3.5.2 Saving and Loading Default Values

There are times when the user wishes to switch between different sets of control parameters repeatedly while using the PTA. It can become frustrating as well as inefficient if manual inputs are required each time. To make it more convenient for a user in such situation, a save/load function is incorporated into the interface.

The save/load functions, in fact, shares the same subroutines with those used to read/write *piezoprcs.dat*. With standard Visual Basic dialog control functions, the user can chose files and save to and load from. The files are in plain text format and are identical to *piezoprcs.dat* for simplicity. Figure 3.7 shows the save/load function in action.

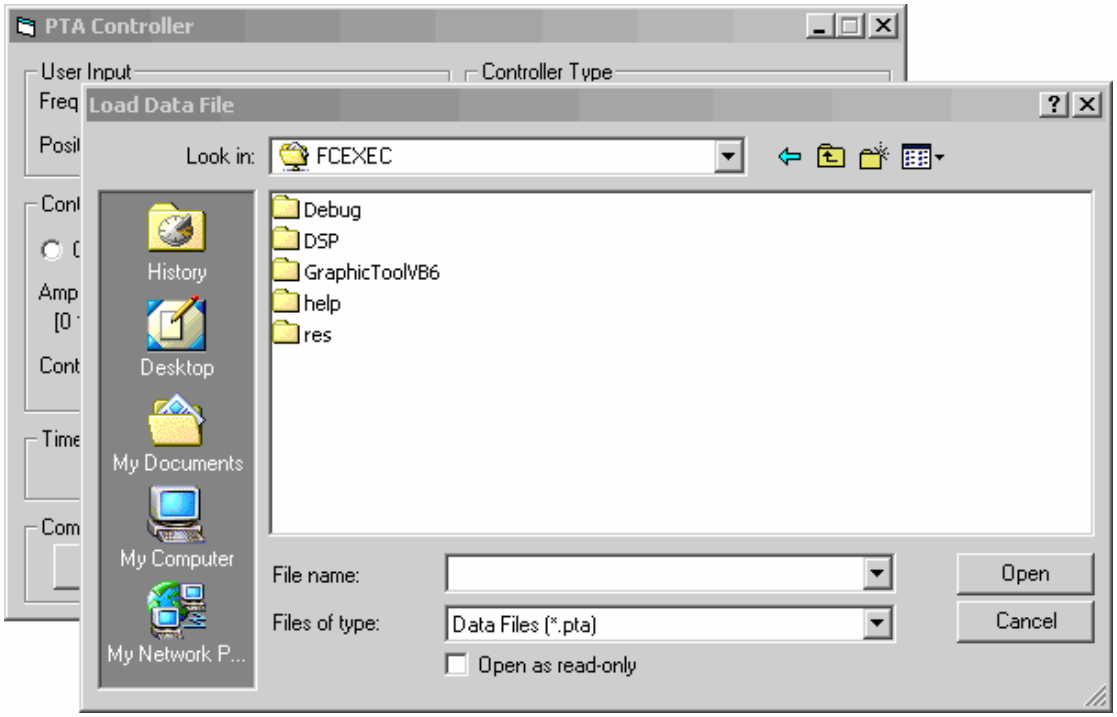

Figure 3.7: Loading values from file

3.5.3 Safety Features

It is safe to assume that the user will eventually make a mistake during the use of the PTA Controller. This usually takes the form of an invalid input in one of the text fields on the GUI. Since the GUI saves all that appear in the textboxes to *piezoprcs.dat*, should they contain anything other than numbers, such as characters and symbols, catastrophic failures may occur as FCExec tries to read and convert them into floating point or integer values.

To make sure that such errors do not occur, safety features have been installed to the GUI. For each textbox a subroutine similar to the following is written:

```
Private Sub TextFreq_KeyPress(KeyAscii As Integer)

  If Not (KeyAscii = vbKeyBack Or (KeyAscii >= vbKey0 And KeyAscii <=
        vbKey9)) Then
    KeyAscii = 0
  End If

End Sub
```

This effectively eliminates any margin for error since nothing will appear in the textbox when any key other than backspace and the number keys are pressed. The above example is for the sampling frequency box, in which a position integer is expected. For the PID and SMC gains, a period (.) is allowed for floating points. An additional minus sigh (-) is allowed for the desired actuator position input. The value in the amplifier voltage textbox is limited to integers within the range of 0 to 1000 volts.

3.6 The Completed PTA Controller

With all of the above features successfully incorporated, the new GUI is now complete. It is given the name "Precision Turning Actuator Controller", or the PTA Controller for short. The figures below show a brief explanation of the interface and the PTA Controller in action.

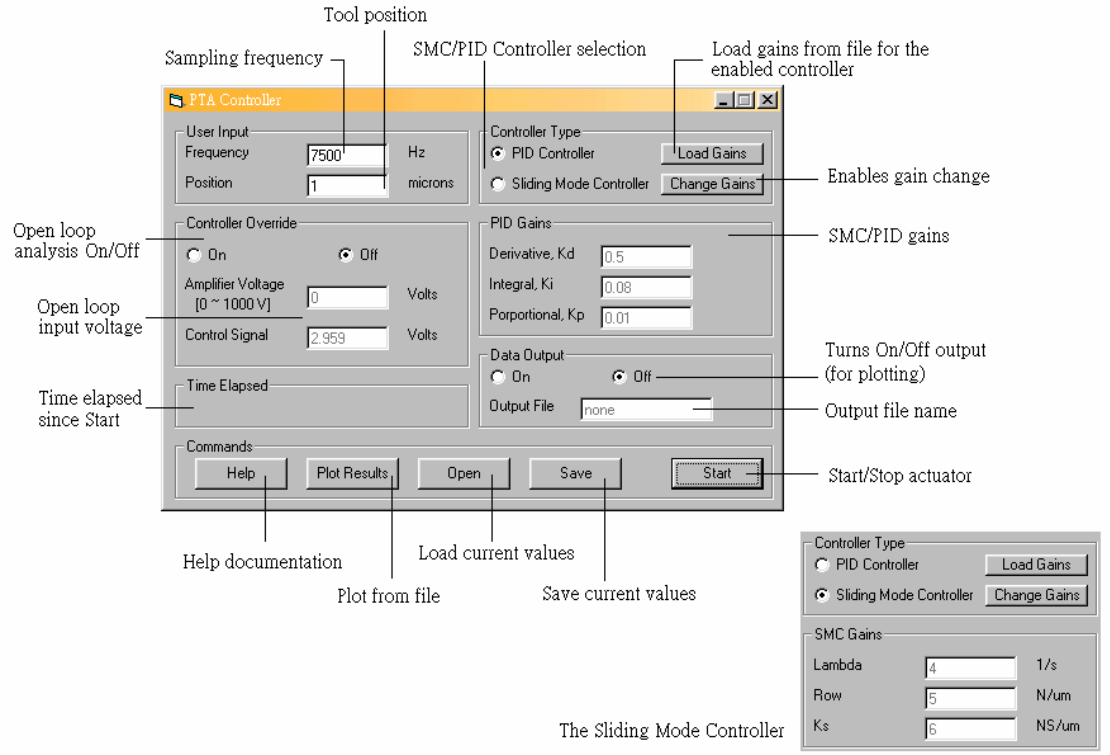

Figure 3.8: The PTA Controller

Figure 3.9: The PTA Controller in action

4.0 Interface Testing and System Response Analyses

4.1 Open Loop Response Analysis

In order to control the PTA precisely, the closed loop transfer function for the system must be identified. However, it is necessary to examine first the uncontrolled dynamic behavior of the system and determine the open loop plant transfer function. This can be achieved by supplying the actuator a step input voltage and observe the dynamic response of the machine.

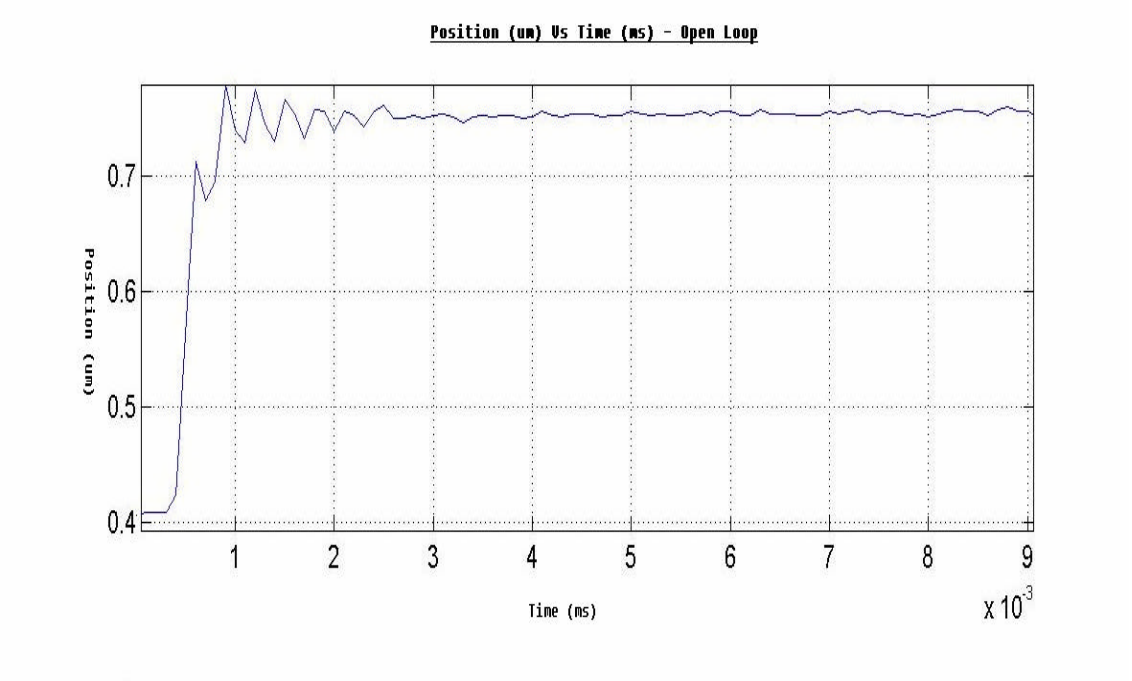

Figure 4.1: Open Loop System Response

As shown in Figure 4.1 above, the PTA is supplied with 0.1 volt step input from a signal generator, and the actual position of the tool head in micrometer is plotted against the time elapsed in millisecond. By observing the system's overshoot behavior, one would notice that the system has the property of a third order system. However, it is very common to simulate the dynamic response as a second order system for simplicity and calculation purpose. The system's damped natural frequency and damping ratio can be found by examining the system's response, and the values are calculated to be 1.996E-4[rad/s] and 0.037 respectively. The steady state error, Xss, is also determined to be about 790 nanometers.

As shown in Figure 4.2 below, the open loop block diagram is established with the two system gains, the standard open loop plant transfer function, position output, and voltage input.

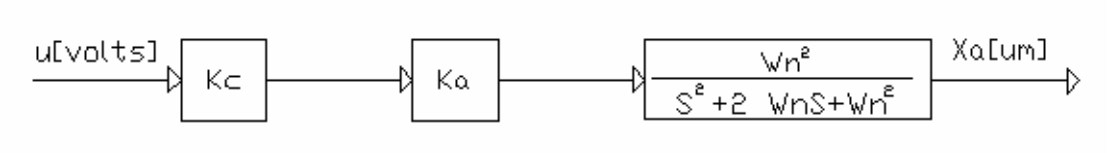

Figure 4.2: Open Loop Block Diagram

As a result, the desired open loop transfer function can be determined as,

$$Gp(s) = \frac{Xa(s)}{U(s)} = \frac{KcKaWn^2}{S^2 + 2xWnS + Wn^2}$$

and the desired system characteristic equation is found to be $S^2 + 2xWnS + Wn^2$.

To simulate the response of the actuator, Newton's second law is applied. The total force induced by a dynamic system can be expressed as follows,

$$f(t) = M\frac{d^2x(t)}{dt^2} + C\frac{dx(t)}{dt} + Kx(t)$$

Change this into Laplace domain for system simulation purpose, we get,

$$F(s) = MS^2X(s) + CSX(s) + KX(s)$$

For the PTA system, the voltage input replaces the force term in the equation above, and the machine open loop plant transfer function is found to be,

$$Gp(s) = \frac{Xa(s)}{U(s)} = \frac{1}{MS^2 + CS + K}$$

where, Xa(s)[m], U(s)[volts], K[volt/m], C[volt/ms], and M[volt/ms^2].

We are now able to find the open loop plant transfer function by comparing the desired characteristic equation with the actual system's characteristic equation, and the system parameters M, C, and K are found to be 0.09435[volt/ms^2], 139.36[volt/ms], and 3.76E7[volt/m] respectively. The detailed calculations are derived later in Appendix B.

As a result, the final form of the open loop plant transfer function for the PTA is expressed as follow,

$$Gp(s) = \frac{Xa[m]}{U[v]} = \frac{1}{0.09435S^2 + 139.36S + 3.76 \times 10^7}$$

4.2 Closed Loop Response Analysis

After the open loop transfer function is determined, it is now necessary to find the closed loop transfer function of the whole system in order to correctly control the PTA and provide precise tool head positioning.

In Figure 4.3 below, a schematic diagram of the computer controlled PTA system is shown. The right hand side box includes the system gains and the plant transfer function and represents the actual machine; the left hand side box which represents the computer control algorithm compares the feedback signals with the desired position input of the tool head, and the closed loop transfer function would adjust the commending voltage accordingly.

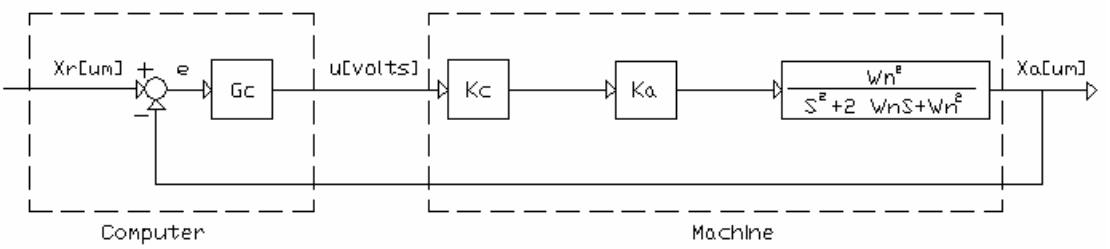

Figure 4.3: Closed Loop Block Diagram

The closed loop transfer function, Gc, is shown in Appendix C, and it is provided by the graduate student, Andrew Woronko who had finished design the control algorithm for the PTA system. However, due to design originality purpose and also the complexity involved in finding the closed loop transfer function is beyond the understanding of the MECH 457 students, the detailed calculations and analysis of determining the closed loop transfer function will not be discussed here. Nevertheless, complete tests on the closed loop system behavior have been successfully conducted and the results of the tests are explained in detail in the following section.

Closed Loop Response Tests (PID Control)

With the control algorithm determined and provided, the students are now able to examine the closed loop dynamic behavior of the actuator. In each experiment, the students have determined the rise time, percent overshoot and steady state error of the controlled system. The governing design criteria is to have a fast response (small rise time) with a small percentage of overshoot.

The students have implemented several different combinations of the proportional, integral, and derivative gains for controlling the PTA, and with the help from the user interface, the control engineers would be able to conduct various tests conveniently so the best combination of the PID controller can be determined.

PID Control Number 1

With the help from past experimental results, the students carefully select three combinations of the PID gains. The first test is conducted with 0.01 proportional gain, 0.03 integral gain, and 0.5 derivative gain. The position response in micrometer versus time in millisecond is shown in figure 4.4 below.

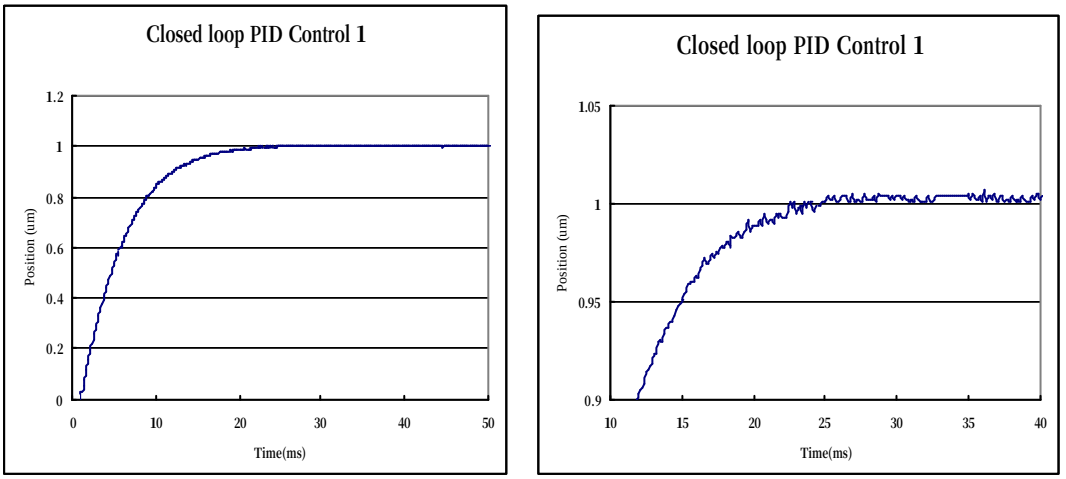

Figure 4.4: Closed Loop PID Control #1

With this PID combination, the system has a response with 22.8 (ms) rise time and 0.41% overshoot. The system parameters are also determined and are summarized in table 4.1 on page 33.

Parameter	Value	Parameter	Value
T(rise)	22.8 (ms)	Mp	0.41%
Time Constant	6.13 (ms)	Wn	232.185 (rad/s)
Tp	36.13(ms)	Damping Ratio	0.8686

Table 4.1: PID Control #1 Result

PID Control Number 2

The second test is conducted with a proportional and derivative gains unchanged but have the integral gain doubled to be 0.06. The position response in micrometer versus time in millisecond is shown in figure 4.5 below.

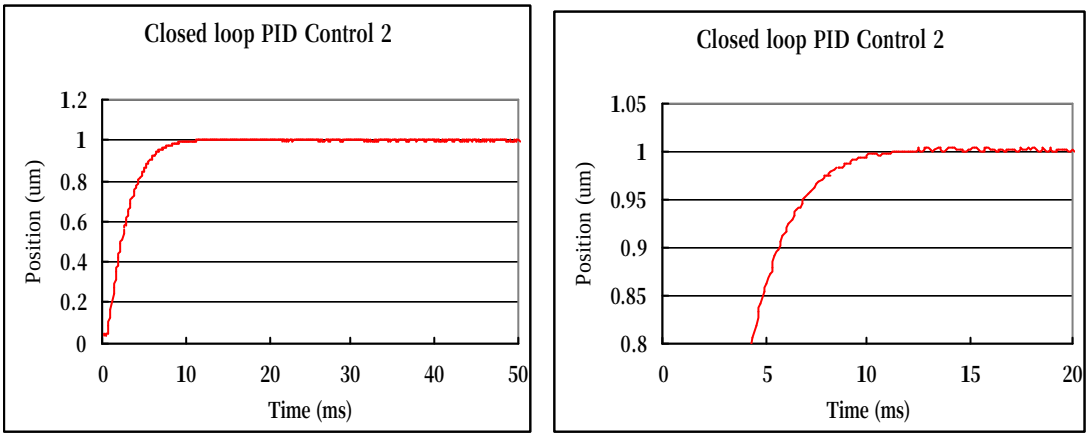

Figure 4.5: Closed Loop PID Control #2

With this PID combination, the system has a response with 11.47 (ms) rise time and 0.41% overshoot. We noticed that with the integral gain doubled, the system's rise time is reduced by half with the percent overshoot unchanged. The system parameters are also determined and are summarized in table 4.2 below.

Parameter	Value	Parameter	Value
T(rise)	11.467(ms)	Mp	0.41%
Time Constant	3.067 (ms)	Wn	461.632 (rad/s)
Tp	24 (ms)	Damping Ratio	0.8686

Table 4.2: PID Control #2 Result

PID Control Number 3

The third test is conducted with the proportional and derivative gains unchanged and has the integral gain increased further to 0.08. The positions in micrometers versus time in millisecond are shown in Figure 4.6 below.

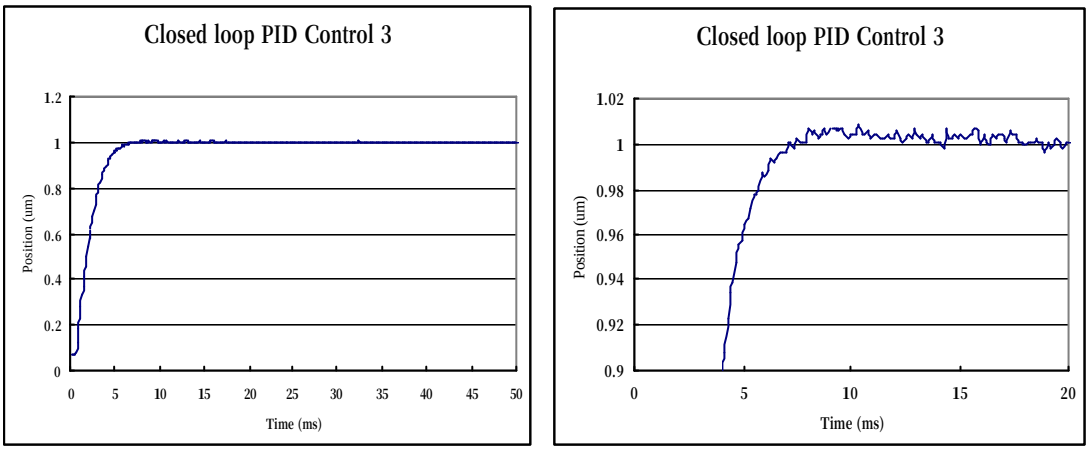

Figure 4.6: Closed Loop PID Control #3

With this PID combination, the system has a response with 7.07 (ms) rise time and 0.81% overshoot. We noticed that with the integral gain increased to 0.08, the system's rise time is reduced further; however, the response now has a percent overshoot twice as large as the previous cases. The system parameters are also determined and are summarized in table 4.3 below.

Parameter	Value	Parameter	Value
T(rise)	7.067 (ms)	Mp	0.86%
Time Constant	2.4 (ms)	Wn	656.68(rad/s)
Tp	10.4 (ms)	Damping Ratio	0.8344

Table 4.3: PID Control #3 Result

By comparing the results of these three experiments, the students find that the system would have a fast response of 11.47 (ms) and a small percent overshoot of 0.41% with the proportional, integral, and derivative gains to be 0.01, 0.06, and 0.5 respectively. Further experiments can be conducted by changing the combination of PID values to acquire a more stable, faster response, and smaller percent overshoot behavior of the PTA system.

4.3 P-270 Amplifier Frequency Analysis

Another system test that the students have conducted during the second term is on examining the frequency response of the piezoelectric translator (PZT). The manufacturer of PZT has provided a frequency diagram indicating the sampling frequency limits for a given voltage input (Appendix D). These theoretical frequency limits are plotted on a logarithmic scale with different translator capacitance shown in Figure 4.7 below. However, it is quite impossible to reach the theoretical values when performing experiments due to system and background noises.

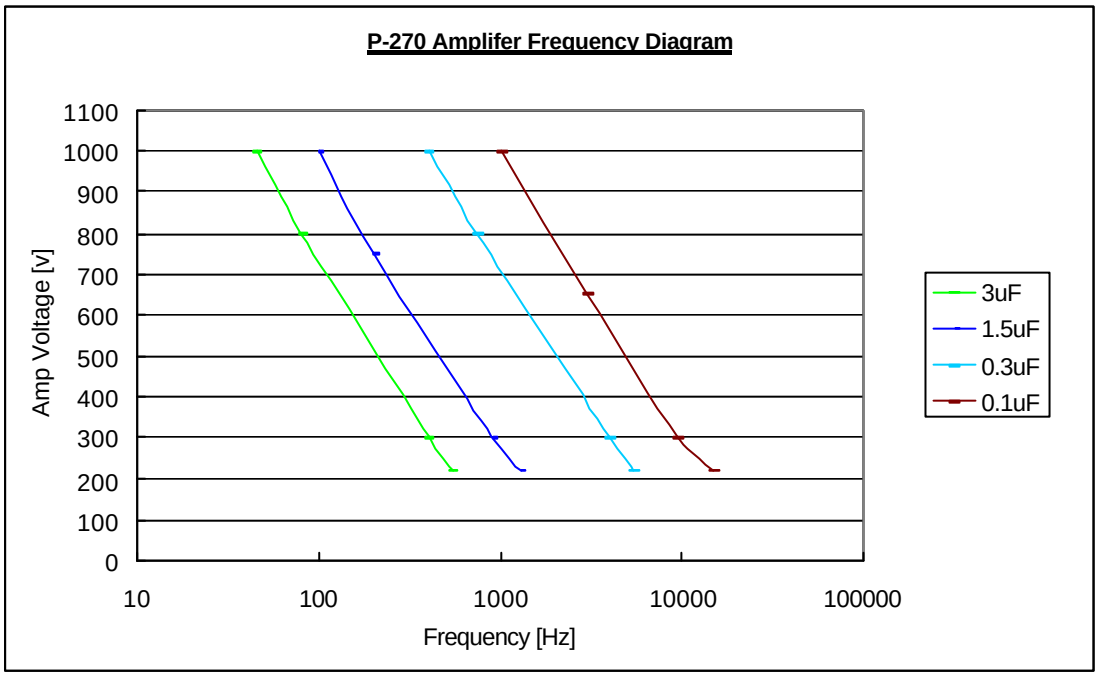

Figure 4.7: Theoretical Frequency Limits Diagram (manufacturer)

The theoretical maximum frequency for a given capacitance and voltage input can be calculated by the equation below.

$$f(\max) = \frac{I(\max)}{pCU_{(p-p)}},$$

For the PZT installed inside the actuator, the translator capacitance is 1(uF) and the maximum acceptable current input is 500(mA). The theoretical frequency limits for a given voltage input can then be calculated from the governing equation above.

To test the frequency limits of the translator, the students input several different step voltages into the PZT and adjust the frequency signals at the same time. A total of seven different voltage inputs are used to conduct the test, namely, 10v, 7v, 5v, 2.5v, 1v, 0.5v, and 0.3v. These voltages are then amplified by a factor of one hundred to test fully the responses of the translator (0v to 1000v). The students examine the frequency response for each voltage input by starting with a low frequency, for example 10Hz, and carefully increase the frequency signals until the PZT reaches its power limit. These results of the frequency tests are also plotted on the logarithmic scale diagram below and compared with the theoretical curve for the 1(uF) capacitance translator. The detailed experimental data are shown in Appendix E.

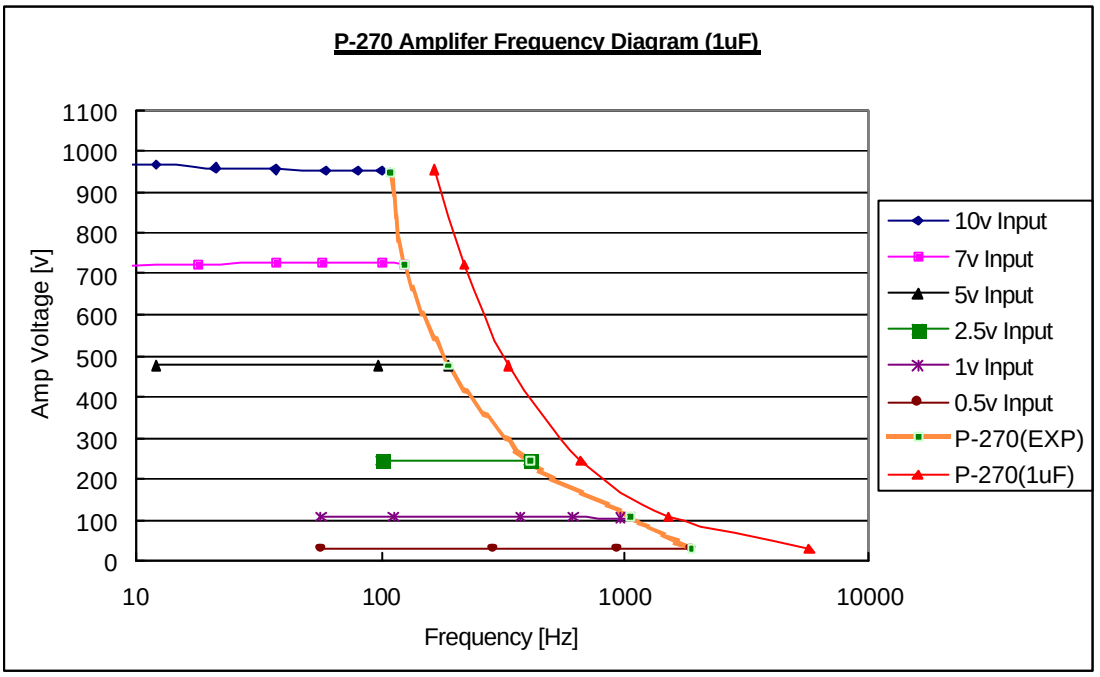

Figure 4.8: Experimental Versus Theoretical Frequency Limits (1uF)

A constant difference between the theoretical and experimental frequency limits can be examined from figure 4.8 above. This offset is present basically due to the material property of the capacitance. In real time experiments, the capacitance installed inside the translator does not always stay the same; in fact, the capacitance transforms a little when receiving a different voltage input. This non-linear hysterisis behavior of the translator would also affect the accuracy of the whole PTA system, and this can be carefully adjusted and resolved when a closed loop control system is implemented in controlling the behavior of the piezoelectric translator.

4.4 Cutting Tests

The students also perform several real time machining tests by using the PTA with the help from the completed user interface at the end of the second term. The work piece used is SAE 4340 steel (36 HRc) with 60 (mm) to 62 (mm) in diameter. With the cutting speed of 150 (m/min), the desired spindle speed is carefully selected by using the equation below.

$$n[rpm] = \frac{V[m/\min]}{pD[m]} = \frac{150}{0.06p} = 795.8[rpm]$$

The federate of the cutting tool is selected to be 0.05 (mm/rev), and the cutting insert used is made of Valenite 35deg. Diamond.

Four different intended depth of cuts, namely 1(um), 10(um), 18(um), and 20(um), are performed for the cutting tests.

It is first necessary to make a rough cut to get rid of the eccentricity problem of a work piece, and the new diameter is found to be about 61.44(mm) after the rough cut. This rough cut is entirely controlled by the CNC machine with the PTA controller turned off.

With the diameter changed, the students recalculate and modify the spindle speed to be 778(rpm) for conducting the finish cut. The finish cut has a depth of cut of 0.1(mm) and is also controlled by the CNC machine but with the PTA controller turned on and preset at the zero reference position. After the finish cut is complete, the z-axis position of the CNC is left unchanged, and from the computer control interface, the students retract the PTA tool head 5 micrometers from the surface of the work piece. New radius measurements are then taken and recorded by the students with the help from a snap gauge.

Final precision cuts are conducted at the end with the desired depth of cut controlled by PTA, and both the feedback signals from the actuator and the finished work piece dimensions are carefully recorded.

Figure 4.9 below shows the actual setup of implementing the PTA with a CNC machine during real time machining.

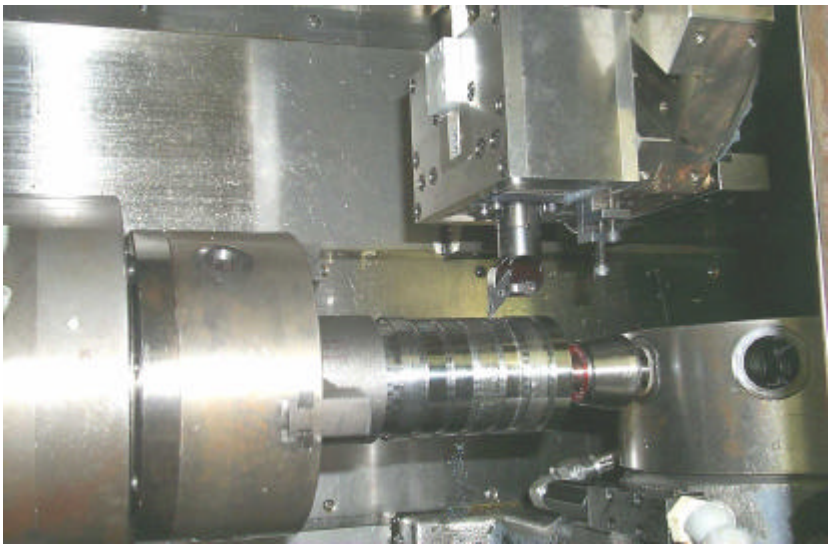

Figure 4.9: PTA Machining

Cutting Test Results

For the 1 (um) intended depth of cuts, it is observed that the tool insert is just rubbing on the surface and does not remove any material off from the work piece. In addition, the snap gauge has a resolution limited to 1(um); therefore, it is very difficult to measure the dimension change for a circular surface with a very short cutting path.

For the 10 (um) intended depth of cuts, the measured average depth of cut is found to be 10.83 (um) which is very close to the intended DOC. The steady state error can also be found by plotting the feedback position in micrometers versus time in second. As shown in Figure 4.10 on page 39, the steady state error, Ess, during cutting for the 10 (um) DOC is about 20 nanometers.

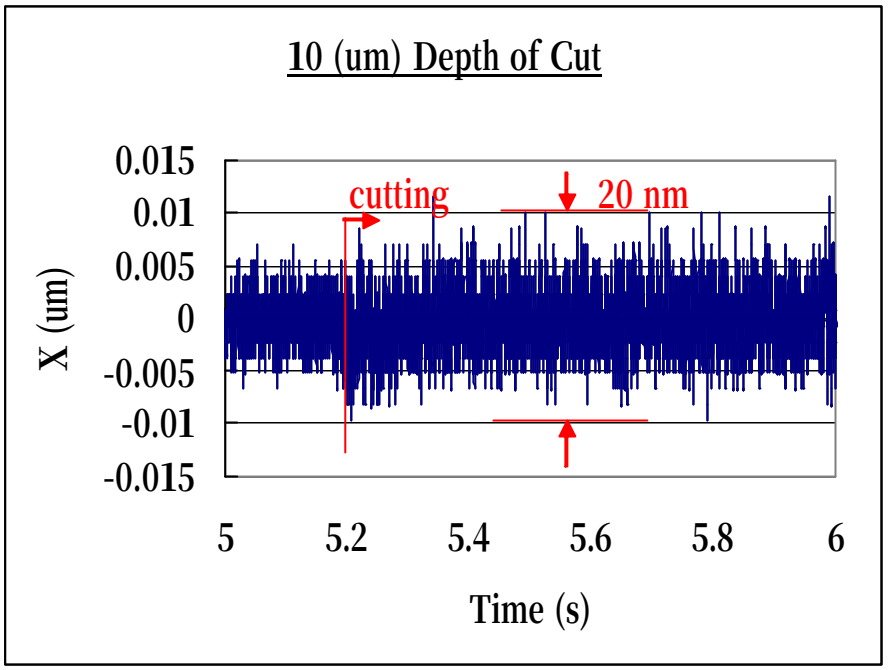

Figure 4.10: Steady State Error for 10(um) Depth of Cut

For the 18 (um) intended depth of cut, the measured average depth of cut is found to be 21.5 (um) which is also quite close to the intended depth. As shown in Figure 4.11 below, the steady state error, Ess, during cutting for the 18 (um) DOC is about 24 nanometers.

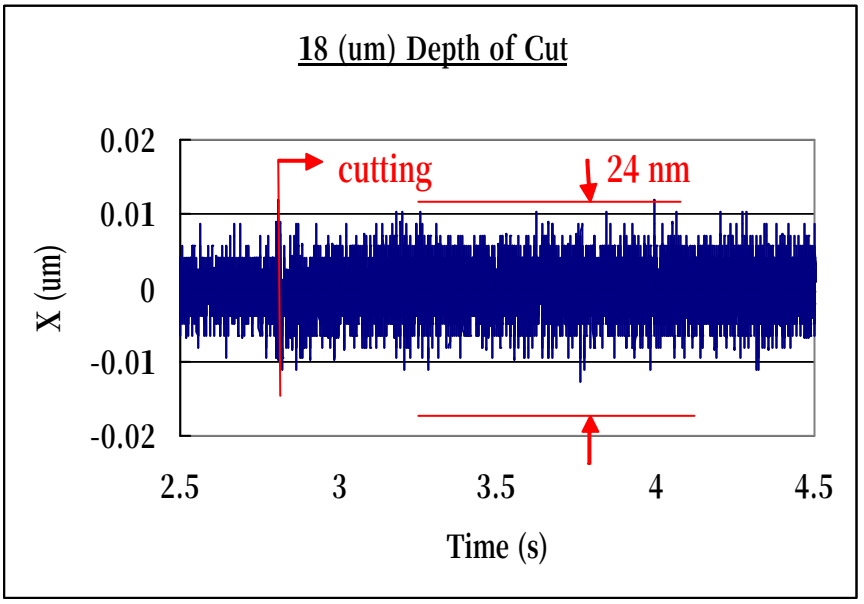

Figure 4.11: Steady State Error for 18(um) Depth of Cut

For the 20 (um) intended depth of cut, the output commending signals exceeds the output voltage limits of 1000 (volts); therefore, the computer algorithm automatically stop the commending process to reduce the voltage output in order to protect the actuator from being damaged. As a result, the dynamic response of the system is not precisely controlled, and intended depth of cut would not be achieved due to feedback forces and system non-linearity during cutting.

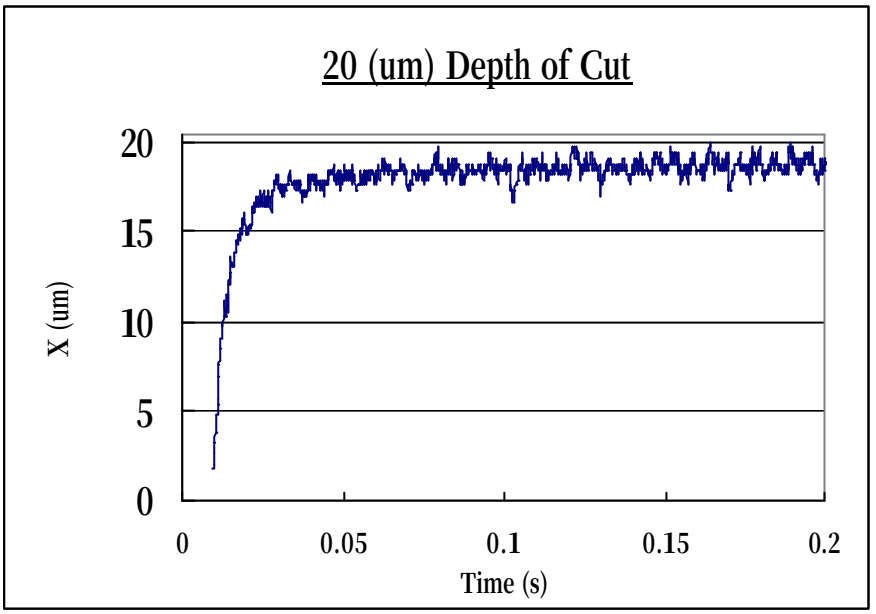

Figure 4.12: 20 (um) Depth of Cut

As shown in Figure 4.12 above, the actual position of the tool insert can never reach the desired depth of cut of 20 (um) due to computer cutting off the commending voltages. A creep behavior of the response is also observed due to system non-linearity.

5.0 Conclusion

As of April 2001, the development the PTA Controller draws to an end. The new interface now consists of a Visual Basic dialog-based form and a controller program (a modified version of the FCExec with data reading capabilities). It gives the user the ability to change parameters for the controller without any recompilation. Other additional features such as graphing and save/load functions are also added to the interface to further aid the user.

Several problems were met during the course of development. One was the incompatibility between the TI compiler and data reading functions; it was resolved by putting the I/O functions on the PC side and passing values to the DSP. Another obstacle was the long delay while plotting the output, which was a result of the inherent long execution time of Visual Basic. This could be resolved by cutting out data points plotted, but this is can become problematic since information can be lost by using this method. An alternative solution is to use graphing programs such as MATLAB, although the involvement of third-party applications is not ideal.

A number of tests were done to investigate into the interaction between the interface and the PTA, and into the PTA itself. Open loop and close loop analyses were performed with the PTA Controller, which yielded compatible results with the original FCExec. By comparing the two interfaces while working with the PTA, it was found that the tests done with the PTA Controller saved approximately one-quarter of the time necessary while using the original FCExec, along with other improvements made for the new interface (see report for details).

Other tests performed included a frequency response test and actual cutting tests. The frequency response test was done to investigate into the relationship between input power, frequency, and actuator capacitance. The experimental results were very close to those of the theoretical case; the differences were likely due to the inevitable errors existent in the apparatus used. Finally, satisfactory results were also obtained from the cutting tests done with the PTA Controller – the work pieces machine were within acceptable ranges of error.

One improvement can be made for the PTA Controller. Currently, FCExec reads data from two separate files: *fcexec.ini* and *piezoprcs.dat*. It is possible to put the two of them together to reduce one file open/close process. The main reason it has not been done is because each value in *fcexec.ini* has a label before it; it can be troublesome for Visual Basic to read value from it. However, if given more time, the parameters in *piezoprcs.dat* can be combined with those the *ini* file, each with its own label to clarify for the user. This minor improvement will be done shortly after the submission of this report.

Appendix B

Open loop transfer function calculation,

From measurement, Xss = 7.9E-7 [m]

Sensor property: 5E-6 [m/volt]

Take 4 complete cycles from the system response curve,

we get, 4Td = (27.62-26.36) = 1.26E-3[s]

therefore, $wd = \frac{1}{Td} = \frac{4}{1.26E-3} = 3.2[KHz]$

We can find parameter K by having input voltage divided by the steady state error.

$$K = \frac{U}{Xss} = \frac{(0.306-0.009)100}{7.9E-7} = 3.76\times10^{7}[volt/m]$$

For finding the damping ratio, we take 3 complete cycles from the curve,

X1 = (0.852-0.840)5E-6 = 6E-8[m]

X3 = (0.846-0.840)5E-6 = 3E-8[m]

Therefore, $x = \frac{\frac{1}{3}(\ln\frac{(6E-8)}{(3E-8)})}{2p} = 0.037$

With the damping ratio found, we can find the undamped nature frequency, Wn as follow,

$$Wn = \frac{Wd}{\sqrt{1-x^{2}}} = \frac{2p\,(3.175E3)}{\sqrt{1-(0.037)^{2}}} = 1.996E4[rad/s]$$

therefore, by comparing the system characteristic equation with the standard second characteristic equation, we find

$$M = \frac{K}{Wn^{2}} = \frac{3.76E7}{(1.996E4)^{2}} = 0.09435[volt/ms^{2}]$$

, and

$$C = 2xWnM = 2(0.037)(1.996E4)(0.09435) = 139.36[volt/ms]$$