Worcester Polytechnic Institute
Digital WPI

Major Qualifying Projects (All Years) | Major Qualifying Projects

March 2012

Active Control of Feedrate and Spindle Speed to Extend Tool Life During CNC Milling Processes

Brenden T. Gibbons
Worcester Polytechnic Institute

Joseph Adam Driscoll
Worcester Polytechnic Institute

Follow this and additional works at: https://digitalcommons.wpi.edu/mqp-all

Repository Citation
Gibbons, B. T., & Driscoll, J. A. (2012). *Active Control of Feedrate and Spindle Speed to Extend Tool Life During CNC Milling Processes*. Retrieved from https://digitalcommons.wpi.edu/mqp-all/3551

This Unrestricted is brought to you for free and open access by the Major Qualifying Projects at Digital WPI. It has been accepted for inclusion in Major Qualifying Projects (All Years) by an authorized administrator of Digital WPI. For more information, please contact digitalwpi@wpi.edu.

Active Control of Feedrate and Spindle Speed to Extend Tool Life During CNC Milling Processes

A Major Qualifying Project

Submitted to the Faculty of

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Bachelor of Science by:

Joseph Driscoll (RBE)

Brenden Gibbons (RBE)

In Collaboration with Huazhong University of Science and Technology

Partners: Yuan Liu and Yuling Niu

Date: 14 December 2011

Approved: _____

Professor Yiming (Kevin) Rong, Major Advisor, ME

Abstract

This project involved design and partial implementation of an active tool wear monitoring and prevention platform. In order to assess the sponsor company's needs, a requirements analysis was performed. After researching and analyzing the current state of the art a program was written in C and C++ to actively monitor axial cutting forces and control spindle speed and feedrate of a CNC to maximize tool life through various control methods. In order to complete the system, a comprehensive, user-friendly GUI was created.

Acknowledgements

We would first like to thank HUST for granting us the use of their facilities and their hospitality. We would also like to thank Professor Rong, Professor He, Professor Li, Dr. Hongqi, and Zhigang Yang for providing support in both setting up and executing the project. Additionally, our thanks go out to Huazhong Numerical Control, our host company. Finally, we would like to thank our partners Neil and Hermione for helping with the project and assisting us with everyday life in China.

Table of Contents

Table of Figures

Table of Tables

1. Introduction

The CNC milling process involves removing material from a workpiece by feeding it past a rotating cutting tool. The process inevitably leads to tool abrasion and therefore increased cutting forces which will eventually cause tool breakage. This tool breakage can result in a ruined workpiece as well as decreases in efficiency. As such an effective tool wear monitor and CNC controller is extremely useful in milling. Figure 1 below displays the rotating shaft of a large generator, of which the center portion is cut using a mill. Milling this piece was the original scope of this project as the ramifications of a broken tool and therefore a broken shaft are much greater than in most other cases.

Figure 1 - Large Generator Shaft

A tool wear and breakage monitor is already in use by the customers of Huazhong Numerical Control Co., Ltd, that machine the piece above, however this has become obsolete. Running on DOS and implementing outdated signal processing methods, this system needs to be replaced. To create a new and effective control system, a list of objectives was created and they are as follows: detect machine tool abrasion and breakage through the analysis of cutting forces, extend tool life by actively controlling feedrate and spindle speed with a response time of less than 100ms, continuously display plots of the spindle speed, feedrate, and cutting force, develop and efficient and stable signal processing method and a user friendly interface, back the tool off of the workpiece within 10ms of extreme tool abrasion or tool breakage, and finally, display an alarm if significant tool wear or tool breakage has occurred.

2. Background

The Background Chapter will convey general background information concerning the host company, its competitors, and iterations of the specific product that this project focuses on. It will also provide a brief literature review of similar past works, and related topics.

2.1 Company Background

Huazhong Numerical Control Co., Ltd. (HNC) was founded in 1994 in Wuhan, China as a researcher, developer, and producer of Computer Numerical Control (CNC) systems. Since its inception, HNC has become one of the leading CNC manufacturers in the world, offering innovative and reliable systems to its customers. HNC's main products are their CNC controllers; however they also offer servo drivers and motors, CNC machines, and infrared products. HNC products are used worldwide in a variety of different businesses such as the textile, mechanical, chemical, medical, and steel industries. HNC also uses their products for educational services, such as technical personnel training.

2.2 Original System

The currently implemented system, which can be seen below in Figure 2**,** is a black box with a screen on the top half and buttons to control the program on the bottom half. In the bottom half of the picture is a screenshot of the system while on but not currently monitoring anything. As can be seen, there is only one area to display information about the cutting force, and a list of options on the right side. This setup is not very user friendly and also is not capable of displaying all of the desired information.

Figure 2 - Top: Original Control System
Bottom: User Interface

On top of a poor user interface, the software of the system is outdated. The system runs on DOS which does not allow for real time operation. Also, the signal processing algorithms being used are slow and do not allow for optimal performance. Unfortunately there was no specific information was available about the controller to allow for a more thorough analysis.

2.3 Competitors

In this section, two competitive products are discussed.

2.3.1 TMAC by Caron Engineering

Founded in 1986, Caron Engineering, Inc. offers manufacturing and engineering companies a single source for the custom development, design, and assembly of tool monitoring and adaptive control systems. Their product, the Tool Monitoring Adaptive Control (TMAC) system, monitors machining horsepower in order to extend cutting tool life. It uses slope monitoring on the horsepower to regulate the machines feedrate, thus maintaining a constant spindle motor horsepower.

The TMACs features include: a response time of less than 10ms, several different machine interfaces, preset tool limits for extreme wear, and a GUI with real-time graphing with the ability to start and stop monitoring.

2.3.2 ACM by OMATIVE Systems

OMATIVE Systems is a developer of advanced manufacturing and productivity technologies, focusing on adaptive control, monitoring, and metal cutting optimization products. They are recognized as an industry and global leader in these technology areas.
The Adaptive Control Monitor (ACM) monitors cutting conditions in real-time and automatically adjusts the feedrate to its optimal level. It uses either maximum load or load band monitoring to optimize tool life and is capable of alarming and stopping the machine in case of tool breakage.

2.4 The Computer Numerical Control Milling Process

The milling process is a procedure by which a workpiece is fed past a rotating cutting tool in order to cut the workpiece. In this process, the cutting tool is held horizontally stationary while translating vertically to increase or decrease the depth of cut. The rate at which the cutting tool is spinning is called spindle speed. Increasing the spindle speed can reduce the cutting force

and increase the quality of the cut. The workpiece is held in place on a horizontally translating table and is fed past the cutting tool. The rate at which the workpiece moves across the cutting tool is called the feedrate. Feedrate has the opposite effect of spindle speed in that decreasing it will reduce the cutting force and increase the quality of the cut. A picture of a mill displaying the spindle and table can be seen below in Figure 3.

Figure 3 - Milling Machine: Table and Spindle

In many cases, this process is controlled by a CNC, or computer numerical controller. This is a computer which controls all aspects of the cutting process and allows accuracy and speed beyond that of manual control. A picture of the HNC 210iB (the project specific CNC) can be seen below in Figure 4.

Figure 4 - Milling Machine: Computer Numerical Controller

2.5 Past Research

Monitoring of tool wear and breakage on milling machines, often referred to as adaptive control in industry, has been a topic of research for quite some time and several different methods of control have been developed over the years. There are three main methods of adaptive control which are: adaptive control with optimization (ACO), adaptive control with constraints (ACC), and geometric adaptive control (GAC) [1]. ACO optimizes systems by analyzing a performance index subject to process constraints. ACC systems don't rely on a performance index, focusing solely on the constraints. GACs maintain part quality through analyzing and compensating for tool wear and deflection. Due to the project requirements and the provided hardware, an ACC system was the chosen implementation.

As mentioned, adaptive control with constraints maintains a given variable, such as cutting force, within provided constraints by altering cutting parameters, such as feedrate and spindle speed. The cutting parameters are manipulated through the use of some control algorithm that has been implemented such as a PID (proportional, integral, derivative) controller or a fuzzy logic controller. "Adaptive Control Systems for Machining" describes a proportional controller in which feedrate is manipulated to maintain cutting force [1]. In the article "Fuzzy Control of Spindle Power in End Milling Processes", a method of control using fuzzy logic to control both spindle speed and feedrate to regulate power is brought forth with reported success is various different scenarios [2]. "Design of a Drilling Torque Controller for a Machining Center" describes a system in which a PID controller was used to control torque while drilling by manipulating the feedrate of the machine [3].

2.6 Control Algorithms Overview

Sections 2.6.1 and 2.6.2 provide brief overviews of the two control methods used in the project.

2.6.1 Proportional Integral Derivative Control

A PID controller is a basic control algorithm that calculates error by taking the difference between the current state and desired state. A basic control block diagram can be seen below in Figure 5. To calculate the proportional term, the error is multiplied by some gain K_P. The integral value is calculated by summing the error with all previous errors and multiplying the new value by some gain K_I. To determine the derivative value, the change in error over time is calculated by subtracting the previous error from the current error and dividing by the change in time since

the two measurements. This value is then multiplied by some gain K_D. All gain values are determined through experimentation. The three terms are then summed together resulting in an output value corresponding to necessary change. The process then loops until the object being controlled has reached the desired state.

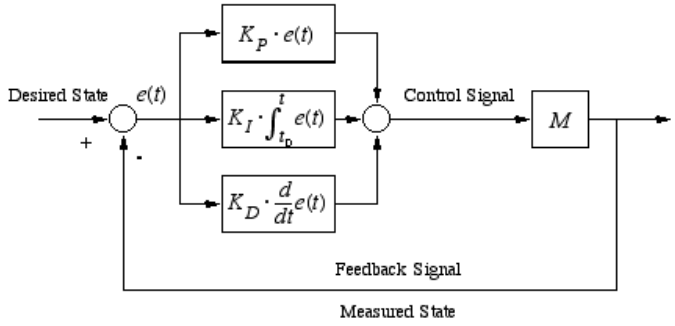


Figure 5 - PID Control Loop Diagram

2.6.2 Fuzzy Logic Control

Fuzzy logic controllers are much more complex and involved than a PID controller. Though the basic forms of the control block diagram, seen below in Figure 6 and above in Figure 5, for each are similar, there are many more steps involved in getting an output value for a fuzzy logic controller.

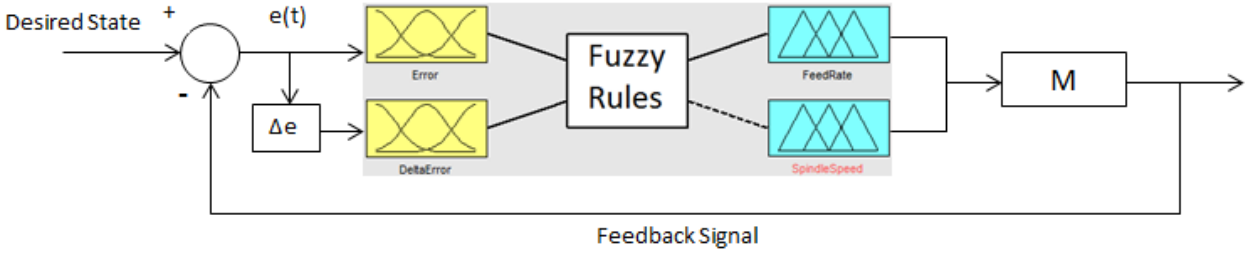


Figure 6 - Fuzzy Logic Control Loop Diagram

A fuzzy controller is made up of some number of fuzzy sets, membership functions to make those sets, a set of several rules, and truth values on a scale of zero to one. An example of a fuzzy set can be seen below in Figure 7. First the error is calculated and is input into a fuzzy set. This results in a membership function and a truth value based on where the input lies in the membership function. For example, a common membership function is in the shape of an isosceles triangle. Let's say a particular membership function ranges from 0 to 100. These means if the input is 50, the truth value will be one. If the input is 25 or 75, the truth value would be .5. If the input does not fall within this range, it will be applied to a different membership function.

Once this is done, a separate input is applied to another fuzzy set in the same manner. The membership functions currently in use by each are then applied to the rule set to determine the output. A basic rule outline is if A set is B function and C set is D function then output set is E function. Determining the truth value to use for the output function depends on how the rule is written. There are two main types of rules, AND rules and OR rules. An AND rule, used in the outline above, compares the two different truth values and uses the lower of the two. An OR rule will use the higher of the two values. It is likely that given just two inputs there will be many more than just one output as membership functions generally overlap, resulting in multiple rules being met. These outputs are all applied to the same output set. Once all rules have been checked and applied, the center of gravity, or centroid, is found based on weighing all of the individual outputs. The position of the centroid along the x-axis is the final output of the system.

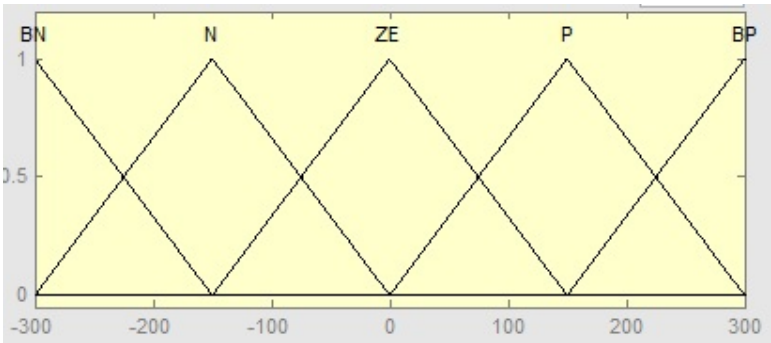


Figure 7 - Sample Fuzzy Set

3. Methodology

The following section describes how the project was approached, specifically what techniques and practices were used. It shows how the project was taken from a concept to an actual product.

3.1 Control Algorithms Methodology

For the adaptive controller, two control algorithms were chosen to be implemented; a PID (proportional, derivative, integral) controller, and a fuzzy logic controller. The PID was chosen as it is a proven and easily implementable method. The fuzzy logic controller was chosen as it provides rapid error correction and tends to be a bit more forgiving than a PID controller. Each algorithm will be discussed further in their respective sections below.

3.1.2 Proportional Integral Derivative Methodology

Because the controller has to alter two different values, spindle speed and feedrate, the PID controller needs two different sets of gains, one set of gains for the feedrate and the other for spindle speed. This allows for more accuracy and higher efficiency from the control function. This is because the two parameters affect the cutting force in slightly different ways.
Altering two different variables also allows for the use of two different versions of a PID. One version that controls the spindle speed and feedrate simultaneously, a synchronous controller, and one that controls the two values separately, an asynchronous controller. The asynchronous controller manipulates spindle speed for larger errors. Once the error crosses a predetermined threshold, control switches to varying the feedrate for the final changes. Both versions were implemented in order to give the user a bit more choice and control for cutting operations depending on the desired response. The two versions of the PID can be viewed in MATLAB code in Appendix B. It should be noted that this form alters slightly from the version in the actual program as this was also used for testing purposes which will be discussed shortly.

3.1.3 Fuzzy Logic Methodology

As discussed in the fuzzy logic background (section 2.6.2), fuzzy controllers have been implemented to control cutting force through spindle speed and feedrate changes successfully. These articles were used as a guide for the implementation of this controller.

The first step was to determine the overall layout of the fuzzy controller, that is to say, the inputs and the outputs. Going along with the method described in "Fuzzy Control of Spindle Power in End Milling Processes" it was decided that the system should take in the current error and the change in change in force and output a new feedrate and spindle speed, as opposed to a change in feedrate and spindle as done in the PID controllers [2]. The next step was to create the two fuzzy sets to take in the error and change in error. The authors used 7 membership functions per fuzzy set [2]. Keeping the user in mind, it was decided to reduce this number five to make it a bit simpler without losing too much functionality. The membership functions were labeled as "Big Negative", "Negative", "Zero Error", "Positive", and "Big Positive" and shaped as triangles, a standard membership function shape. After this, the output fuzzy sets were created, again using five triangle membership functions and the same naming convention with the "spindle" or "feed" as prefixes for the spindle speed and feedrate outputs respectively.

Once all of the fuzzy sets were created a rule set needed to be created. There needs to be a rule for each possible combination of input fuzzy sets that alters the feedrate and spindle speed in some way, resulting in 25 different rules. Again, "Fuzzy Control of Spindle Power in End Milling Processes" was used as a starting place to create the rule set [2]. However, because they used seven membership functions per fuzzy set their rule sets didn't directly translate and adjustments had to be made. Figures 8 and 9 below display the rules used for the controller.

1. If (Error is BN) and (Change_Error is cBN) then (Spindle_Speed is sBN) (1)
2. If (Error is BN) and (Change_Error is cN) then (Spindle_Speed is Z) (1)
3. If (Error is BN) and (Change_Error is cZ) then (Spindle_Speed is sP) (1)
4. If (Error is BN) and (Change_Error is cP) then (Spindle_Speed is sP) (1)
5. If (Error is BN) and (Change_Error is cBP) then (Spindle_Speed is sP) (1)
6. If (Error is N) and (Change_Error is cBN) then (Spindle_Speed is sN) (1)
7. If (Error is N) and (Change_Error is cN) then (Spindle_Speed is sN) (1)
8. If (Error is N) and (Change_Error is cZ) then (Spindle_Speed is sP) (1)
9. If (Error is N) and (Change_Error is cP) then (Spindle_Speed is sP) (1)
10. If (Error is N) and (Change_Error is cBP) then (Spindle_Speed is sP) (1)
11. If (Error is Z) and (Change_Error is cBN) then (Spindle_Speed is sN) (1)
12. If (Error is Z) and (Change_Error is cN) then (Spindle_Speed is sN) (1)
13. If (Error is Z) and (Change_Error is cZ) then (Spindle_Speed is Z) (1)
14. If (Error is Z) and (Change_Error is cP) then (Spindle_Speed is sP) (1)
15. If (Error is Z) and (Change_Error is cBP) then (Spindle_Speed is sP) (1)
16. If (Error is P) and (Change_Error is cBN) then (Spindle_Speed is sN) (1)
17. If (Error is P) and (Change_Error is cN) then (Spindle_Speed is sN) (1)
18. If (Error is P) and (Change_Error is cZ) then (Spindle_Speed is sN) (1)
19. If (Error is P) and (Change_Error is cP) then (Spindle_Speed is Z) (1)
20. If (Error is P) and (Change_Error is cBP) then (Spindle_Speed is sP) (1)
21. If (Error is BP) and (Change_Error is cBN) then (Spindle_Speed is sN) (1)
22. If (Error is BP) and (Change_Error is cN) then (Spindle_Speed is sN) (1)
23. If (Error is BP) and (Change_Error is cZ) then (Spindle_Speed is sN) (1)
24. If (Error is BP) and (Change_Error is cP) then (Spindle_Speed is Z) (1)
25. If (Error is BP) and (Change_Error is cBP) then (Spindle_Speed is sP) (1)

Figure 8 - Fuzzy Logic Spindle Speed Rules

1. If (Force is BN) and (Delta_Force is cBN) then (Feed_Rate is fBN) (1)
2. If (Force is BN) and (Delta_Force is cN) then (Feed_Rate is fN) (1)
3. If (Force is BN) and (Delta_Force is cZ) then (Feed_Rate is fP) (1)
4. If (Force is BN) and (Delta_Force is cP) then (Feed_Rate is fP) (1)
5. If (Force is BN) and (Delta_Force is cBP) then (Feed_Rate is fBP) (1)
6. If (Force is N) and (Delta_Force is cBN) then (Feed_Rate is fN) (1)
7. If (Force is N) and (Delta_Force is cN) then (Feed_Rate is fN) (1)
8. If (Force is N) and (Delta_Force is cZ) then (Feed_Rate is fP) (1)
9. If (Force is N) and (Delta_Force is cP) then (Feed_Rate is fP) (1)
10. If (Force is N) and (Delta_Force is cBP) then (Feed_Rate is fBP) (1)
11. If (Force is ZE) and (Delta_Force is cBN) then (Feed_Rate is fBN) (1)
12. If (Force is ZE) and (Delta_Force is cN) then (Feed_Rate is fN) (1)
13. If (Force is ZE) and (Delta_Force is cZ) then (Feed_Rate is fZ) (1)
14. If (Force is ZE) and (Delta_Force is cP) then (Feed_Rate is fP) (1)
15. If (Force is ZE) and (Delta_Force is cBP) then (Feed_Rate is fBP) (1)
16. If (Force is P) and (Delta_Force is cBN) then (Feed_Rate is fBN) (1)
17. If (Force is P) and (Delta_Force is cN) then (Feed_Rate is fN) (1)
18. If (Force is P) and (Delta_Force is cZ) then (Feed_Rate is fN) (1)
19. If (Force is P) and (Delta_Force is cP) then (Feed_Rate is fZ) (1)
20. If (Force is P) and (Delta_Force is cBP) then (Feed_Rate is fP) (1)
21. If (Force is BP) and (Delta_Force is cBN) then (Feed_Rate is fBN) (1)
22. If (Force is BP) and (Delta_Force is cN) then (Feed_Rate is fBN) (1)
23. If (Force is BP) and (Delta_Force is cZ) then (Feed_Rate is fN) (1)
24. If (Force is BP) and (Delta_Force is cP) then (Feed_Rate is fZ) (1)
25. If (Force is BP) and (Delta_Force is cBP) then (Feed_Rate is fP) (1)

Figure 9 - Fuzzy Logic Feedrate Rules

3.2 Testing the Control Algorithms

A control algorithm cannot be implemented without first testing it. In a real scenario, the controller would send the mill some value, the mill would apply the change and a new force would result. This new force would be sent back into the control algorithm to continue making adjustments. Because the test is a simulated scenario, there is no live update of the cutting force that happens as a natural progression. Because of this, a separate function had to be implemented to calculate what the resultant force would be based on the output and then feed that force back in. The equation used to accomplish this is as follows:

Equation 1 - Cutting Force Calculation

$$F = K_S * a * \frac{V}{pN}^{U}$$

Where F is the cutting force in Newtons, a is the depth of cut in mm, V is the feedrate in rpm, p is the number of teeth on the tool, N is the spindle speed in millimeters per minute (mm/m), K_S is the specific cutting force in Newtons, and u is some value ranging from .6 <u<1. Both K_S and u depend on the tool and workpiece material.[1]

The values chosen for testing were selected as average values based on knowledge of machining and industry standards. The values used were as follows:

Table 1 - Cutting Force Equation Historical Trends

$$K_S = 500\ N$$

$$a = 4\ mm$$

$$p = 4\ \ teeth$$

$$u = .8$$

with a desired cutting force of $F = 500\ N$. Various starting forces were chosen, both above and below the desired cutting force, to verify that the algorithms work in multiple different scenarios.

3.2.1 Proportional Integral Derivative Control Testing

Tuning the two PID controllers involved adjusting the gain values corresponding to the feedrate and spindle speed until an acceptable response was produced. This means having fast rise and settling times and a small percent overshoot. This is the same process that needs to be done for every new cutting operation. The code used to test the two controllers can be found in Appendix B.

3.2.2 Fuzzy Logic Control Testing

To test the fuzzy control, input fuzzy sets had to be defined and output sets had to be tuned. The two input sets were given the same parameters and one of the can be seen below in Figure 8.

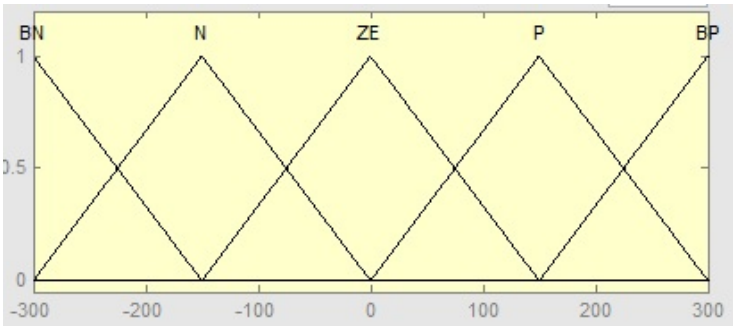


Figure 10 - Input Fuzzy Set

The displayed range is -300N to 300N, however the two membership functions on the end extend out to +-10000N, used to simulate infinity. The three inner functions were each given a range of 300N with a standard isosceles shape. This results in each function overlapping with the functions on either side of it.

Tuning the fuzzy controller involved altering the ranges and shapes of each output membership function, again until an acceptable response was produced through simulation. While it is important to be accurate with these numbers, they do not have to be as exact as the gains in the PID because the fuzzy controller is much more forgiving in this sense. Unlike the PID controllers, Simulink was used in order to test the fuzzy controller. The model used can be seen below in Figure 9.

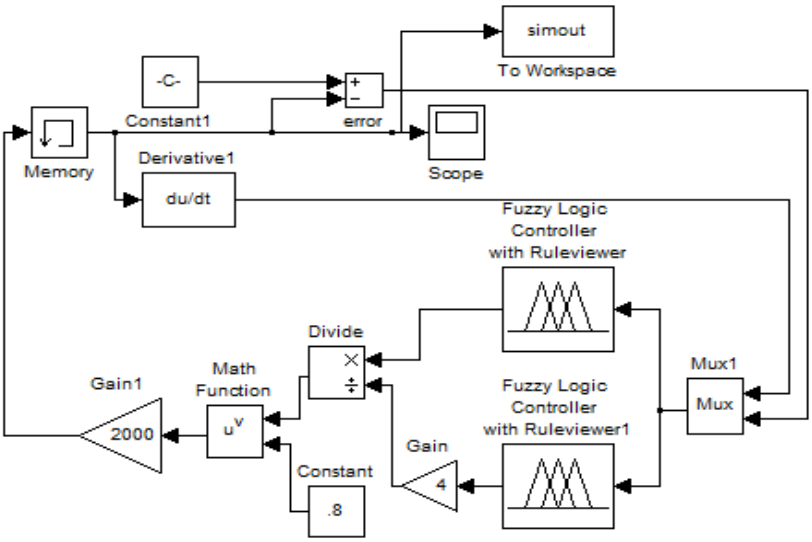


Figure 11 - Fuzzy Control Testing Loop

To understand the model, it is easiest to start at the memory block. This block is used to set the initial force for the test. This sends a signal to three places: a scope block to measure the signal, a derivative block to calculate the change in force, and the negative input of an error block. On the positive input of the error block is a constant corresponding to the desired force. This block, as its name implies, calculates the error between the desired and current force. This signal, along with the derivative signal, is sent to a Mux block to combine the data into one signal so it can be interpreted by fuzzy controller. This model makes use of two separate fuzzy controllers, one for spindle speed and one for feedrate, instead of one as described above in the fuzzy methodology section (section 3.1.3). This was due to limitations in Simulink that did not allow for multiple output sets in a single controller. The two fuzzy logic controls run and calculate their output. These outputs are then combined over the next five blocks which calculates the above cutting force equation. The simulation continues to loop around until the set run time has completed. It is also important to note that with this setup, the controller does not begin actively controlling until the second loop.

3.3 Software Methodology

Before any actual code could be written, a requirements analysis for the software was completed. This entailed listing the Customer, Architectural, Structural, Behavioral, Functional, Non-Functional, and Design requirements. All data was obtained in an interview with Dr. Hongqi Liu. Table 2 below shows this analysis. Although most of the following requirements were by the company, some additional ones were added to during the creation of the software to allow extra features, or to ensure other ones were met.

Table 2 - Requirements Analysis

Requirement	
Customer	• The system will be used in HNC's customer's factories on assorted milling machines in order to monitor tool wear and reduce tool breakage. • The system will be expected to operate effectively in a typical factory setting
Architectural	• The system must be comprised of: A usb mouse, an analog keyboard, a monitor, an Advantech PCM3718HO, an Advantech PCM3355, a Kistler 5070A charge amplifier, and a Kistler 9129AA Multicomponent Dynamometer. The NC model used is the HNC 210B.
Structural	• Expandable Code base

	• Object oriented-style coding • Data modules for easy data recording
Behavioral	• Responds to Possible Cutting Tool breakage by setting off the Mill alarm, showing a flashing red and black screen, backing the milling tool away from the workpiece, and stopping the spindle within 10ms. • Responds to "Start" and "Stop" buttons by turning on/off data collection, control, and updated display of values. • Responds to pressing restart button in alarm screen by turning off the alarm, saving all stored data, and allowing new operations to commence • Responds to clear button being pressed • Allows the creation of Individual tools and operations • Allows the creation of control systems • Responds to "Save Data" button being pressed by writing stored data to a formatted text file • "Quit button" exits the program cleanly • Two tabs for basic and advanced users
Functional	• Monitors and display 3 axis force sensor data in real time (100ms) versus time. • Determines the state of tool breakage at any given moment • Display real time (100ms) plots of the current feedrate and spindle speed versus time. • Display numbers correlating to values of plots in real time • Actively control the feedrate and spindle speed of the NC to extend tool life using a selected control algorithm.
Non-Functional	• Usable by basic Factory worker • Robust and hard to break – extensively tested • Efficient enough to run on the limited hardware selected
Performance	• 10ms hard real-time response for setting off the alarm and backing the mill of the workpiece in the event of likely tool wear • 100ms soft real-time response for collecting data, controlling the feedrate and spindle speed, and displaying collected data
Design	• Whole program done in C, except for GUI elements • GUI elements done in C++ • Use Red Hat 9 Linux as the operating system • Expandable code base

In order to best meet all of the requirements of the preceding analysis, several features of the code were planned out in detail before any coding began. It was decided that for ease of usability, compatibility, and portability that all non C++ code would be styled in an object-oriented manner. This involved the extensive use of structs, and passing pointers to them to all

functions modifying them. All code would be written in the ANSI C99 standard to ensure portability between compilers.

For planning purposes, a UML (Unified Modeling Language) diagram was created. This allowed separate functions in each class to be specifically defined before coding, and also a visualization of how incoming data travels through and is modified in each of the modules. Several iterations of the UML diagram were created, both before and during coding. The iterations were necessary when certain requirements could not easily be met with the UML diagram at the time. A simplified version of the latest and most complete iteration of the UML diagram is shown below in Figure 10. The full version can be found in Appendix A.

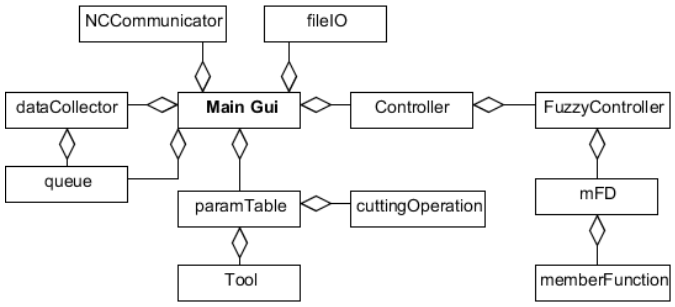


Figure 12 - Simplified UML Diagram

Many measures were taken to ensure that the code base was easily expandable. The first of these was the heavy compartmentalization of the code. Separate C modules (header and .c file) were created for almost every task in code. In all, nine individual-purpose modules were written, and a tenth for the GUI. The separation of tasks into modules also allowed ease of testing and debugging for each section of code. The second measure was to comment all code consistently and thoroughly. Each file in the code base is extensively commented with great detail in the DoxyComment format. The final measure to ensure ease of expandability / modification was to create a .pdf file outlining how all code fits together and explaining each module by a line by line basis.

3.4 Graphical User Interface Design

As shown previously in the Requirements Analysis, there were minimal specific requirements for the system's GUI. The first step taken in creating a GUI was to choose the most appropriate and usable framework. Possible frameworks included wxWidgets, Qt, GTK+, and BVRDE. Ultimately QT was chosen due to its C++ cross platform nature, signals and slots

implementation, and ease of use. Additionally, several team members had used it previously to avail.

The 2003 Qt version 3.1.0 was used to ensure that the PCM 3355 could run it in a Red Hat Linux 9 environment, as required by HNC. This version comes built-in with larger RH9 installs, so compatibility issues were reduced naturally. In addition to providing a C++ programming framework, the Qt install includes many useful tools such as Qmake for making large projects with ease, and QML for documentation of both C and C++ GUI files. Qwt widgets version 4.2.0 for Qt were used for displaying real time plots.

A process of programming a GUI, analyzing it, and iteratively improving it to better meet/exceed requirements was used as a method to achieve the best possible outcome. Figure 11 below shows the first GUI created.

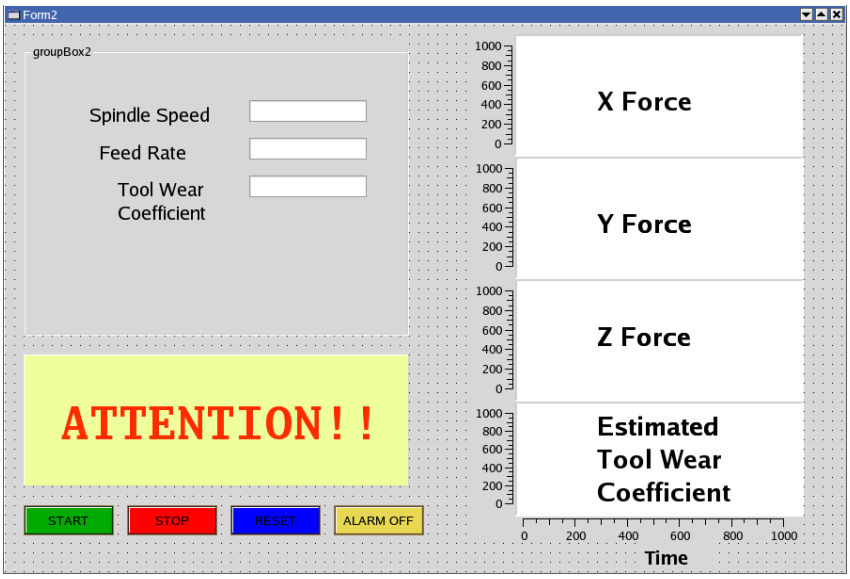


Figure 13 - Initial GUI Design

This GUI met all of HNC's desired functionalities at the time. It had start, stop, reset, and alarm off buttons, a small alarm screen that flashes in response to severe tool wear. Real-Time plots of three-axis forces are also displayed, along with a real time plot of estimated tool wear coefficient. These four plots were placed vertically atop one another, so that only one time axis was displayed for screen space efficiency. Not long after this initial GUI was created, HNC greatly changed their requirements, shifting the focus of the project to actively controlling feedrate and spindle speed to extend tool life rather than predicting tool wear. As a response to these changes, a second GUI was made. It is shown in Figure 12 below.

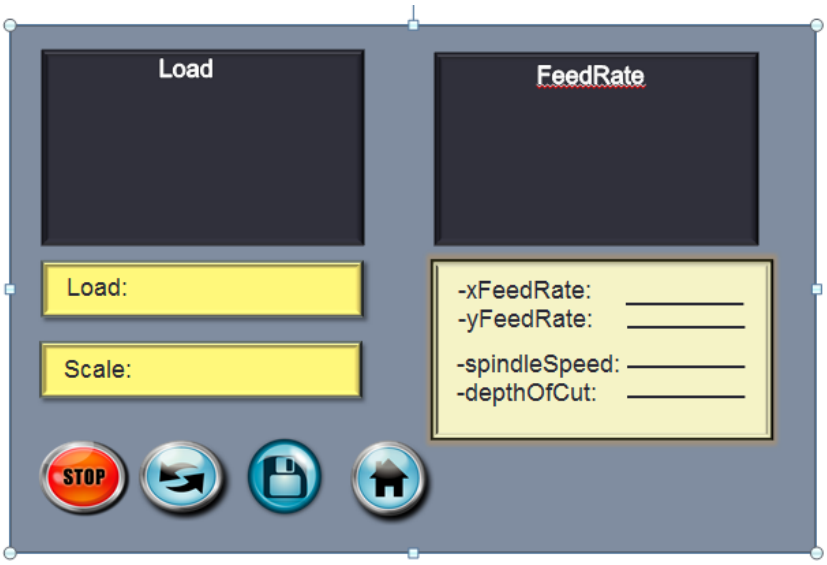


Figure 14 - Second Iteration of GUI Design

Although iteration 2 of the GUI met the new requirements of displaying the workpiece load and feedrate on real time plots, as well as showing actual numerical values of the feedrate and spindle speed, it did not display a plot of spindle speed, and had limited buttons for user input. Its color scheme and layout style also did not match that of HNC's CNCs, which was desired. Additionally, no "Quit" button was created by accident. Iteration 2 was therefore scrapped in favor of the next iteration, shown in Figure 13 below.

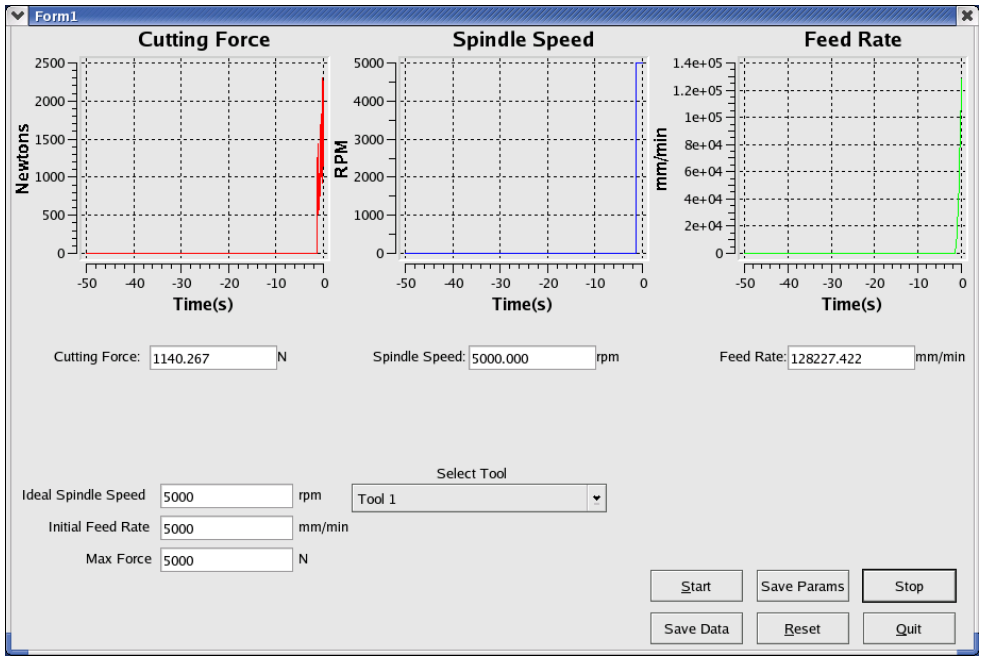


Figure 15 - Third Iteration of GUI Design

Iteration 3 of the GUI met all requirements laid out by HNC. It contained six user input buttons (Start, Stop, Reset, Save Params, Save Data, and Quit), as well as the three required real-time plots. Additionally, a drop down box for cutting tool selection was added for ease. There are manual number entry boxes for the experimentally determined maximum allowable cutting force, ideal spindle speed and initial feedrate of the cutting operation. Although all strict requirements were met, one more GUI iteration was created to ensure multi – level usability and ease of operation. The final iteration of the GUI is shown in Figure 14 below.

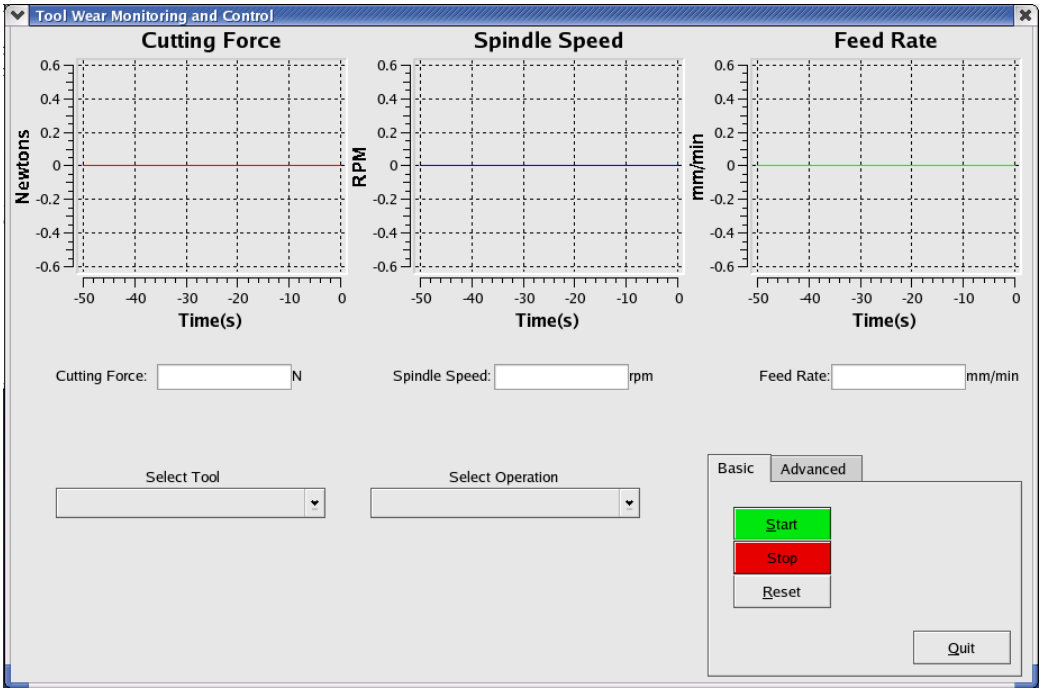


Figure 16 - Final GUI Design

The final iteration of the GUI was a significant improvement to all others in terms of functionality, multi-level usability, and style. It retained all of the required features from previous iterations (such as the six buttons and three real time plots), but also added several new features. These features include a drop down box for selecting cutting operations, button tabs for both basic and advanced users, and the ability to create separate controllers with specific parameters linked to a desired pair of operation and tool. A full description of this GUI's functionality can be found in section 4.2.1 Results of GUI.

3.5 Hardware System Design

The system hardware architecture was primarily decided by HNC before the project began, although slight modifications had to be made for various reasons. Figure 15 below shows the initial hardware provided by HNC.

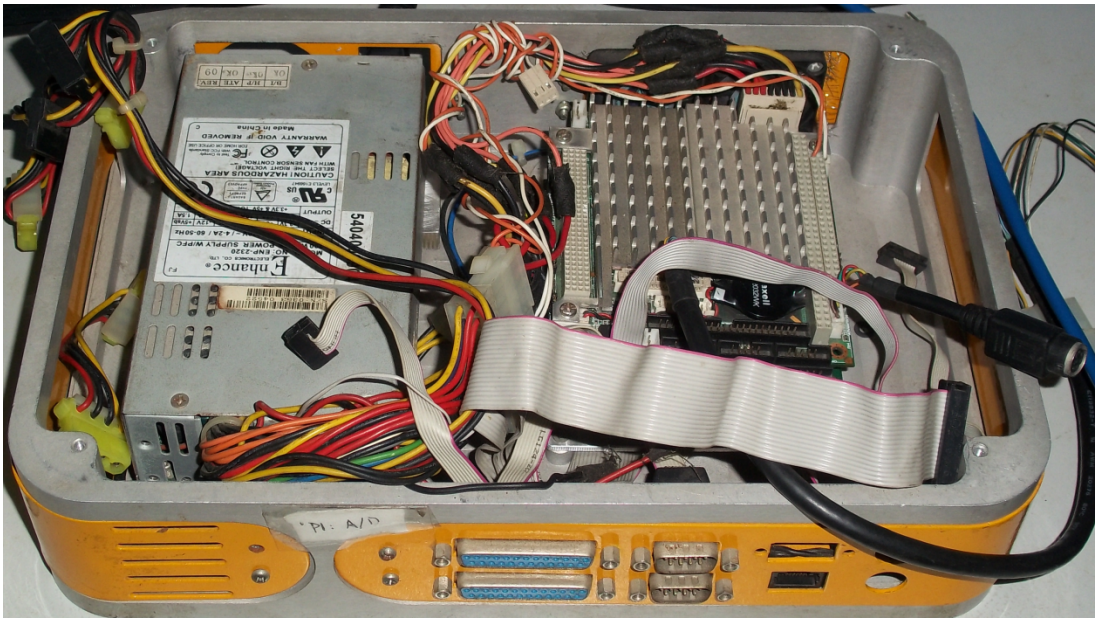


Figure 17 - Inner Components of the New Control System

It consisted of a power supply, a PCM3718HO board, a PCM 3778 board, a 5" TFT LCD display, five buttons for user input, and a small fan inside of a steel and plastic case. Upon initial inspection, many wires appeared to be broken, or disconnected in potentially harmful ways. To avoid serious problems, datasheets for all components were analyzed to determine exactly what wires needed to be replaced/ connected to what. A total of three wires needed to be fixed, as well as a plastic end cap used to connect analog keyboard input to the PCM3778. Despite limited resources, these repairs were made successfully. Once a keyboard could successfully be connected, installation of the operating system began. A problem arose when connecting the PCM3778 board to the TFT LCD display. Although all wires were undamaged, the display would only turn on about two percent of the time, and only for moments. This problem was traced to a faulty PCM 3778 board, so it was replaced with a newer model with similar capabilities, the PCM 3355. Once installed, the new board also did not work with the TFT LCD display. For diagnostic purposes, a normal LCD desktop monitor was hooked up and tested. It worked successfully, indicating that the TFT LCD itself was malfunctioning.

Although the Desktop LCD Display could interface with the new PCM 3355 board, the board did not have any firmware or BIOS installed by the factory, rendering it unusable until the proper manufacturer's CDs were obtained. Finally, one week before the end of the project, a PCM 3355 board was delivered in working condition. The final hardware system diagram is shown in Figure 16 below.

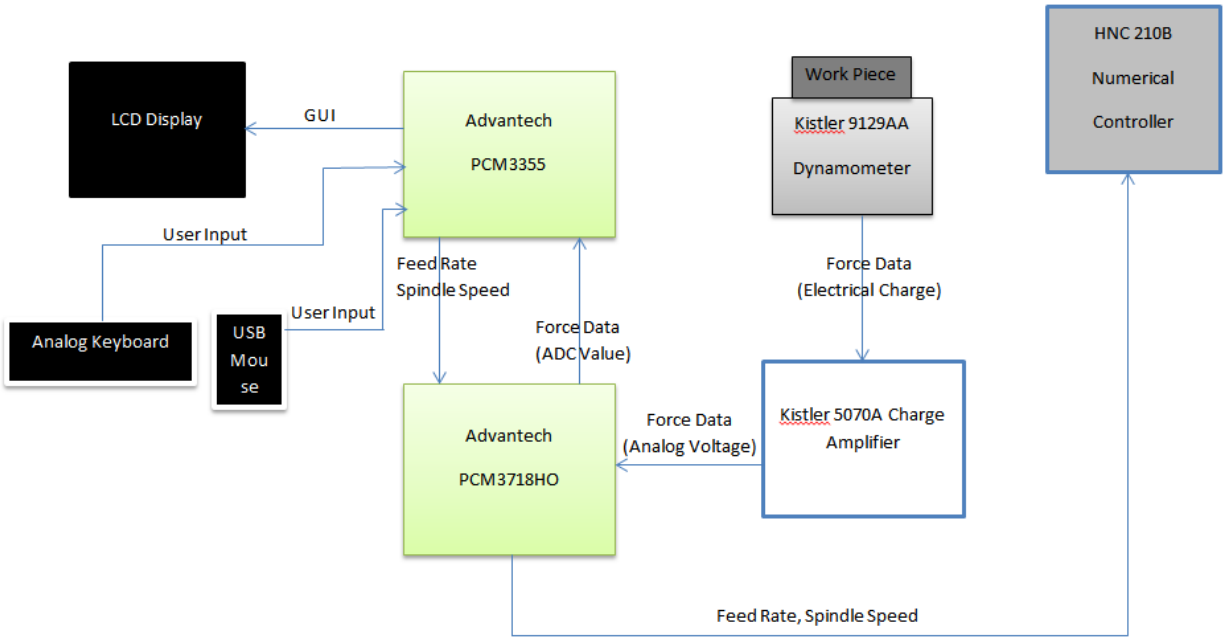


Figure 18 - Hardware System Diagram

The flow of data and in what form it is at any stage is clearly labeled. The function of each component is explained in Table 3 below.

Table 3 - Hardware Components

Advantech PCM3355	Single board PC104 computer used for main control calculations, driving the display, handling user input, and overall system coordination.
Advantech PCM3718HO	Analog I/O board used for both collecting analog workpiece force data from the Charge Amplifier and outputting digital words to the HNC 210B to change the feedrate and spindle speed.
LCD Display	Displays the GUI, allowing the user to interact with the program.
Analog keyboard and USB mouse	Input devices used to operate the system.
HNC 210B	The numerical controller model that the system is made to operate with.
Kistler 9129AA Multi-Component Dynamometer	Measures forces on the workpiece in the x, y, and z axes and outputs proportional electrical charge across three channels

Kistler 5070A Charge Amplifier	Takes in the three-channel electrical charge output from the Dynamometer and converts them to corresponding analog voltages

This final hardware system configuration would allow precise control of the HNC210B feedrate and spindle speed, sufficient user input capabilities, and a large display for the GUI.

3.6 Operating System Selection

The choice of what operating system to run was mainly determined by three important factors, the first of which is as follows: The PCM3355 board needed to be able to communicate with the PCM3718HO board through PCM104 pins in order to receive three axis real time force information. Advantech only offers the drivers with this functionality for Windows Compact Edition and Red Hat Linux 9. Therefore, it was decided that in the interest of time, and to prevent complications, it was best to use one of these two operating systems, as the drivers for data collection were already written. Therefore, the choice was immediately limited to two operating systems.

The next factor that played an important role in operating system selection was the limited power and storage of the PCM 3355 board. Table 4 below shows the specifications that the board is capable of compared to shows the board's configuration as provided by HNC:

Table 4 - PCM 3355 Specifications

Spec	Capable	Provided
CPU	AMD Geode 500MHz	AMD Geode 500MHz
RAM	1GB DDR333SRAM	512MB DDR333SRAM
BIOS	4mbit flash bios	4mbit flash bios
Compact Flash	Up to 128GB (Dependent on OS)	1GB

The specifications of the provided board presented an issue with what operating systems would be installable. With only 1 GB of compact flash, a typical install of red hat Linux 9 would be impossible, so custom options would have to be chosen. With a slightly larger (2GB) compact flash card, this would not be a problem. Windows CE has a very small memory

footprint (32MB), so it could easily be installed on the system, however the third factor involved in operating system choice made it an unviable option.

The final and most influential factor that led to the choice of Red Hat 9 for the operating system was the suggestions of Huazhong Numerical Control. The sponsor company suggested using Red Hat 9 to meet the real-time requirement of the project. The team's advisor Dr. Hongqi Liu and his graduate student Zhigang Yang both agreed with this suggestion, as they had worked with it previously. As a group of non-computer scientists, the team thought it best to listen to this advice.

3.7 Data Collection Methodology

The Advantech advDAQ libraries were used for all Data collection. The main challenge was to obtain a force value in Newtons from the charge amplifier, which outputted voltages. To implement this functionality, the following steps were taken: The raw output voltages of the charge amplifier range from -10V to 10V. The Advantech PCM3718 board's Analog to Digital Converter is twelve bits, number of states is 4096. This gives a resolution of .00488V/bit. The output from the ADC is therefore multiplied by this value to obtain the actual voltage. Once an actual voltage has been obtained, this voltage can be converted into a force value by first subtracting ten volts (to normalize the range), the multiplying by 10000Newtons/20V. A function for this purpose can be found in Appendix C, DataCollector.c.

Two methods for collecting data were implemented. The first (using a software timer to poll the charge amplifier) is an easy to implement, yet somewhat unreliable method. It involves simply using the advDAQ method to get an ADC value from a channel on a software driven interrupt. Problems can occur however when other software processes are scheduled to be executed before this timer interrupt happens, and therefore, it may not be accurate every time. The second method is using a hardware interrupt. This is slightly more difficult to implement than the previous method, however it is more precise and reliable. A hardware interrupt is set on the PCM3718 to trigger every so often. Then, when this occurs, an interrupt service routine is run to record the data. Because the timer on the PCM3718 is very reliable, and the interrupt service routine always pre-empts the currently running process, it is rare that the collected data would not match up perfectly with the time step.

All collected data for each axis is stored in a corresponding queue of a variable specified length. This data structure allows a constant flow of incoming data to be easily processed and displayed versus time.

Summary

In summary, three separate control methods were proposed, including fuzzy, synchronous PID and asynchronous PID. Tests were then designed for these algorithms. Next, a data collection scheme was created, and a suitable operating system was selected. Finally, a software architectural structure was conceived, and a comprehensive and user-friendly GUI was designed.

4. Results

The following section describes the final outcome of the project. It outlines what was completed, and how it corresponds to the initial goals laid out in the Introduction section.

4.1 Control Algorithms

Although the controller was not able to be tested in a real life scenario, all three implemented control algorithms were able to be simulated and perform quite well under the theoretical conditions after some fine tuning. In the next sections, the performance of each control algorithm along with sample results will be discussed

4.1.2 Proportional Integral Derivative Control System Development

Figure 17 below displays the response of the synchronous PID to the sample data discussed is section 3.2.

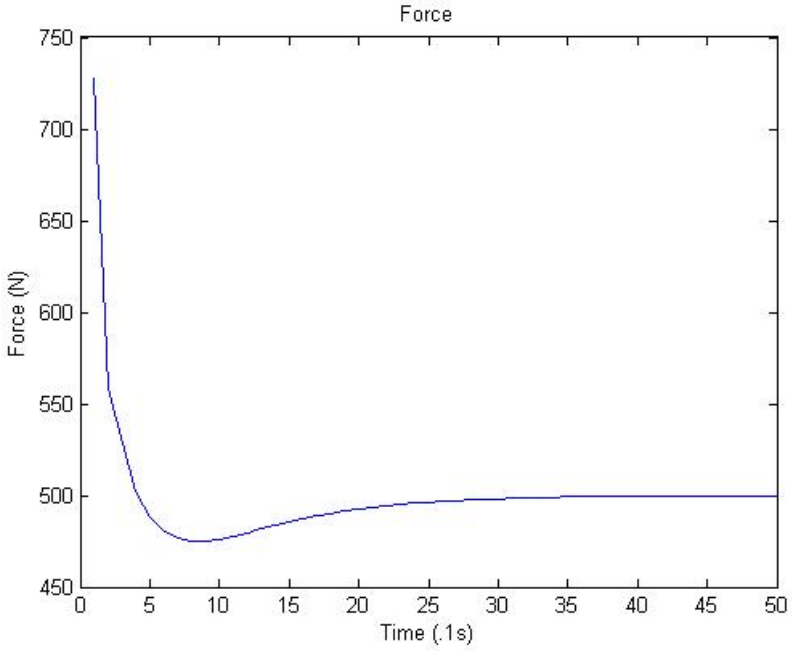


Figure 19 - Synchronous PID Control Response

After some tuning, the final gain values for the system were:

Table 5 - Synchronous PID Gain Values

Feedrate Gains:	Spindle Speed Gains:
$K_P = 1$	$K_P = 1.5$
$K_I = .11$	$K_I = .15$

$K_D = .01$ $K_D = .03$

These values result in a very rapid adjustment to the desired force while overshooting the desired force by a small amount. In more specific terms, a rise time of about .35s, a settling time of about .9s, and about an 11% overshoot.

Figure 18 below displays the tuned response of the asynchronous PID controller.

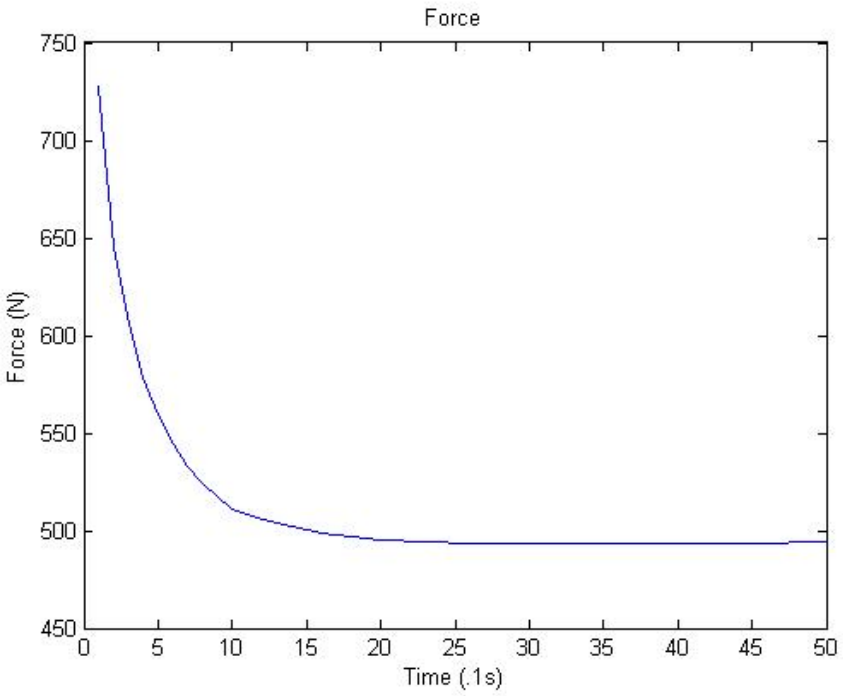


Figure 20 - Asynchronous PID Control Response

The gains used for this system were as follows:

Table 6 - Asynchronous PID Control Gain Values

Feedrate Gains:	Spindle Speed Gains:
$K_P = 1$	$K_P = 1.3$
$K_I = .01$	$K_I = .03$
$K_D = .05$	$K_D = .01$

These values, unlike the synchronous version, provide a much slower rise with a very small percent overshoot. Because the rise time of the system is so slow, the rise and settling times are about the same at roughly .8s with about a 3% overshoot. While the settling time is actually smaller than the synchronous function, the asynchronous function takes a very long time to reach the actual desired value.

4.1.3 Fuzzy Logic Control System Development

Tuning fuzzy controller resulted in the two following output sets:

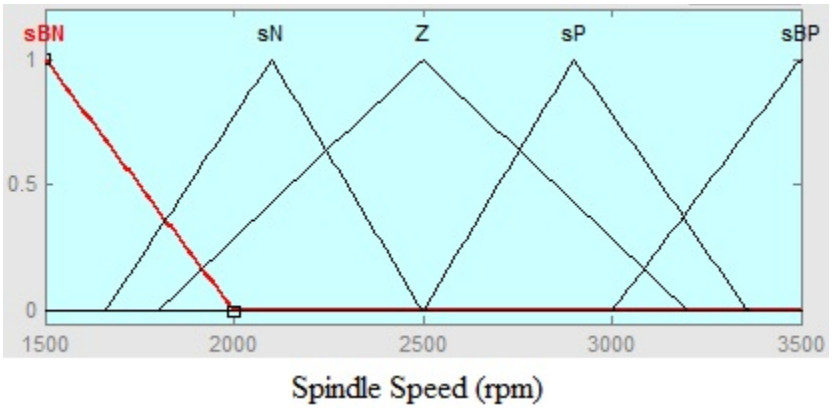


Figure 21 - Spindle Speed Output Fuzzy Set

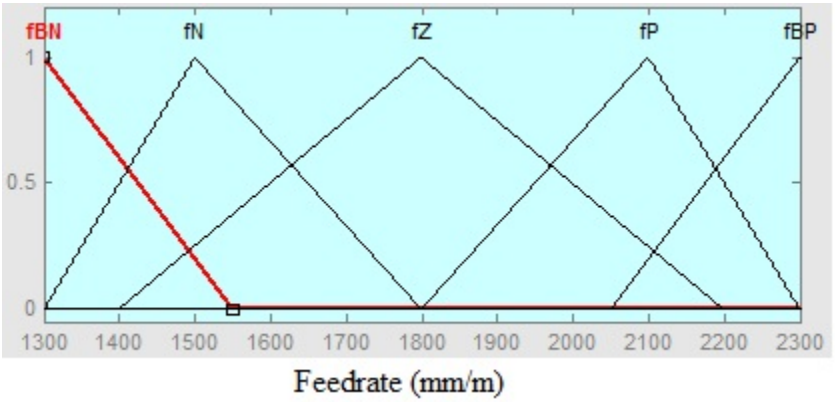


Figure 22 - Feedrate Output Fuzzy Set

Figure 19 shows that the spindle speed set ranges from 1500 rpm to 3500 rpm, centering on 2500 rpm. The spindle negative (sN) and spindle positive (sP) both have a range of 840 rpm with the peak shifted 20 rpm towards the center of the set. The two outer functions are standard have a range of 500 rpm their peaks at the borders of the sets. The zero function was given a significantly larger range than the other sets, total range being 1400 rpm with the peak right at the center of both the function and the set.

Figure 20 shows that the feedrate set ranges from 1300 mm/m to 2300 mm/m, centering on 1800mm/m. The two outer functions again have their peaks on the borders of the set, this time with a range of 250 mm/m each. The negative and positive sets were given a range of 500 mm/m with the peaks shifted 50 mm/m towards the outside of the set. Again, the zero function was

given a much larger range than the rest with a range of 800 mm/m and the peak both in the center of both the function and sets range.

The actual results of running through a simulation with the fuzzy set are displayed below in Figure 21.

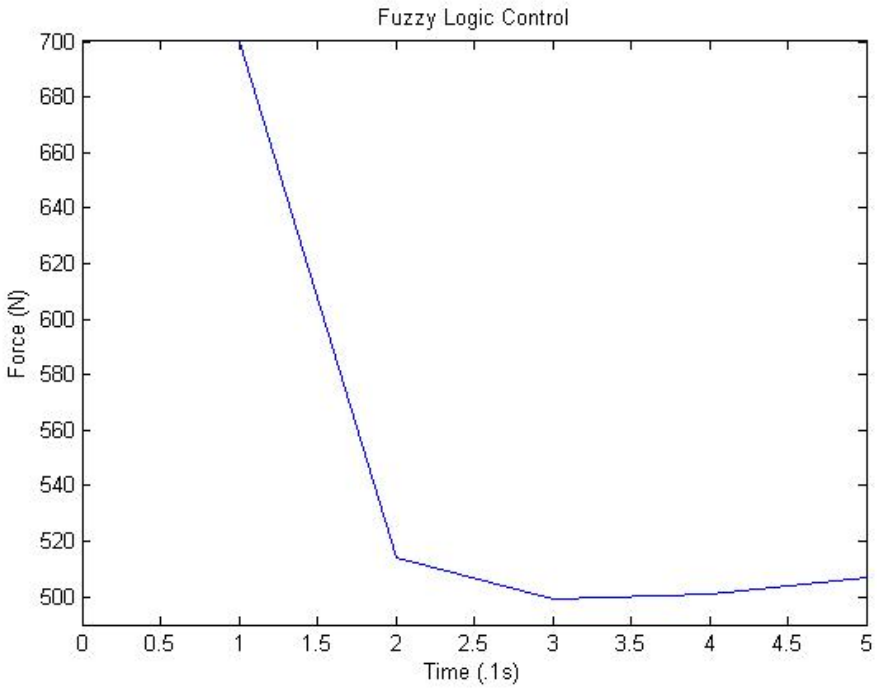


Figure 23 - Fuzzy Logic Control Response

Because of the zero values for the two output sets, 1800mm/m for feedrate and 2500 rpm for spindle speed, the desired force came out to be 507.28N instead of the 500N that was used for the PID simulation. Because Simulink runs in discrete time steps, it is difficult to determine any values such as rise and settling time. Though the fuzzy the simulation results show a correction time of .4s as the first .1s involved getting the simulation to run properly as a result of the Simulink setup.

4.2 Software System Development

This section describes what was achieved in terms of the overall software codebase, and the Graphical User Interface.

4.2.1 Final Graphical User Interface Design

Figure 22 below shows the multi – functional GUI. As the guidelines for the user interface were not outlined in depth by HNC, the team tried to create one that would encompass a range of usage and abilities from simple to complex. This would enable basic users (i.e. factory workers) to use the program simply to monitor cutting operations, while advanced users could create and save individual control systems that best fit specific cutting tool and operation combinations.

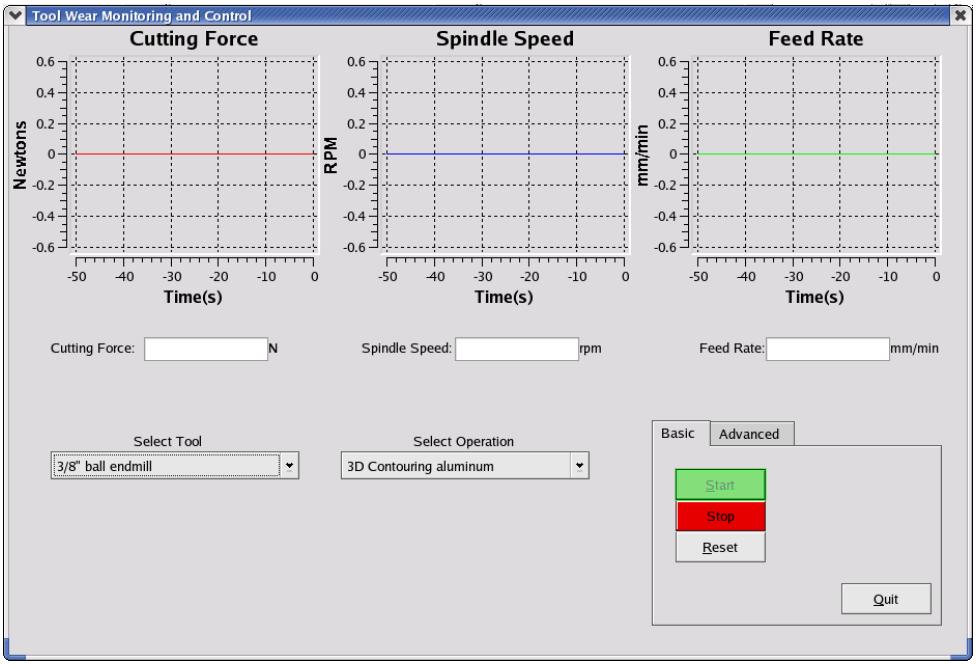


Figure 24 - Final GUI Design

The foremost specified requirement of the GUI was to display real-time plots of the cutting force, spindle speed, and feedrate. This was accomplished by expanding upon the qwtPlot widget from the QWT library. Each plot dynamically changes its y axis scaling to best fit the range of incoming data. The x-axes range from zero to a negative number of seconds that is specified in pre-processor defines of the program. The three graphs are very large and placed prominently at the top of the GUI.

Display boxes below each of the three plots show the numerical value of the current data. They are updated along with the three plots every 100 milliseconds. They cannot be clicked on

or typed in, and are intended to be useful when the data range in a plot is large enough to create confusion and misestimating upon visual inspection.

On the lower left hand side of the GUI, drop down boxes for selecting both a tool and an operation are present. Although initially empty, they can be added to by using features in the "Advanced Tab" When

To the lower right of the GUI, there are two tabs: "Basic", and "Advanced. All of the buttons required for basic usage of the GUI exist under the "Basic" tab. These buttons would be used by a typical factory worker when monitoring a cutting operation with previously defined parameters and controllers. Figure 23 below shows the basic tab.

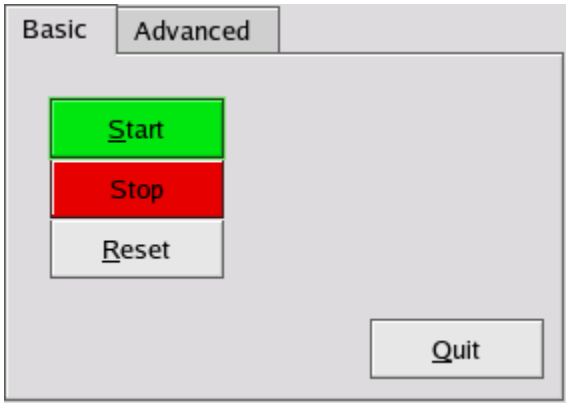


Figure 25 - GUI: Basic Tab

The green "Start" button begins cutting force monitoring, control of spindle speed and feedrate, and real-time data collection and plotting. It is dynamically enabled when a valid exists for the selected Tool and Operation and disabled when no such controller exists.

The red "Stop" button temporarily suspends cutting force monitoring, real-time control of spindle speed and feedrate, and data display. Pressing "Stop" does not clear or reset any data. The "Reset" button clears all existing data and the three plots. It is used when changing the Tool and operation combination, or when performing a new cut, as all stored errors within the current controller will be cleared.

The "Quit" button to the absolute lower right of the interface clears all data, tools, operations, and controllers and exits the program.

4.2.1.1 Advanced Tab

The "Advanced" Tab (shown below in Figure 24) houses four buttons.

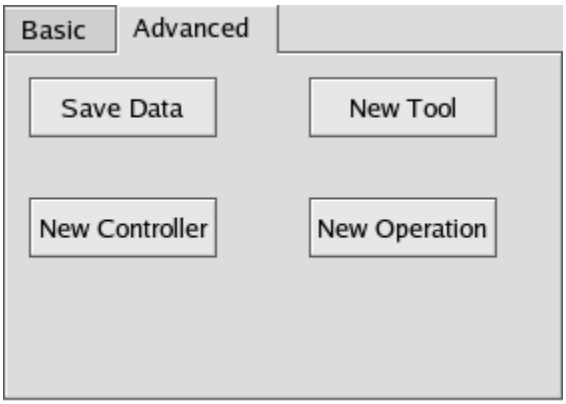


Figure 26 - GUI: Advanced Tab

The "Save Data" button writes all currently stored data to a text file for future viewing. The file is named with a timestamp for ease of filing. The first column displays the time increment. The second, third, and fourth columns show the cutting force, spindle speed, and feedrate respectively. The final three columns are the x, y and z forces as measured by the force plate. The file is stored in the same directory as the program is run from. This functionality was one of the specific details communicated by HNC.

The New Tool Dialog, shown below in Figure 25, is evoked when the "New Tool" button is pressed. This dialog allows users to create tools with specified names and maximum cutting forces. The maximum cutting force value is an experimentally determined value for each tool, over which a tool break will likely occur. The Alarm is triggered when cutting force exceeds this value. Once a new tool is created, it is added to the drop down box of tools on the left hand side of the GUI, allowing it to be selected whenever desired. There is a maximum of 25 unique tools, after which a message indicating that no more new ones may be created is shown.

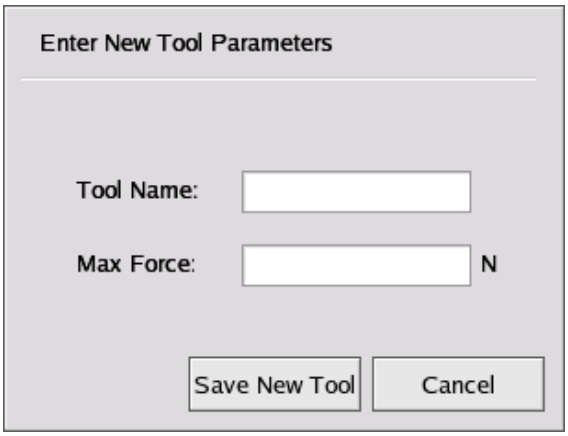


Figure 27 - GUI: New Tool Parameters Entry

"New Operation" – The New Operation Dialog, Figure 26 below, is conjured when the button is pressed. This dialog allows users to create operations with specified names. Each new operation created represents an action one would perform using a CNC machine, using a given tool. An example would be "3D contouring ANSI 6160 Aluminum", or "Grooving generator shaft". Once a new operation is created, it is added to the drop down box of operations on the left hand side of the GUI, allowing it to be selected whenever desired, just like a tool. There is a maximum of 25 unique operations, after which a message indicating that no more new ones may be created is shown.

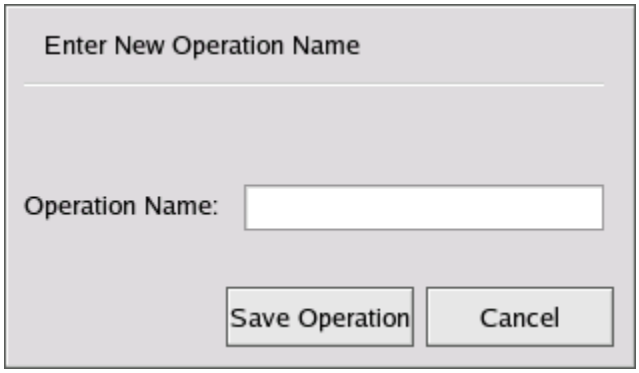


Figure 28 - GUI: New Operation Entry

The "New Controller" button can only be pressed when both a Tool and an Operation have been selected from their drop down menus. When pressed, it pops up a window for selecting the type of new controller the user wishes to create. As shown below in Figure 27, one of three controller types may be selected: Synchronous PID, Asynchronous PID, or Fuzzy.

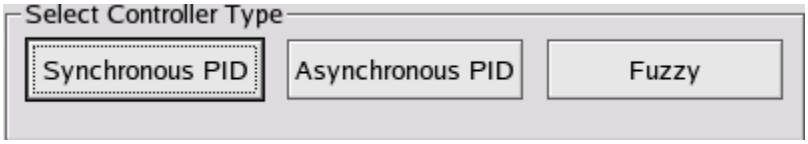


Figure 29 - GUI: Control Algorithm Selection

In each instance, the new controller is bound to the currently selected Tool/Operation combination. This means that it will be saved upon creation and automatically selected in the future whenever that Tool/Operation combination is selected from the drop down menus. This feature allows advanced users to set up complicated parameters, while allowing basic users to operate the program by remaining in the "Basic" tab.

The "Create new Fuzzy Controller" dialog pops up when the "Fuzzy" Button is pressed. Once all of the parameters for creating a new Fuzzy Controller have been entered, the user clicks

the “Create” button. A fuzzy controller with two member function diagrams of five member functions each is dynamically created from the entered values.

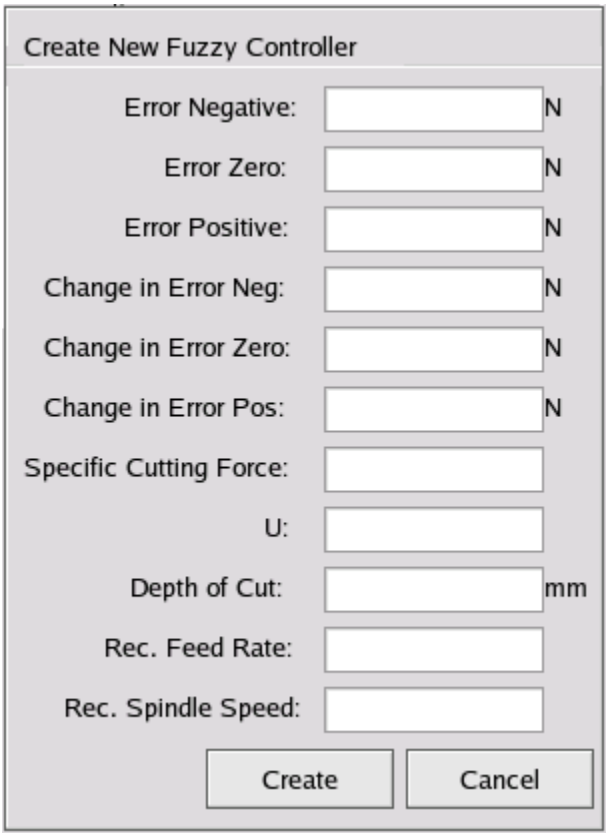


Figure 30 - GUI: Fuzzy Controller Setup

The “Asynchronous PID” button opens the “Create new Asynchronous PID Controller” dialog (Figure 29). Here, the user inputs all of the parameters necessary for creating an Asynchronous PID Controller. Once “Create” is clicked, the controller is created and bound to the specific Tool/Operation combination.

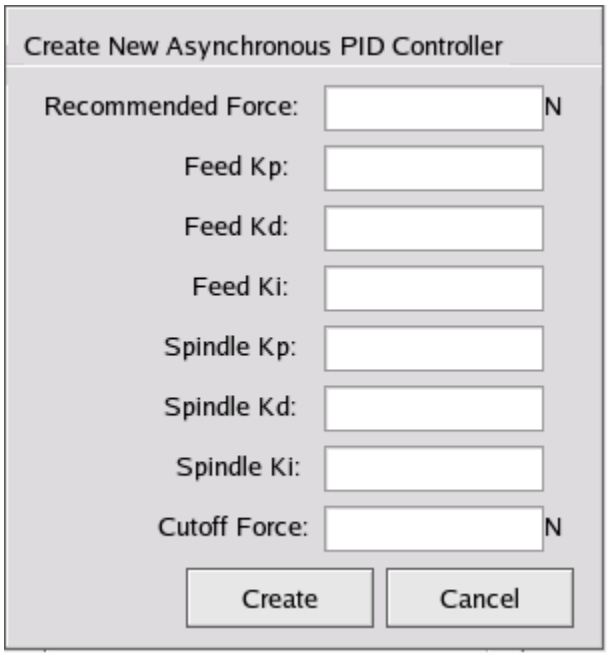


Figure 31 - GUI: Asynchronous PID Controller Setup

The "Create new Synchronous PID Controller" dialog (Figure 30), opened with the "Synchronous PID" button is identical to the "Create new Asynchronous PID Controller" dialog explained previously, except that it lacks an input box for Cutoff Force, as it is not needed for a synchronous PID controller.

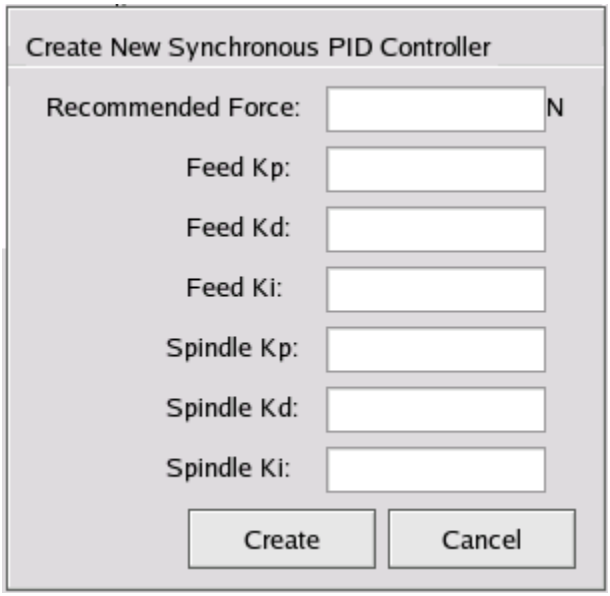


Figure 32 - GUI: Synchronous PID Controller Setup

If the cutting force maximum value (specified for each tool upon creation) is ever exceeded, a stop command is issued to the Numerical Controller within 10ms. Additionally, an

Alarm screen, seen below in Figure 31, is displayed, which encompasses the entirety of the GUI. Data Collection and plotting are immediately halted and adaptive control of the feedrate and spindle speed is suspended. All current stored data is saved to file, to permit later inspection of the error. The Alarm screen strobes between bright red and black and displays a highly contrasting warning message. The" Reset" button on the lower right hand side allows the user to end the alarm.

Figure 33 - GUI: Alarm Screen

4.4 Data Collection Results

All of the Advantech DAQ boards use the same API for collecting data: ActiveDaq. This means that if a program is able to successfully acquire data from one Advantech DAQ board, it will also succeed on any other board. As the main program was never successfully run on the PCM3355 board, the program to collect data from the Kistler 5070A charge amplifier was tested on an Ubuntu computer using an Advantech USB 4718. Data was successfully collected in this manner. The results are shown in Figure 32 below.

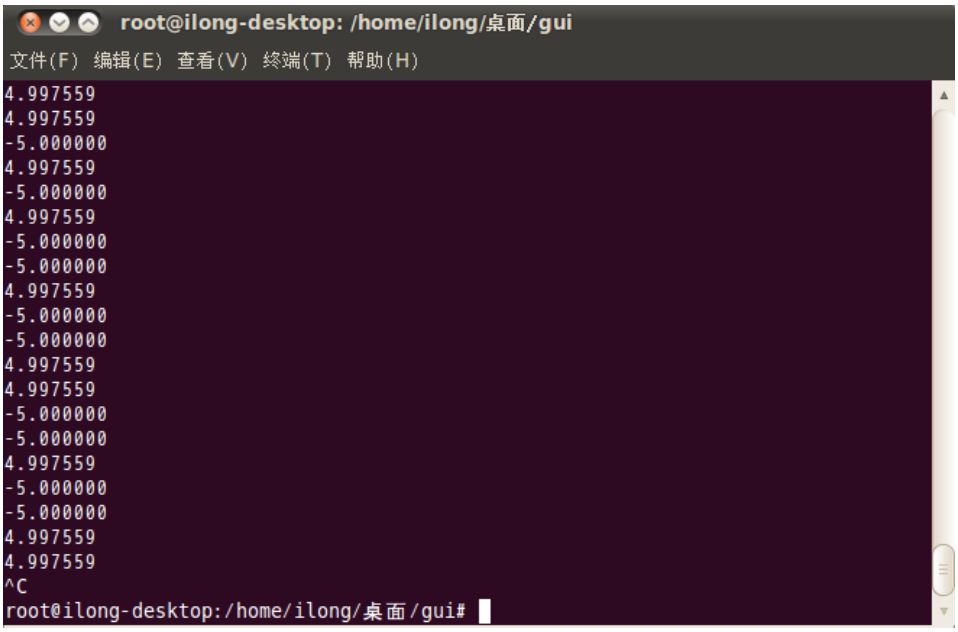


Figure 34 - Data Collection Testing Results

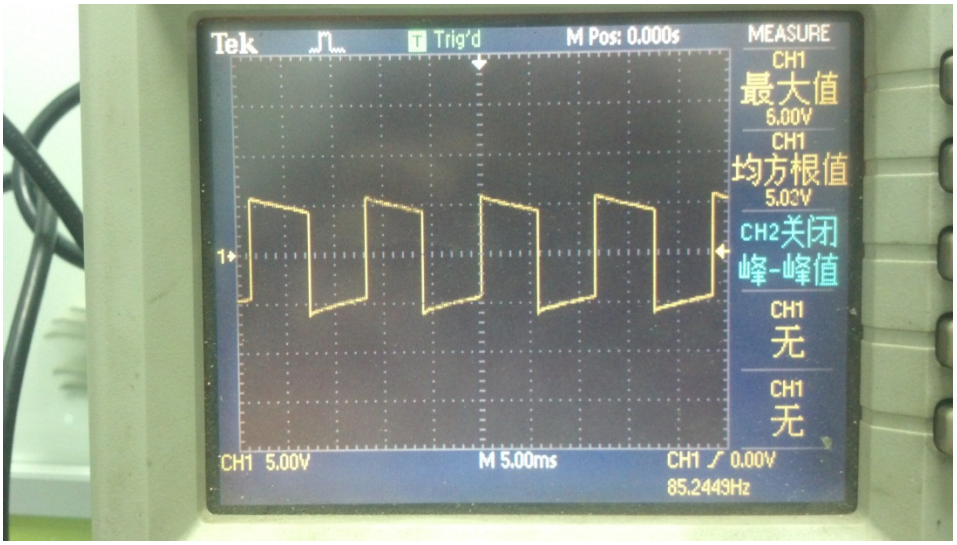


Figure 35 - Data Collection Testing: Input Square Wave

The software timer driven data collection code was able to accurately and timely collect voltage values ranging from -10V to +10V. This is the same as the range of Kistler 5070A charge amplifier output voltages, ensuring proper data reading. As seen in the above Figure 33, a -5V to +5V 85Hz square wave was read and the output printed to the terminal every 100 milliseconds.

Code was written to implement the use of the PCM3718HO hardware timer to collect data. The hardware interrupt driven code is located in Appendix C in the DataCollector.c

section. For both timing methods, examples from the Advantech user's manual were studied and modified with appropriate values [4]. Although both software and hardware timer driven code were implemented, only the software timer driven code was tested. This was due to the hardware timer driven data collection code needing to be run on the board.

Two versions of the Data Collecting program were written. The first uses a software timer (QtTimer) to poll the force acting on. The Kistler 9129AA Multicomponent Dynamometer used to directly gather the tri-axial forces acting upon the workpiece outputs electrical charges in three channels, which correspond to the forces [5, 6]. As the PCM 3718HO DAQ board's analog input ports can only measure changes in voltage, a Kistler 5070A charge amplifier was used to convert the dynamometer's output of electrical charge to a corresponding change in voltage.

4.5 CNC Communication Results

Although information about communicating on-the-fly with other companies' numerical controllers through RS32 or Ethernet could be found, no such material from HNC was available. Due to time constraints and the aforementioned, this task of the project was left uncompleted.

4.6 Summary

Overall, the three control algorithms conceptualized in the methodology section were implemented and tested in MATLAB. Data was successfully collected from and verified with an oscilloscope. Finally, an appropriate software system was developed in C and C++ to meet most of the project goals, and a user-friendly GUI was created.

5. Conclusion

Overall, the project was a success. Four out of six of the objectives laid out in the introduction were met, and another was partially met. A viable replacement for HNC's current machining tool wear and breakage monitor that operates in soft real-time and implements a tool life extending spindle speed and feedrate control system was produced. This system is able to detect significant machining tool abrasion and breakage through the analysis of cutting forces, extend tool life by actively controlling feedrate and spindle speed, continuously display plots of relevant information, and display an alarm in the event of excessive tool wear/breakage, all via a user-friendly interface. Functionality to back the machining tool off of the workpiece within 10ms of extreme cutting tool abrasion/tool breakage was not implemented due to time constraints, and no available resources about communication protocol with the HNC210iB.

Although all of these functionalities were implemented, few could be tested due to setbacks such as not receiving fully functional hardware until the week before the end of the project and not being able to obtain an optimal hard drive / compact flash card. Accordingly, tasks whose outcome could be tested in simulation were, reducing the possibility of errors occurring when the physical system is fully realized. Even so, a significant portion of all the necessary work to fully implement the system has been accomplished.

6. Future Works

6.1 Response time

As currently implemented, the device runs on “soft” real-time. That is, the 10ms response time to tool breakage is met most of the time, however this is not guaranteed. As the program currently stands, it uses the Qt class Qtimer which has an accuracy on a Linux system such as this of +/- 5ms [7]. This is a similar problem for the 100ms response time for graphing and actively controlling the system.

In order to fix this issue, “hard” real-time must implemented using the hardware timer interrupts of the PCM3718HO. Code has been written to implement this feature; however it has not been tested due to unsuccessfully setting up the hardware.

6.2 Communication with the Numerical Controller

Assuming the proper documentation can be found, the dummy function currently implemented would have to be replaced by the proper protocols. The function calls are already implemented in the correct places, so replacing the protocols is all that would need to be done.

6.3 Live Testing

Assuming Red Hat 9 can be successfully installed with a larger compact flash card, the entire system would have to be tested and debugged. Some error would likely occur, and these would have to be addressed.

6.4 Database Implementation

An SQL database could be used as a more efficient storage method for individual tool and cutting operation information and sets. This would also allow tools that have been created to persist in the event that the program is closed.

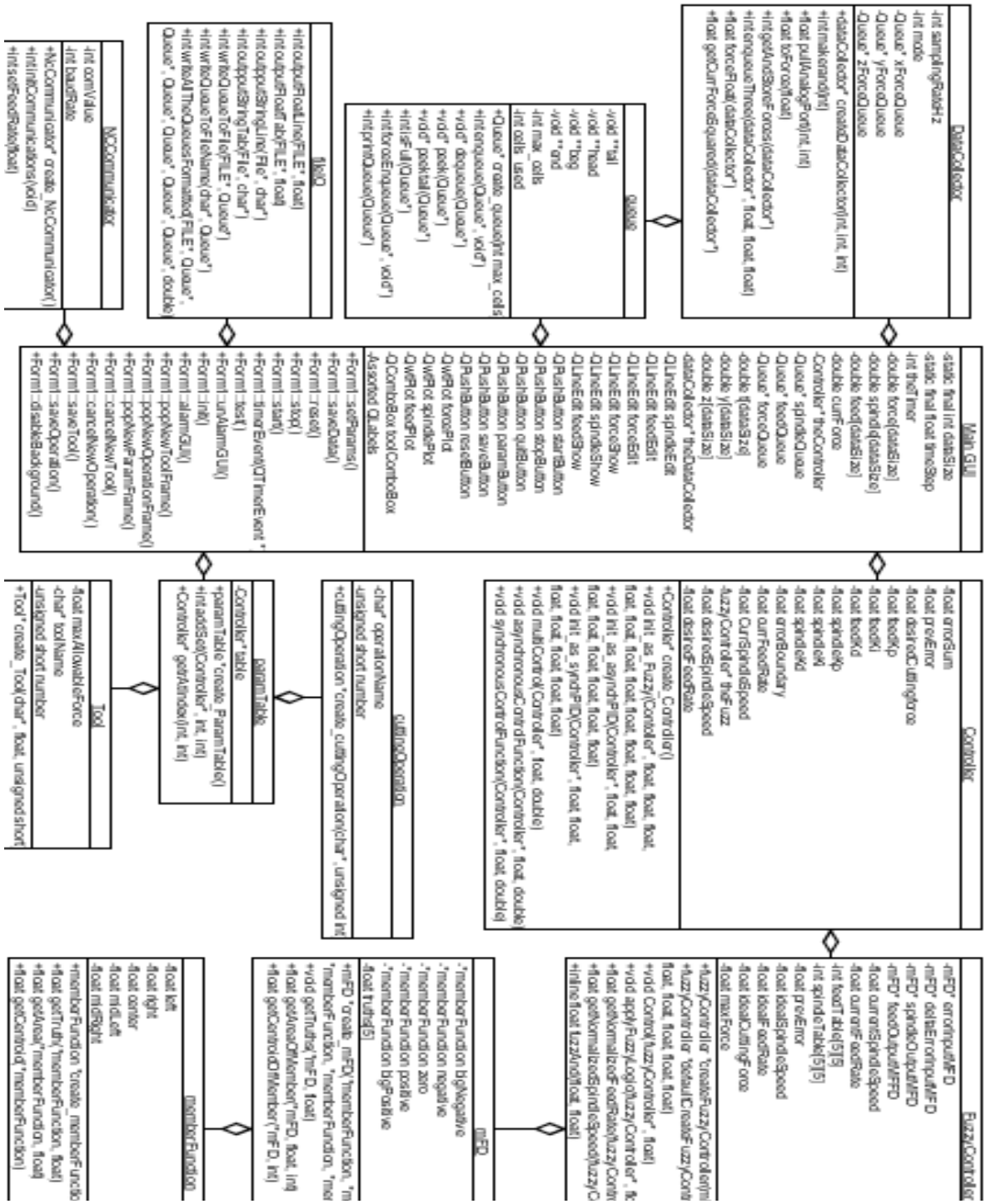
6.5 Load Band Monitoring

In the current implementation, the system uses an experimentally determined maximum cutting force value to decide when to trigger the alarm. Another option would be to use load band monitoring for deciding when to trigger the alarm. This would allow for early warnings in the case of imminent tool breakage. This would be easily implementable in the code base.

Works Cited

1. Koren, Yoram. "Adaptive Control Systems for Machining." *Manufacturing Perspective* 2.1 (1989). Web. Autumn 2011. <http://www-personal.umich.edu/~ykoren/uploads/Adaptive_control_systems_for_machining.pdf>.
2. Liang, Ming, Tet Yeap, Saeed Rahmati, and Zhixin Han. "Fuzzy Control of Spindle Power in End Milling Processes." *International Journal of Machine Tools and Manufacture* 21.14 (2002): 1487-496. *Science Direct*. 12 Oct. 2002. Web. Autumn 2011. <http://www.sciencedirect.com/science/article/pii/S0890695502001402>.
3. Oh, Young T., Gi D. Kim, and Chong N. Chu. "Design of a Drilling Torque Controller for a Machining Center." *The International Journal of Advanced Manufacturing Technology* 22.5-6 (2003): 329-35. *Springer Link*. Web. Autumn 2011. <http://www.springerlink.com/content/tag7gj1qdppb8wyn/fulltext.pdf>.
4. "PCM-3355 User Manual". Advantech Co., Ltd <http://support.advantech.com.tw/support/DownloadDatasheet.aspx>
5. "Kistler Type 5070A DataSheet". 2010 The Kistler Group <http://www.kistler.com/mediaaccess/5070A_BP__000-485e-03.10.pdf>
6. "Operating Instructions: Multichannel Charge Amplifier". 1996 The Kistler Group <http://ust.fme.vutbr.cz/obrabeni/rozvoj/Kistler-CNC-5019B-Description-En.pdf>
7. "QTimer Class Reference ". 2011 Nokia Corporation <http://developer.qt.nokia.com/doc/qt-4.8/qtimer.html>

Appendix A: Full UML Diagram



Appendix B: Matlab Test Code

```
u = .8;          %User Input
a = 4;           %User Input, depth of cut (mm)
p = 4;           %User Input, number of teeth
K = 500;         %User Input, specific cutting force
N = 2000;        %Starting Spindle Speed
V = 2262;        %Starting feedrate
threshold = 15; %User Input, cutting force threshold used for asynch
refFor = 500;    %User Input desired force

%Starting cutting forces
xForce = 600;
yForce = 400;
zForce = 100;

err = 0;         %current error
prevErr = 0;     %previous error
errSum = 0;      %sum of errors

%Feedrate Gains
fKp = 1;         %User Input
fKi = .01;       %User Input
fKd = .05;       %User Input

%Spindle Speed Gains
sKp = 1.3;         %User Input
sKi = .03;       %User Input
sKd = .01;       %User Input

%Testing arrarys
feedVal = zeros(1,50);
spinVal = zeros(1,50);
forceVal = zeros(1,50);
PID = zeros(1,50);
error = zeros(1,50);

%Magnitude of cutting force
curFor = sqrt(xForce^2 + yForce^2 + zForce^2);

%loop 50 times
for i=1:50

    forceVal(i) = curFor;

    err = refFor - curFor;          %calculcate error
    error(i) = err;
    errSum = err + errSum;          %calculate sum of errors

    %Calculate feedrate PID terms
    fP = err*fKp;
    fI = errSum*fKi;
    fD = ((err-prevErr)/.1)*fKd;
    deltaFeed = (fP+fI+fD);         %sum the terms
```

```
    %Calculate spindle speed PID terms
    sP = err*sKp;
    sI = errSum*sKi;
    sD = ((err-prevErr)/.1)*sKd;
    deltaSpindle = (sP+sI+sD);      %sum the terms

    PID(i) = deltaSpindle;

    %Difference between synchronous and asychrous control
    %   With the conditional statement, asychronous
    %   Without the conditional statement, synchronous
    if err > threshold || err < -threshold
        N = N - deltaSpindle;       %Update Spindle Speed
    else
         V = V + deltaFeed;         %Update Feedrate
    end

    feedVal(i) = V;
    spinVal(i) = N;

    %calculate new force
    f = V/(p*N);
    curFor = K*a*f^u;

    %set error to previous error
    prevErr = err;
end

%Graph the results
x=1:50;
figure(1);
plot(x,feedVal);
title('Feed Rate');

figure(2);
plot(x, spinVal);
title('Spindle Speed')

figure(3);
plot(x, forceVal);
title('Force');
xlabel('Time (.1s)');
ylabel('Force (N)');
```

Appendix C: Program Code

Controller

```
/**
 * \class Controller
 * \brief Contains struct for a controller. Control functions for both the Spindle speed and
feedrate
 * \author Brenden Gibbons
 * Contact: btgibbons@wpi.edu
 */
//allows use of this C file in CPP classes
#ifdef __cplusplus
 extern "C" {
 #endif
//include guards
#ifndef CONTROLLER_H
#define CONTROLLER_H
#include "tool.h"
#include "FuzzyController.h"

/** Struct to define a Controller
 */
typedef struct controller {
    unsigned short controllerType;//0 for synchPID, 1 for asynchPID, 2 for default Fuzzy
      //!The PID params
      float errorSum; //!Error sum for integral term
      float prevError; //!Previous Error to be used for derivative term

      float desiredCuttingForce;  //!previously determined optimal cutting force
      float feedKp; //!Proportional Gain for feed PID
      float feedKi; //!Integral Gain for feed PID
      float feedKd; //!Derivative Gain for feed PID
      float spindleKp; //!Proportional Gain for spindle PID
      float spindleKi; //!Integral Gain for spindle PID
      float spindleKd; //!Derivative Gain for spindle PID
      //only used for asynch
      float errorBoundary; //!error boundary used in asynchronous control mode
  //used for all
  float currFeedRate; //!The current feed rate of the NC
  float currSpindleSpeed; //!The current spindle speed of the NC
  /*
  float controllerGain;  //! The controller gain used in calculation
  float controllerSignal;  //! The controller signal
  float feedRateGain;  //! Gain used when adjusting the feedrate
  float previousCommandSignal;  //!The previous signal sent by the controller
*/
  //for fuzzy
  fuzzyController* theFuzz;
  float desiredSpindleSpeed; //!The optimal spindle speed for the current operatio
  float desiredFeedRate;
  //float desiredCuttingForce;  //!The optimal cutting force for the current operation
} Controller;

/** \brief Creates a Controller by allocating mem for the struct and its members and initializes
it.
  * \param cGain - (float) the Controller gain of the new Controller
  * \param cSignal - (float) The controller signal
  * \param fGain - (float) Gain used when adjusting the feedrate
  * \param pSignal -(float) The previous signal sent by the controller
  * \param cRate - (float) The current feed rate of the NC
  * \param dSpeed - (float) The optimal spindle speed for the current operation
  * \param dForce - (float) The optimal cutting force for the current operation
  * \param t - (tool *) pointer to the tool used for the current operation.  Has a maxForce
  * \return - (Controller *) returns a pointer to the newly allocated Controller
  */
Controller *create_Controller();
```

```
void init_as_Fuzzy(Controller *cont,float eNeg,  float E, float ePos, float deNeg, float DE,
float dePos, float recFeed, float recSpindle, float Ks, float u, float doc);

void init_as_asynchPID(Controller *cont, float desF, float fkp, float fki, float fkd, float skp,
float ski, float skd, float errbound);

void init_as_synchPID(Controller *cont, float desF, float fkp, float fki, float fkd, float skp,
float ski, float skd);
//will use control function of whatever type of controller
void multiControl(Controller *cont, float currForce, double timestep);

/** \brief Control function for determing the correct feed rate.
  * \param cont - (Controller*) A pointer to the controller in question
  * \param currForce - (float) The current cutting force read by the 3 axis sensor
  * \return - (float) returns the updated feed rate to be sent to the NC
  */
void asynchronousControlFunction(Controller *cont, float forceError, double timestep);

/** \brief Control function for determing the correct Spindle Speed.
  * \param cont - (Controller*) A pointer to the controller in question
  * \param currForce - (float) The current cutting force read by the 3 axis sensor
  * \return - (float) returns the updated spindle speed to be sent to the NC
  */
void synchronousControlFunction(Controller *cont, float forceError, double timestep);

//called when changing controllers or resetting to clear past sums
void clearOldData(Controller *cont);

#endif
#ifdef __cplusplus
 }
 #endif

#include <stdlib.h>
#include <stdio.h>
#include "Controller.h"

Controller *create_Controller(){

  Controller *new_controller; // Holds pointer to the newly-allocated Controller structure
  new_controller = (Controller *) malloc(sizeof(Controller));
  if (new_controller == NULL) return NULL; // Error--unable to allocate.
  // Fill in some of the struct
  new_controller->controllerType = 3;
  new_controller->errorSum = 0;
  new_controller->prevError = 0;

  return new_controller;
}

void init_as_asynchPID(Controller *cont, float desF, float fkp, float fki, float fkd, float skp,
float ski, float skd, float errbound){
    cont->controllerType = 1;//asynch
    cont->desiredCuttingForce = desF;
    cont->feedKp =fkp;
    cont->feedKi =fki;
    cont->feedKd =fkd;
    cont->spindleKp=skp;
    cont->spindleKi=ski;
    cont->spindleKd=skd;
    cont->errorBoundary = errbound;
    cont->currFeedRate = 0;
    cont->currSpindleSpeed = 0;
    //dont touch theFuzz
    cont->desiredSpindleSpeed=0;
    cont->desiredFeedRate=0;
}

void init_as_synchPID(Controller *cont, float desF, float fkp, float fki, float fkd, float skp,
float ski, float skd){
    cont->controllerType = 0;//synch
```

```
    printf("Setting desF to: %f",desF);
    cont->desiredCuttingForce = desF;
    cont->feedKp =fkp;
    cont->feedKi =fki;
    cont->feedKd =fkd;
    cont->spindleKp=skp;
    cont->spindleKi=ski;
    cont->spindleKd=skd;
    cont->errorBoundary = 0;
    cont->currFeedRate = 0;
    cont->currSpindleSpeed = 0;
    //dont touch theFuzz
    cont->desiredSpindleSpeed=0;
    cont->desiredFeedRate=0;
}

void init_as_Fuzzy(Controller *cont,float eNeg,  float E, float ePos, float deNeg, float DE,
float dePos, float recFeed, float recSpindle, float Ks, float u, float doc){

    cont->controllerType = 2;//fuzzy
    cont->desiredCuttingForce = 0; //actually we need to do the cutting force equations here!
    cont->feedKp =0;
    cont->feedKi =0;
    cont->feedKd =0;
    cont->spindleKp=0;
    cont->spindleKi=0;
    cont->spindleKd=0;
    cont->errorBoundary = 0;
    cont->currFeedRate = 0;
    cont->currSpindleSpeed = 0;
    cont->theFuzz=defaultCreateFuzzyController(eNeg, E, ePos, deNeg, DE, dePos, recFeed,
recSpindle);
    cont->desiredSpindleSpeed=recSpindle;
    cont->desiredFeedRate=recFeed;
}


void multiControl(Controller *cont, float currForce, double timestep){
    if(cont->controllerType == 0){
        synchronousControlFunction(cont, currForce, timestep);
    }
    else if(cont->controllerType == 1){
        asynchronousControlFunction(cont, currForce, timestep);
    }
    else if(cont->controllerType==2){
        Control(cont->theFuzz, currForce);
        cont->currSpindleSpeed = getNormalizedSpindleSpeed(cont->theFuzz);
        cont->currFeedRate = getNormalizedFeedRate(cont->theFuzz);
    }

}

void asynchronousControlFunction(Controller *cont, float currForce, double
timestep){
   //get the force error
  float error = cont->desiredCuttingForce - currForce;
  //temp float calculation variables
  float pVal, iVal, dVal;
  //sum the error
  cont->errorSum = error + cont->errorSum;
  //check whether to adjust the Spindle speed or Feed Rate
  if(error>cont->errorBoundary||error<cont->errorBoundary*-1){
      //large error so adjust the Feed Rate
      pVal = error * cont->feedKp;
      iVal = cont->errorSum * cont->feedKi;
      dVal = (float)((error - cont->prevError)/timestep) * cont->feedKd;
      cont->currFeedRate+=(pVal + iVal + dVal);
  }
  else{
      //small error so adjust the Spindle Speed
      pVal = error * cont->spindleKp;
```

```
      iVal = cont->errorSum * cont->spindleKi;
      dVal = (float)((error - cont->prevError)/timestep) * cont->spindleKd;
      cont->currSpindleSpeed+=(pVal + iVal + dVal);
  }
  cont->prevError = error;
}


void synchronousControlFunction(Controller *cont, float currForce, double timestep){
   //get the force error
  float error = cont->desiredCuttingForce - currForce;
  printf("desF: %f\n", cont->desiredCuttingForce);
  printf("currF: %f\n", currForce);
  printf("Error: %f\n", error);
  //temp float calculation variables
  float pVal, iVal, dVal;
  //sum the error
  cont->errorSum = error + cont->errorSum;
  //Adjust both the Feed rate and the Spindle speed

      //adjust Feed Rate
      pVal = error * cont->feedKp;
      iVal = cont->errorSum * cont->feedKi;
      dVal = (float)((error - cont->prevError)/timestep) * cont->feedKd;
      cont->currFeedRate+=(pVal + iVal + dVal);

      //adjust the Spindle Speed
      pVal = error * cont->spindleKp;
      iVal = cont->errorSum * cont->spindleKi;
      dVal = (float)((error - cont->prevError)/timestep) * cont->spindleKd;
      cont->currSpindleSpeed+=(pVal + iVal + dVal);

  cont->prevError = error;
}

void clearOldData(Controller *cont){
    cont->prevError = 0;
    cont->errorSum = 0;
}
```

Cutting Operation

```
#ifdef __cplusplus
 extern "C" {
 #endif

#ifndef CUTTINGOPERATION_H
#define CUTTINGOPERATION_H

/** Struct to define a tool and related functions
 */
typedef struct cuttingoperation {
  char *operationName;
  unsigned short number; //for easy ID
  }cuttingOperation;

/** \brief Creates a tool by allocating mem for the struct and its members and initializes it.
  * \param t- (float) the Controller gain of the new Controller
  * \param mat - (float) The controller signal
  * \param maxForce - (float) Gain used when adjusting the feedrate
  * \param refForce -(float) The previous signal sent by the controller
  * \return - (Controller *) returns a pointer to the newly allocated Controller
  */
cuttingOperation* create_cuttingOperation(char *name, unsigned short num);

#endif

#ifdef __cplusplus
 }
 #endif
```

```
#include <stdlib.h>
#include <string.h>
#include "cuttingoperation.h"

cuttingOperation* create_cuttingOperation(char* name, unsigned short num){
  cuttingOperation* new_operation;
  new_operation = (cuttingOperation *) malloc(sizeof(cuttingOperation));
  new_operation->operationName = name;
  new_operation->number = num;
  return new_operation;
}
```

Data Collector

```
/**
 * \class DataCollector
 * \brief Contains struct for a dataCollector, and functions for collecting data
 * \author Brenden Gibbons
 * Contact: btgibbons@wpi.edu
 */

//allows use of this C file in CPP classes
#ifdef __cplusplus
 extern "C" {
 #endif
//include guards
#ifndef DATACOLLECTOR_H
#define DATACOLLECTOR_H
#include "queue.h"

/** Struct to define a Controller
 */
typedef struct DataCollector {
  int samplingRateHz;         //! the AD sampling rate (Hz)
  int mode;           //! 1 for sim,  0 for real
  Queue *xForceQueue;  //! pointer to the z Force Queue
  Queue *yForceQueue;  //! pointer to the y Force Queue
  Queue *zForceQueue;  //! pointer to the z Force Queue
  int lastvalue;              //! used for simulation
  int cf;                     //! used for simulation
  int cs;                     //! used for simulation
} dataCollector;

/** \brief Create and initialize a DataCollector
    \param size - (int) The size of the three queues
    \param srate - (int) The sampling rate of data acq.
    \param mode - (int) 1 if simulation,  0 if real
    \return - (*dataCollector) The pointer to the new dataCollector, NULL if fail
*/
dataCollector *createDataCollector(int size, int srate, int m);

/** \brief Make a random int
    \param high - (int) The highest random value to receive
    \return - (int) Pseudorandom number between 0 and high
*/
int makerand(int high);

/** \brief Get an analog voltage value off of a specified port
    \param channelNum - (int) The port number in question
    \param m - (int) 1 if simulation, 0 if run on board
    \return - (float) representing the AD value
*/
int pullAnalogPort(dataCollector* the_dataCollector, int channelNum, int m);

/** \brief Convert a raw AD input to a force
    \param rawInput - (float) The raw AD value
    \return - (float) number representing the axial force
*/
float toForce(int rawInput);
```

```
/** \brief Get and store the forces on all 3 axes in their respective queues
    \param the_dataCollector - (dataCollector*) The dataCollector in question
    \return - (int) 1 for success, 0 for failure
*/
int getAndStoreForces(dataCollector *the_dataCollector);

/** \brief return the magnitude of the latest entries in a dataCollector's Queues
    \param the_dataCollector - (dataCollector*) the dataCollector to act on
    \return - (float) The magnitude of the latest Queue entries
*/
float forceFloat(dataCollector *the_dataCollector);

float getCurrForceSquared(dataCollector* the_dataCollector);

//float getForceFromFS(dataCollector* the_dataCollector, Tool* t, cuttingOperation* co,
Controller* cont);

#endif
#ifdef __cplusplus
 }
 #endif

#include <stdlib.h>
#include <stdio.h>
/*
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <string.h>
#include <sys/mman.h>
#include <termios.h>
#include <signal.h>
#include <Advantech/advdevice.h>
*/
#include "queue.h"
#include "fileIO.h"
#include "DataCollector.h"

//#define MODE 0

dataCollector *createDataCollector(int size, int srate, int m){
dataCollector *new_dataCollector;//pointer to the new dataCollector
//allocate mem for datacollector
new_dataCollector = (dataCollector *) malloc(sizeof(dataCollector));
if (new_dataCollector == NULL) return NULL; // Error--unable to allocate.
  //fill in the struct
  //set the sample rate
  new_dataCollector->samplingRateHz = srate;
  //printf("The sampling rate is %d\n", srate);
  //printf("The sampling rateHZ is %d\n", new_dataCollector->samplingRateHz);
  //set the mode
  new_dataCollector->mode = m;
  //printf("The entered mode is %d\n", m);
  //printf("The recorded mode is is %d\n", new_dataCollector->mode);
  //set the Queues
  new_dataCollector->xForceQueue = create_queue(size);
  new_dataCollector->yForceQueue = create_queue(size);
  new_dataCollector->zForceQueue = create_queue(size);
  //make sure the Queues were allocated
  if((!new_dataCollector->xForceQueue)||(!new_dataCollector->yForceQueue)
  ||(!new_dataCollector->zForceQueue)){
    printf("Empty Queues");
    return NULL;
  }
  new_dataCollector->lastvalue = 1000;
return new_dataCollector;
}
```

```
int makerand(int high){
return rand()%(high+1); //returns a random number from 0 to high
}

int pullAnalogPort(dataCollector* the_dataCollector, int channelNum, int m){
//if we are not in computer simulation

  //we must pull from the appropriate port
  /*This can't run on the computer so its commented out
/* Step 1: Open Device */
/*
        PT_AIConfig AIConfig;
        PT_AIBinaryIn AIBinaryIn;
        PT_AIVoltageIn AIVoltageIn;
        PT_AIScale AIScale;
        unsigned short wdata;
        unsigned short channel;
        unsigned short gain;
        unsigned int buffer;
        float voltage = 0;
        char err_msg[100];
        char* filename = "/dev/advdaq0";
        int ret;
        int fd;

        ret = DRV_DeviceOpen(filename, &fd);
        if (ret) {     DRV_GetErrorMessage(ret, err_msg);
                printf("err msg: %s\n", err_msg);
                return -1;
        }

        memset(&AIConfig, 0, sizeof(PT_AIConfig));
        memset(&AIBinaryIn, 0, sizeof(PT_AIBinaryIn));
        memset(&AIVoltageIn, 0, sizeof(PT_AIVoltageIn));
*/
/* Step 3: Set Single-end or Differential */
/*      buffer = 0x0000;       /* 0: single-end */
        //set device properties
/*      ret = DRV_DeviceSetProperty(fd, CFG_AiChanConfig, &buffer, sizeof(unsigned int));
        if (ret) {
                DRV_GetErrorMessage(ret, err_msg);
                printf("err msg: %s\n", err_msg);

                DRV_DeviceClose(&fd);
                return -1;
        }
        /* Step 2: Config AI Setting */
/*      AIConfig.DasChan = channel;
        AIConfig.DasGain = gain;

        ret = DRV_AIConfig(fd, &AIConfig);
        if (ret) {
                DRV_GetErrorMessage(ret, err_msg);
                printf("err msg: %s\n", err_msg);

                DRV_DeviceClose(&fd);
                return -1;
        }

if(m==0){  //Binary In
        AIBinaryIn.chan = channel;
        AIBinaryIn.TrigMode = 0;
        AIBinaryIn.reading = &wdata;

        ret = DRV_AIBinaryIn(fd, &AIBinaryIn);
        if(ret){
        DRV_GetErrorMessage(ret, err_msg);
        printf("err msg:  %s\n", err_msg);
        DRV_DeviceClose(&fd);
        return -1;
        }
```

```
        AIScale.reading = wdata;
        AIScale.MaxVolt = 5.0;
        AIScale.MaxCount = 4095;
        AIScale.offset = 2048;
        AIScale.voltage = &voltage;

        ret = DRV_AIScale(fd, &AIScale);
        if(ret){
          DRV_GetErrorMessage(ret, err_msg);
        printf("err msg:  %s\n", err_msg);
        DRV_DeviceClose(&fd);
        return -1;
        }
        return voltage;
}
else if(m==1){  //voltage in
        AIVoltageIn.chan = channelNum;
        AIVoltageIn.gain = gain;
        AIVoltageIn.TrigMode = 0;
        AIVoltageIn.voltage = &voltage;

        ret=DRV_AIVoltageIn(fd, &AIVoltage);
        if(ret){
          DRV_GetErrorMessage(ret, err_msg);
        printf("err msg:  %s\n", err_msg);
        DRV_DeviceClose(&fd);
        return -1;
        }
        return voltage;

}
//sim
*/

//we must create a simulated AD value that is reasonable
  //this should give a lowish standard deviation
  int result = (makerand(128)+makerand(128)-makerand(128)-makerand(128))+the_dataCollector-
>lastvalue;
  the_dataCollector->lastvalue=result;
  return (float) result;



}

float toForce(int rawInput){
//simply convert raw ad data to force
//the raw output voltage rangle of the charge amp is between -10 and 10V
//there is a 12 bit AD converter, so between 0 and 4096 resolution
//first multiply by .0048828125Volts/Digital to convert to a voltage
float result;
result = rawInput*.0048828125;
//Next subtract ten volts to get the proper range
result = result-10;
//finally multiply by (Newtons/Volts)
result = result*(10000/20);
return result;
}

int getAndStoreForces(dataCollector *the_dataCollector){
//get the forces
float oner = toForce(pullAnalogPort(the_dataCollector,1,the_dataCollector->mode));
float twor = toForce(pullAnalogPort(the_dataCollector,2,the_dataCollector->mode));
float threer = toForce(pullAnalogPort(the_dataCollector,3,the_dataCollector->mode));
//allocate some memory
float* fp1;
fp1 = (float *) malloc(sizeof(float));
*fp1 = oner;
float* fp2;
fp2 = (float *) malloc(sizeof(float));
```

```
*fp2 = twor;
float* fp3;
fp3 = (float *) malloc(sizeof(float));
*fp3 = threer;
//Force enqueue the forces into their appropriate queues
//printf(" Forceenqueuing Dereferenced ONE: %f\n", *fp1);
forceEnqueue(the_dataCollector->xForceQueue, (void *)fp1);
//printf(" Forceenqueuing Dereferenced TWO: %f\n", *fp2);
forceEnqueue(the_dataCollector->yForceQueue, (void *)fp2);
//printf(" Forceenqueuing Dereferenced THREE: %f\n", *fp3);
forceEnqueue(the_dataCollector->zForceQueue, (void *)fp3);
return 0;
}

float forceFloat(dataCollector *the_dataCollector){
//make some float pointers
float totalForce;
float *xp;
float *yp;
float *zp;

//look at the most recent queue entries
xp = (float*)peekTail(the_dataCollector->xForceQueue);
yp = (float*)peekTail(the_dataCollector->yForceQueue);
zp = (float*)peekTail(the_dataCollector->zForceQueue);

//magnitude the forces
totalForce = ((*xp)*(*xp))+((*yp)*(*yp))+((*zp)*(*zp));

return totalForce;
}

float getCurrForceSquared(dataCollector* the_dataCollector){
    float totalForce;
    float xp=toForce(pullAnalogPort(the_dataCollector,0,the_dataCollector->mode));
    float yp=toForce(pullAnalogPort(the_dataCollector,1,the_dataCollector->mode));
    float zp=toForce(pullAnalogPort(the_dataCollector,2,the_dataCollector->mode));

    totalForce = ((xp)*(xp))+((yp)*(yp))+((zp)*(zp));
    return totalForce;
}
/*
float getForceFromFS(dataCollector* the_dataCollector, Tool* t, cuttingOperation* co, Controller*
cont)
{
    //float mf = f/(t->numteeth*s);
    //float force = exp((co->Ks*co->DoC),co->u);
    return 1.0;
}*/
```

File IO

```
/**
 * \class fileIO
 * \brief Functions for writing to file (saving data)
 * \author Brenden Gibbons
 * Contact: btgibbons@wpi.edu
 */

//allows use of this C file in CPP classes
#ifdef __cplusplus
 extern "C" {
 #endif
//include guards
#ifndef FILEIO_H
#define FILEIO_H
#include "queue.h"

/** \brief Outputs a float to a given FILE* and inserts a new line character
  * \param stream - (FILE*) A pointer to the file to be written to
```

```
  * \param f - (float) The float to print
  * \return - (int) 1 if success, 0 if fail
  */
int outputFloatLine(FILE *stream, float f);

/** \brief Outputs a float to a given FILE*, then a tab character
  * \param stream - (FILE*) A pointer to the file to be written to
  * \param f - (float) The float to print
  * \return - (int) 1 if success, 0 if fail
  */
int outputFloatTab(FILE *stream, float f);

/** \brief Outputs a String to a given FILE*, then a tab character
  * \param stream - (FILE*) A pointer to the file to be written to
  * \param string - (char*) The string to print
  * \return - (int) 1 if success, 0 if fail
  */
int outputStringTab(FILE *stream, char* string);

/** \brief Outputs a String to a given FILE*, then a newline character
  * \param stream - (FILE*) A pointer to the file to be written to
  * \param string - (char*) The string to print
  * \return - (int) 1 if success, 0 if fail
  */
int outputStringLine(FILE *stream, char* string);

/** \brief write a Queue in a column to a file
    \param file- (FILE*) a pointer to the file we wish to write to
    \param queue- (Queue) a pointer to the Queue we wish to write
    \return - (int) 1 for success, 0 for failure
*/
int writeQueueToFile(FILE* file, Queue* queue);

/** \brief Creates a file with a given name and writes a queue to it
    \param fname - (char*) the file name as a string
    \param queue- (Queue) a pointer to the Queue we wish to write
    \return - (int) 1 for success, 0 for failure
*/
int writeQueueToFileName(char* fname, Queue* queue);

/** \brief write 6 queues formatted in columns to a file
    \param file- (FILE*) a pointer to the file we wish to write to
    \param queue1- (Queue) a pointer to the first Queue we wish to write
    \param queue2- (Queue) a pointer to the second Queue we wish to write
    \param queue3- (Queue) a pointer to the third Queue we wish to write
    \param queue4- (Queue) a pointer to the fourth Queue we wish to write
    \param queue5- (Queue) a pointer to the fifth Queue we wish to write
    \param queue6- (Queue) a pointer to the sixth Queue we wish to write
    \param time - the time step
    \return - (int) 1 for success, 0 for failure
*/
int writeAllTheQueuesFormatted(FILE* file, Queue* queue1, Queue* queue2,
                                        Queue* queue3, Queue* queue4,
                                        Queue* queue5, Queue* queue6, double time);

#endif
#ifdef __cplusplus
 }
 #endif


#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <stdlib.h>
#include "queue.h"

int outputFloatLine(FILE *stream, float f){
//if the inputted file stream hasnt been terminated
if (stream != '\0'){
  //print to file the float
```

```
  fprintf(stream, "%.2f\n", f);
}
//file pointer is terminated or null
else {
  //print error
  printf("Error: %d: %s.\n", errno, strerror(errno));
  return 0;
}
return 1;
}

int outputFloatTab(FILE *stream, float f){
//if the inputted file stream hasnt been terminated
if (stream != '\0'){
  //print to file the float
  fprintf(stream, "%.2f\t", f);
}
//file pointer is terminated or null
else {
  //print error
  printf("Error: %d: %s.\n", errno, strerror(errno));
  return 0;
}
return 1;
}

int outputStringTab(FILE *stream, char* string){
//if the inputted file stream hasnt been terminated
if (stream != '\0'){
  //print to file the string
  fprintf(stream, "%s\t", string);
}
//file pointer is terminated or null
else {
  //print error
  printf("Error: %d: %s.\n", errno, strerror(errno));
  return 0;
}
return 1;
}

int outputStringLine(FILE *stream, char* string){
//if the inputted file stream hasnt been terminated
if (stream != '\0'){
  //print to file the string
  fprintf(stream, "%s\n", string);
}
//file pointer is terminated or null
else {
  //print error
  printf("Error: %d: %s.\n", errno, strerror(errno));
  return 0;
}
return 1;
}

int writeQueueToFile(FILE* file, Queue* queue){
//loop counter
int i;
//temporary float
for(i = 0; i<queue->cells_used;i++){
  //dequeue from the queue, print the value to a file, then re-enqueue
  float* tempfloat;
  tempfloat=(float*)malloc(sizeof(float));
  tempfloat=(float *) dequeue(queue);
  outputFloatLine(file,(*tempfloat));
  enqueue(queue, (void*)tempfloat);
}
//I want to free it to avoid mem leak, but there is a segfault
//free(tempfloat);
fclose(file);
```

```
}

int writeQueueToFileName(char* fname, Queue* queue){
//loop counter
int i;
//temporary float
float* tempfloat;
tempfloat=(float*)malloc(sizeof(float));
//make new file
FILE *file = fopen(fname, "w"); // Open/create file
//make sure it was created
if (file != '\0'){
  for(i = 0; i<queue->cells_used;i++){
    //dequeue from the queue, print the value to a file, then re-enqueue
    tempfloat=(float *) dequeue(queue);
    outputFloatLine(file, *tempfloat);
    enqueue(queue, (void*)tempfloat);
  }
  //free the tempfloat
  free(tempfloat);
  fclose(file);
  return 1;
}
//failed
return 0;
}

int writeAllTheQueuesFormatted(FILE* file, Queue* queue1, Queue* queue2,
                                        Queue* queue3, Queue* queue4,
                                        Queue* queue5, Queue* queue6, double timeStep)
{
//loop counter
int i;
//temporary float
float* tempfloat;
//all queues will have the same number of entries, so only need to for with 1
outputStringTab(file, "Time(s)");
outputStringTab(file, "C Force");
outputStringTab(file, "FeedRate");
outputStringTab(file, "SSpeed");
outputStringTab(file, "xFqueue");
outputStringTab(file, "yFqueue");
outputStringLine(file, "zFqueue");
for(i = 0; i<queue1->cells_used;i++){
//print the time (like a boss)
outputFloatTab(file, (float)((double)i*timeStep*-1));
//dequeue from the 1queue, print the value to a file, then re-enqueue
tempfloat=(float*)malloc(sizeof(float));
tempfloat= (float *) dequeue(queue1);
outputFloatTab(file, *tempfloat);
enqueue(queue1, (void*)tempfloat);
//dequeue from the 2queue, print the value to a file, then re-enqueue
tempfloat=(float*)malloc(sizeof(float));
tempfloat=(float *) dequeue(queue2);
outputFloatTab(file, *tempfloat);
enqueue(queue2, (void*)tempfloat);
//dequeue from the 3queue, print the value to a file, then re-enqueue
tempfloat=(float*)malloc(sizeof(float));
tempfloat=(float *) dequeue(queue3);
outputFloatTab(file, *tempfloat);
enqueue(queue3, (void*)tempfloat);
//dequeue from the 4queue, print the value to a file, then re-enqueue
tempfloat=(float*)malloc(sizeof(float));
tempfloat=(float *) dequeue(queue4);
outputFloatTab(file, *tempfloat);
enqueue(queue4, (void*)tempfloat);
//dequeue from the 5queue, print the value to a file, then re-enqueue
tempfloat=(float*)malloc(sizeof(float));
tempfloat=(float *) dequeue(queue5);
outputFloatTab(file, *tempfloat);
enqueue(queue5, (void*)tempfloat);
```

```
//dequeue from the 6queue, print the value to a file, then re-enqueue
tempfloat=(float*)malloc(sizeof(float));
tempfloat=(float *) dequeue(queue6);
outputFloatLine(file, *tempfloat);
enqueue(queue6, (void*)tempfloat);
}
fclose(file);
return 1; //success!
}
```

Fuzzy Controller

```
/**
 * \class FuzzyController
 * \brief A fully customizable fuzzy controller for use in the GUI
 * \author Brenden Gibbons
 * Contact: btgibbons@wpi.edu
 */

//allows the use of this C file in CPP classes
#ifdef __cplusplus
 extern "C" {
 #endif
//include guards
#ifndef FUZZYCONTROLLER_H
#define FUZZYCONTROLLER_H

//#defines for member functions
#define EBN  -100
#define DEBN -100
#define SBN  1
#define FBN  1
#define EN   -50
#define DEN  -50
#define SN   2
#define FN   2
#define Z    3
#define EP   50
#define DEP  50
#define SP   4
#define FP   4
#define EBP  100
#define DEBP 100
#define SBP  5
#define FBP  5

#include "mFD.h"
/** Struct to define a FuzzyController
 */
typedef struct FuzzyController {
  mFD* errorInputMFD;  //! The mFD for input error
  mFD* deltaErrorInputMFD;  //! The mFD for input delta error
  mFD* spindleOutputMFD;  //! The mFD for output spindle
  mFD* feedOutputMFD;  //! The mFD for output spindle
  float currentSpindleSpeed; //! Holds the last updated spindle speed
  float currentFeedRate; //! Holds the last updated feed rate
  int feedTable[5][5];  //! table for feed fuzzy rules/sets
  int spindleTable[5][5]; //! table for spindle fuzzy rules/sets
  float prevError;  //! The previous force error
  float idealSpindleSpeed; //! Holds the ideal spindle speed (used in auto-creation)
  float idealFeedRate; //! Holds the ideal feed rate (used in auto-creation)
  float idealCuttingForce; //! Holds the ideal feed rate (used in auto-creation)
  float maxForce; //! force that triggers the alarm
}fuzzyController;

/** \brief Create and initialize a fuzzyController
    \param eI - (mFD*) The input error mFD
    \param dEI - (mFD*) The input change in error mFD
    \param sO - (mFD) The output spindle mFD
    \paramfO - (mFD*) The output feed mFD
```

```
    \return - (*fuzzyController) The pointer to the new fuzzyController, NULL if fail
*/
fuzzyController *createfuzzyController(mFD* eI, mFD* dEI, mFD* sO, mFD* fO);

/** \brief Create and initialize a default fuzzyController
    \return - (*fuzzyController) The pointer to the new fuzzyController, NULL if fail
*/
fuzzyController *defaultCreateFuzzyController(float eNeg, float E, float ePos, float deNeg, float
DE, float dePos, float recFeed, float recSpindle);

/** \brief Applies the INPUT fuzzy logic stage and calls applyFuzzyLogic for OUTPUT stage
    \param FC - (fuzzyController*) The fuzzyController
    \param forceIn - (float) The current cutting force
    \return - (void) Sets the currentFeedRate and currentSpindleSpeed
*/
void Control(fuzzyController * FC, float forceIn);

/** \brief Applies the OUTPUT stage of fuzzy control
    \param FC - (fuzzyController*) The fuzzyController
    \param errorTruths - (float []) array of error truths
    \param deltaErrorTruths - (float[]) array of change in error truths
    \return - (void) Sets the currentFeedRate and currentSpindleSpeed
*/
void applyFuzzyLogic(fuzzyController * FC, float errorTruths[], float deltaErrorTruths[]);

/** \brief Return the Feed Rate to be sent to the mill
    \param FC - (fuzzyController*) The fuzzyController to get the feed rate of
    \return - (*float) The normalized feedrate
*/
float getNormalizedFeedRate(fuzzyController * FC);

/** \brief Return the Spindle to be sent to the mill
    \param FC - (fuzzyController*) The fuzzyController to get the spindle speed of
    \return - (*float) The normalized spindle speed
*/
float getNormalizedSpindleSpeed(fuzzyController * FC);

//Inline function for ANDing fuzzy logic style
inline float fuzzAnd(float a, float b);

#endif
#ifdef __cplusplus
 }
#endif

#include <stdlib.h>
#include <stdio.h>
#include "FuzzyController.h"

fuzzyController *createfuzzyController(mFD* eI, mFD* dEI, mFD* sO, mFD* fO){
//pointer to the new mFD
fuzzyController *new_fuzzyController;
//allocate mem for mFD
new_fuzzyController = (fuzzyController *) malloc(sizeof(fuzzyController));
if (new_fuzzyController == NULL) return NULL; // Error--unable to allocate.
//set the mFDs
new_fuzzyController->errorInputMFD=eI;
new_fuzzyController->deltaErrorInputMFD=dEI;
new_fuzzyController->spindleOutputMFD=sO;
new_fuzzyController->feedOutputMFD=fO;
//here is where all the fuzzy sets are defined in fuzzy tables
//These are based on values from "Fuzzy Control of spindle power in end milling process"
//for Feeds
new_fuzzyController->feedTable[0][0] = FN;
new_fuzzyController->feedTable[1][0] = FN;
new_fuzzyController->feedTable[2][0] = FBN;
new_fuzzyController->feedTable[3][0] = FBN;
new_fuzzyController->feedTable[4][0] = FBN;
new_fuzzyController->feedTable[0][1] = FN;
new_fuzzyController->feedTable[1][1] = FN;
new_fuzzyController->feedTable[2][1] = FN;
```

```
new_fuzzyController->feedTable[3][1] = FN;
new_fuzzyController->feedTable[4][1] = FBN;
new_fuzzyController->feedTable[0][2] = FP;
new_fuzzyController->feedTable[1][2] = FP;
new_fuzzyController->feedTable[2][2] = Z;
new_fuzzyController->feedTable[3][2] = FN;
new_fuzzyController->feedTable[4][2] = FN;
new_fuzzyController->feedTable[0][3] = FP;
new_fuzzyController->feedTable[1][3] = FP;
new_fuzzyController->feedTable[2][3] = FP;
new_fuzzyController->feedTable[3][3] = Z;
new_fuzzyController->feedTable[4][3] = Z;
new_fuzzyController->feedTable[0][4] = FBP;
new_fuzzyController->feedTable[1][4] = FBP;
new_fuzzyController->feedTable[2][4] = FBP;
new_fuzzyController->feedTable[3][4] = FP;
new_fuzzyController->feedTable[4][4] = FP;
//for spindle speed
new_fuzzyController->spindleTable[0][0] = SN;
new_fuzzyController->spindleTable[1][0] = SN;
new_fuzzyController->spindleTable[2][0] = SN;
new_fuzzyController->spindleTable[3][0] = SN;
new_fuzzyController->spindleTable[4][0] = SN;
new_fuzzyController->spindleTable[0][1] = Z;
new_fuzzyController->spindleTable[1][1] = SN;
new_fuzzyController->spindleTable[2][1] = SN;
new_fuzzyController->spindleTable[3][1] = SN;
new_fuzzyController->spindleTable[4][1] = SN;
new_fuzzyController->spindleTable[0][2] = SP;
new_fuzzyController->spindleTable[1][2] = SP;
new_fuzzyController->spindleTable[2][2] = Z;
new_fuzzyController->spindleTable[3][2] = SN;
new_fuzzyController->spindleTable[4][2] = SN;
new_fuzzyController->spindleTable[0][3] = SP;
new_fuzzyController->spindleTable[1][3] = SP;
new_fuzzyController->spindleTable[2][3] = SP;
new_fuzzyController->spindleTable[3][3] = Z;
new_fuzzyController->spindleTable[4][3] = Z;
new_fuzzyController->spindleTable[0][4] = SP;
new_fuzzyController->spindleTable[1][4] = SP;
new_fuzzyController->spindleTable[2][4] = SP;
new_fuzzyController->spindleTable[3][4] = SP;
new_fuzzyController->spindleTable[4][4] = SP;
//set the other variables
new_fuzzyController->currentSpindleSpeed=1;
new_fuzzyController->currentFeedRate=1;
new_fuzzyController->prevError=0;
new_fuzzyController->idealSpindleSpeed=0;
new_fuzzyController->idealFeedRate=0;
new_fuzzyController->idealCuttingForce=0;
new_fuzzyController->maxForce = 99999;
//return the fuzzyController pointer
return new_fuzzyController;
}

fuzzyController *defaultCreateFuzzyController(float eNeg, float E, float ePos, float deNeg, float
DE, float dePos, float recFeed, float recSpindle)
{
  //MAKE THE FUZZY CONTROLLER (default membership functions equilaterally spaced)
  memberFunction* mf1;
  memberFunction* mf2;
  memberFunction* mf3;
  memberFunction* mf4;
  memberFunction* mf5;
  mFD* mFD1;
  mFD* mFD2;
  mFD* mFD3;
  mFD* mFD4;
  //input error
  float step = (ePos-E)/3;
  mf1=createMemberFunction(E-3*step,E-step,eNeg);
```

```
  mf2=createMemberFunction(E-2*step,E,E-step);
  mf3=createMemberFunction(E-step,E+step,E);
  mf4=createMemberFunction(E,E+2*step,E+step);
  mf5=createMemberFunction(E+step,E+3*step,ePos);
    mFD1 = createMFD(0,mf1,mf2,mf3,mf4,mf5);
  //input delta error
  step = (dePos-DE)/3;
  mf1=createMemberFunction(DE-3*step,DE-step,deNeg);
  mf2=createMemberFunction(DE-2*step,DE,DE-step);
  mf3=createMemberFunction(DE-step,DE+step,DE);
  mf4=createMemberFunction(DE,DE+2*step,DE+step);
  mf5=createMemberFunction(DE+step,DE+3*step,dePos);
  mFD2 = createMFD(0,mf1,mf2,mf3,mf4,mf5);
  //output spinlde
  mf1=createMemberFunction(.15,.51,.33);
  mf2=createMemberFunction(.48,.84,.66);
  mf3=createMemberFunction(.75,1.25,1);
  mf4=createMemberFunction(1.15,1.51,1.33);
  mf5=createMemberFunction(1.48,1.84,1.66);
  mFD3 = createMFD(1,mf1,mf2,mf3,mf4,mf5);
    //output feed
  mf1=createMemberFunction(.15,.51,.33);
  mf2=createMemberFunction(.48,.84,.66);
  mf3=createMemberFunction(.82,1.18,1);
  mf4=createMemberFunction(1.15,1.51,1.33);
  mf5=createMemberFunction(1.48,1.84,1.66);
  mFD4 = createMFD(1,mf1,mf2,mf3,mf4,mf5);
  fuzzyController* the_fuzzyController;
  the_fuzzyController = createfuzzyController(mFD1, mFD2, mFD3, mFD4);
  return the_fuzzyController;
}

void Control(fuzzyController * FC, float forceIn){
    //temporary floats
    float error, de;
    //get error and change in error
    error = forceIn-FC->idealCuttingForce;
    de=error-FC->prevError;
    //update the truths for all member functions in the input MFDs
    getTruths(FC->errorInputMFD, error);
    getTruths(FC->deltaErrorInputMFD, de);
    //feed in the truth arrays, first errorinput, then deltaerrorinput to applyFuzzyLogic
    applyFuzzyLogic(FC, FC->errorInputMFD->truths, FC->deltaErrorInputMFD->truths);
    //set the error to the previous error
    FC->prevError=error;
}

float getNormalizedSpindleSpeed(fuzzyController * FC){
    //multiply the ideal spindle speed by the (currentSpindleSpeed), which is
    //a percentage of ideal spindle speed, and return it
    return FC->idealSpindleSpeed*FC->currentSpindleSpeed;
}

float getNormalizedFeedRate(fuzzyController * FC){
    //multiply the ideal feed rate by the (currentFeedRate), which is
    //a percentage of ideal feed rate, and return it
     return FC->idealFeedRate*FC->currentFeedRate;
}

//takes in array of input truths and applies fuzzy logic rules and defuzzifies
//storing the result in the fuzzyController
void applyFuzzyLogic(fuzzyController * FC, float errorTruths[], float deltaErrorTruths[]){
//sums of Areas and sums of Areas*Centroids
float FACSum = 0;
float FASum = 0;
float SACSum = 0;
float SASum = 0;

//loop counters
int i, j;
```

```
//running through error
for(i = 0; i<5; i++){
  //running through change in error
  for(j = 0; j<5; j++){

  //if there is a triangle
      //dont use ==0 because of floating point errors
     if((deltaErrorTruths[j]>.01)&&(errorTruths[i]>.01)){
       //defuzzification - sum the area*centroid of all the triangles with truth
       //then divide by the sum of the areas
       FACSum +=
       getAreaOfMember(FC->feedOutputMFD,fuzzAnd(deltaErrorTruths[j],errorTruths[i]),
       FC->feedTable[j][i])*getCentroidOfMember(FC->feedOutputMFD, FC->feedTable[j][i]);

       FASum +=
       getAreaOfMember(FC->feedOutputMFD,fuzzAnd(deltaErrorTruths[j],errorTruths[i]),
       FC->feedTable[j][i]);

       SACSum +=
       getAreaOfMember(FC->spindleOutputMFD,fuzzAnd(deltaErrorTruths[j],errorTruths[i]),
       FC->spindleTable[j][i])
       *getCentroidOfMember(FC->spindleOutputMFD, FC->spindleTable[j][i]);

       SASum +=
       getAreaOfMember(FC->spindleOutputMFD,fuzzAnd(deltaErrorTruths[j],errorTruths[i]),
       FC->spindleTable[j][i]);

     }
  }
}
//set the FC currentSpindleSpeed to the centroid of the entire weighted area
FC->currentSpindleSpeed=(SACSum/SASum);
//set the FC  currentFeedRate to the centroid of the entire weighted area
FC->currentFeedRate=(FACSum/FASum);
}

inline float fuzzAnd(float a, float b) {
  //fuzzy AND logic returns the smaller of the two numbers
  return a > b ? b : a;
}
```

Member Function

```
/**
 * \class MemberFunction
 * \brief Member function used in fuzzy controller - Represents a triangle
 * \author Brenden Gibbons
 * Contact: btgibbons@wpi.edu
 */

//allows the use of this C file in CPP classes
#ifdef __cplusplus
 extern "C" {
 #endif

//include guards
#ifndef MEMBERFUNCTION_H
#define MEMBERFUNCTION_H

/** Struct to define a MemberFunction.  Only triangle member functions allowed
 */
typedef struct MemberFunction {
  float left;     //! The x val of left corner of the triangle member
  float right;  //! The x val of right corner of the triangle member
  float center;   //! The x val of the third vertex of the triangle
  float midLeft;   //! Point used in both area and centroid calcs
  float midRight;  //! Point used in both area and centroid calcs
}memberFunction;

/** \brief Create and initialize a MemberFunction
```

```
    \param left - (float) The left bottom x coord of the member function
    \param right - (float) The right bottom x coord of the member function
    \param mode - (float) The center top x coord of the member function
    \return - (*memberFunction) The pointer to the new memberFunction, NULL if fail
*/
memberFunction *createMemberFunction(float left, float right, float center);

/** \brief Get a truth value from a member function, given an input value.  INPUT phase
    \param mF - (*memberFunction) The member function in question pointer
    \param input - (float) The input value
    \return - (float) The truth value
*/
float getTruth(memberFunction* mF, float input);

/** \brief Gets area of the trapezoid formed by a line at y=truth and the memberfunction OUTPUT
PHASE
    \param mF - (*memberFunction) The member function in question pointer
    \param truth - (float) The truth value
    \return - (float) The Area of the trapezoid
*/
float getArea(memberFunction* mF, float truth);

/** \brief Gets centroid of the trapezoid formed by a line at y=truth and the memberfunction
OUTPUT PHASE
    \param mF - (*memberFunction) The member function in question pointer
    \return - (float) The centroid on the x axis of the trapezoid
*/
//special note!  if this is called before GetArea on a memberfunction, midLeft and midRight
//will be set to zero, so it will not find the centroid at all
float getCentroid(memberFunction* mF);

#endif
#ifdef __cplusplus
 }
#endif


#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "MemberFunction.h"

memberFunction *createMemberFunction(float left, float right, float center){
//pointer to the new memberFunction
memberFunction *new_memberFunction;
//allocate mem for datacollector
new_memberFunction = (memberFunction *) malloc(sizeof(memberFunction));
if (new_memberFunction == NULL) return NULL; // Error--unable to allocate.
//set all the members
new_memberFunction->left = left;
new_memberFunction->right = right;
new_memberFunction->center = center;
//these start as 0 because we can' know them without a truth value
new_memberFunction->midLeft = 0;
new_memberFunction->midRight = 0;
return new_memberFunction;
}

//for input phase
float getTruth(memberFunction* mF, float input){
    //if the input is within the memberfunction range
  if((input>mF->left)&&(mF->right>input)){
     //return the associated truth
    return 1-fabs((mF->left+mF->right-2*input)/(mF->right-mF->left));
  }
  //else the truth is 0
  else{
    return 0;
  }
}
```

```
//for output phase
float getArea(memberFunction* mF, float truth){
   //make temporary floats
  float totalArea, tipArea, slopeLeft, slopeRight;
  //first get the area of the whole triangle
  totalArea = (.5*(mF->right-mF->left));
  //find the slopes of the two edges
  slopeLeft = (1.0/(mF->center-mF->left));
  slopeRight = (1.0/(mF->right-mF->center));
  //find intersection of both edges with line y=truth
  mF->midLeft = mF->left+((1.0/slopeLeft)*truth);
  mF->midRight = mF->right-((1.0/slopeRight)*truth);
  //get the area of the small triangle at the top
  tipArea = (mF->midRight-mF->midLeft)*(1-truth)*.5;
  //return area of trapezoid
  return totalArea-tipArea;
}

//DONT CALL WITHOUT ABOVE FUNCTION FIRST
float getCentroid(memberFunction* mF){
  //temporary floats
  float a, b, c;
  //get three lengths representative of the trapezoid created in prev. function
  a = mF->midRight-mF->midLeft;
  b = mF->right-mF->left;
  c = mF->midLeft-mF->left;
  //Find the x axis centroid of the trapezoid using geometry
  float result = mF->midLeft+((2*a*c)+(a*a)+(c*b)+(a*b)+(b*b))/(3*(a+b));
  //return the centroid
  return result;
}
```

Member Function Diagram

```
/**
 * \class mFD
 * \brief Member Function diagram - holds 5 members and is used as part of a  fuzzyController
 * \author Brenden Gibbons
 * Contact: btgibbons@wpi.edu
 */

//allows the use of this C file in CPP classes
#ifdef __cplusplus
 extern "C" {
 #endif
//include guards
#ifndef MFD_H
#define MFD_H
/** Struct to define a MFD-membership function diagram - 5 entries for all
 */
#include "MemberFunction.h"

typedef struct MFD {
  memberFunction* bigNegative;  //! The memberFunction associated with big negative values
  memberFunction* negative;  //! The memberFunction associated with negative values
  memberFunction* zero;   //! The memberFunction associated with middle values
  memberFunction* positive;   //! The memberFunction associated with positive values
  memberFunction* bigPositive;  //! The memberFunction associated with big positive values
  float truths[5];  //! An array representing the truths of each member function in the MFD
} mFD;

/** \brief Create and initialize a MFD
    \param i - (int) functionality not implimented yet
    \param BNeg - (memberFunction*) The leftmost member function
    \param BNeg - (memberFunction*) The left member function
    \param BNeg - (memberFunction*) The middle member function
    \param BNeg - (memberFunction*) The right member function
    \param BNeg - (memberFunction*) The rightmost member function
    \return - (*mFD) The pointer to the new mFD, NULL if fail
*/
```

```
mFD* createMFD(int i, memberFunction* BNeg, memberFunction* Neg, memberFunction* Zer,
memberFunction* Pos, memberFunction* BPos);

//goes through each 5 membership diagrams and calls getTruth on them
//returns an array filled with the 5 truth values

/** \brief goes through each 5 membership diagrams and stores their truths for given input
    \param mFD - (MFD*) The mFD in question
    \param input - (float) Input value to get truths for
    \return - (void) The truths are stored in mFD->truths
*/
void getTruths(mFD* mFD, float input);

/** \brief gets the x axis centroid of the desired member
    \param mFD - (MFD*) The mFD in question
    \param i - (int) Indicates which member function to get the centroid of
    \return - (float) The desired centroid
*/
float getCentroidOfMember(mFD* mFD, int i);

/** \brief gets the area of the desired member
    \param mFD - (MFD*) The mFD in question
    \param i - (int) Indicates which member function to get the area of
    \return - (float) The desired area
*/
float getAreaOfMember(mFD* mFD,float truth, int i);

#endif
#ifdef __cplusplus
 }
#endif


#include <stdlib.h>
#include <stdio.h>
#include "mFD.h"

mFD *createMFD(int i, memberFunction* BNeg, memberFunction* Neg, memberFunction* Zer,
memberFunction* Pos, memberFunction* BPos){
//pointer to the new mFD
mFD *new_mFD;
//allocate mem for mFD
new_mFD = (mFD *) malloc(sizeof(mFD));
if (new_mFD == NULL) return NULL; // Error--unable to allocate.
//set the member triangles
new_mFD->bigNegative=BNeg;
new_mFD->negative=Neg;
new_mFD->zero = Zer;
new_mFD->positive=Pos;
new_mFD->bigPositive=BPos;
//return pointer to new MFD
return new_mFD;
}

void getTruths(mFD* mFD, float input){
//if the input is off-the-charts negative, the big negative truth will be 1
if(input<=mFD->bigNegative->left){
       mFD->truths[0]=1;
}
//else treat normally
else{
       mFD->truths[0] = getTruth(mFD->bigNegative, input);
}
//middle three memberfunctions
mFD->truths[1] = getTruth(mFD->negative, input);
mFD->truths[2] = getTruth(mFD->zero, input);
mFD->truths[3] = getTruth(mFD->positive, input);
//if the input is off-the-charts negative, the big negative truth will be 1
if(input>=mFD->bigPositive->right){
       mFD->truths[4]=1;
}
```

```
//else treat normally
else{
        mFD->truths[4] = getTruth(mFD->bigPositive, input);
}
}

float getAreaOfMember(mFD* mFD, float truth, int i){
//depending on the memberfunction stated and the truth value
//return the area
if(i==1){
return getArea(mFD->bigNegative, truth);
}
if(i==2){
return getArea(mFD->negative, truth);
}
if(i==3){
return getArea(mFD->zero, truth);
}
if(i==4){
return getArea(mFD->positive, truth);
}
if(i==5){
return getArea(mFD->bigPositive, truth);
}
else return 0;
}


float getCentroidOfMember(mFD* mFD, int i){
//depending on the memberfunction stated
//return the centroid
if(i==1){
return getCentroid(mFD->bigNegative);
}
if(i==2){
return getCentroid(mFD->negative);
}
if(i==3){
return getCentroid(mFD->zero);
}
if(i==4){
return getCentroid(mFD->positive);
}
if(i==5){
return getCentroid(mFD->bigPositive);
}
else return 0;
}
```

NCCommunicator

```
#ifdef __cplusplus
 extern "C" {
 #endif

#ifndef NCCOMMUNICATOR_H
#define NCCOMMUNICATOR_H

#include <stdlib.h>
/** Struct to define a Controller
 */
struct NCCommunicator {

int comValue;
int baudRate;

};

typedef struct NCCommunicator NcCommunicator;
```

```
NcCommunicator* create_NcCommunicator();

int initCommunications(NcCommunicator * nc);

int setSpindleSpeed(NcCommunicator * nc, float rpm);

int setFeedRate(NcCommunicator * nc, float mmPerMinute);

int alarmRoutine(NcCommunicator * nc);

#endif

#ifdef __cplusplus
 }
 #endif

#include <stdlib.h>
#include <stdio.h>
#include "NCCommunicator.h"
#include "Controller.h"

NcCommunicator* create_NcCommunicator(){
  NcCommunicator *tehpointer; // Holds pointer to the newly-allocated Queue structure.
  tehpointer = (NcCommunicator *) malloc(sizeof(NcCommunicator));
  return tehpointer;
}

int initCommunications(NcCommunicator * nc){
  printf("Initializing Communications");
  return 1;
}

int setSpindleSpeed(NcCommunicator * nc, float rpm){
  printf("Setting Spindle Speed to: %f rpm", rpm);
  return 1;
}

int setFeedRate(NcCommunicator * nc, float mmPerMinute){
  printf("Setting Feed Rate to: %f mm/min", mmPerMinute);
  return 1;
}

int alarmRoutine(NcCommunicator * nc){
    //set spindle speed to zero
    setSpindleSpeed(nc, 0);
    return 1;
}
```

Cutting Operation

```
#include <stdlib.h>
#include <string.h>
#include "cuttingOperation.h"

cuttingOperation* create_cuttingOperation(char* name, unsigned short num){
  cuttingOperation* cuttingOperation;
  new_operation = (cuttingOperation *) malloc(sizeof(cuttingOperation));
  new_operation->operationName = name;
  new_operation->number = num;
  return new_operation;
}
```

ParamTable

```
/**
 * \class ParamTable
 * \author Brenden Gibbons
 * Contact: btgibbons@wpi.edu
 */
```

```
//allows use of this C file in CPP classes
#ifdef __cplusplus
 extern "C" {
 #endif
//include guards
#ifndef PARAMTABLE_H
#define PARAMTABLE_H
#include "tool.h"
#include "cuttingoperation.h"
#include "Controller.h"
     typedef struct paramtable {
        Controller* table[25][25];
        unsigned short filltable[25][25];
     }paramTable;

     paramTable *create_paramTable();
     int addSet(paramTable* pt, Controller* newCont, int toolnum, int opnum);
     Controller* getAtIndex(paramTable* pt, int toolnum, int opnum);
     int confirmEntry(paramTable* pt, int toolnum, int opnum);

#endif
#ifdef __cplusplus
 }
 #endif


#include <stdlib.h>
#include <string.h>
#include "ParamTable.h"

paramTable* create_paramTable(){
  paramTable* new_paramTable;
  new_paramTable = (paramTable *) malloc(sizeof(paramTable));
  //init table to null
  int ii, jj;
  for(ii = 0; ii< 25; ii++){
      for(jj = 0; jj< 25; jj++){
       new_paramTable->filltable[ii][jj]=0;
      }
  }
  return new_paramTable;
}

int addSet(paramTable* pt, Controller* newCont, int toolnum, int opnum){
    pt->table[toolnum][opnum] = newCont;
    pt->filltable[toolnum][opnum] = 1;
    return 1;
}

Controller* getAtIndex(paramTable* pt, int toolnum, int opnum){
    //printf("getting at index %d , %d \n", toolnum, opnum);
    return pt->table[toolnum][opnum];
}

int confirmEntry(paramTable* pt, int toolnum, int opnum){
    return pt->filltable[toolnum][opnum];
}
```

Queue

```
/**
 * \class Queue
 * \brief Queue used for all types of things
 * \author Brenden Gibbons
 * Contact: btgibbons@wpi.edu
 */

//allows the use of this C file in CPP classes
#ifdef __cplusplus
 extern "C" {
```

```
 #endif
//include guards
#ifndef QUEUE_H
#define QUEUE_H
/** Struct to define a queue; each entry can hold a pointer to anything.
 */
typedef struct queue {
  void **tail;     //! Pointer to tail (FI) element of queue
  void **head;  //! Pointer to head (FO) of queue
  void **beg;   //! Pointer to the beginning of the queue
  void **end;   //! Pointer to the end of the queue
  int max_cells;  //! Maximum number of entries in the queue
  int cells_used; //! Currently used number of cells
}Queue;

/** \brief - Creates a queue by allocating a queue structure, initializing it,
 *  and allocating memory to hold the queue entries.
 * \param max_cells - (int) Maximum entries in the queue
 * \return - (Queue*) Pointer to newly-allocated Queue structure, NULL if error.
 */
Queue *create_queue(int max_cells);

/** \brief - Enqueues a pointer onto the tail of a Queue.
 * \param which_queue - (Queue*)Pointer to queue you want to enqueue onto.
 * \param ptr - (void*)Pointer to be enqueued.
 * \return - (int) 0 if successful, -1 if not.
 */
int enqueue(Queue *which_queue, void *ptr);

/** Forces a pointer onto the tail of a Queue, regardless if it is full
 * \param which_queue - (Queue*) Pointer to queue you forceEnqueue
 * \param ptr - (void*)Pointer to be force enqueued.
 * \return - (int) 0 if successful, -1 if not.
 */
int forceEnqueue(Queue *which_queue, void *ptr);

/** \brief - Dequeues a pointer from head of Queue,and returns it.
 * \param which_queue - (Queue*)Pointer to Queue you want to dequeue from.
 * \return - (void*) head entry of queue, NULL if queue is empty.
 */
void* dequeue(Queue *which_queue);

/** \brief - returns a pointer to what the head of the Queue was pointing to
 * \param which_queue - (Queue*)Pointer to Queue you want to peek.
 * \return - (void*) head entry of queue, NULL if queue is empty.
 */
void* peek(Queue *which_queue);

/** \brief - returns a pointer to what the tail of the Queue was pointing to
 * \param which_queue - (Queue*)Pointer to Queue you want to peek.
 * \return - (void*) tail entry of queue, NULL if queue is empty.
 */
void* peekTail(Queue *which_queue);

/** \brief - returns an int indicating whether or not the Queue is full
 * \param which_queue - (Queue*)Pointer to Queue
 * \return - (int) 1 if full, 0 otherwise
 */
int isFull(Queue *which_queue);

/** Prints an entire queue to standard output
 * \param which_queue - (Queue*)Pointer to queue to print
 * \return - (void)
 */
void printQueue(Queue *which_queue);

#endif
#ifdef __cplusplus
 }
 #endif
```

```
#include <stdlib.h>
#include <stdio.h>
#include "/home/btgibbons/MQP/queue.h"


Queue *create_queue(int max_cells) {
Queue *new_queue; // Holds pointer to the newly-allocated Queue structure.
new_queue = (Queue *) malloc(sizeof(Queue));
if (new_queue == NULL) return NULL; // Error--unable to allocate.
  // Fill in the struct
  new_queue->max_cells = max_cells;
  new_queue->cells_used = 0; // Empty to start
  // Now allocate space for the queue entries.
  void** arr = (void**) calloc (max_cells, sizeof(void*));
  new_queue->beg = arr; //que beg now points to new queue start
  new_queue->end = &arr[max_cells-1]; //end now points to end
  // Make the Queue end pointer
  if (new_queue->beg == NULL) {
    free(new_queue); // Unable to allocate queue entries, so free struct.
    return NULL;
  }
  new_queue->head = new_queue->beg; // Start at beginning
  new_queue->tail = new_queue->beg; //start at beginning
  return new_queue;
}

int enqueue(Queue *which_queue, void *ptr) {
// Check if queue is already full
if ((which_queue->cells_used) >= (which_queue->max_cells)) {
  which_queue->cells_used = which_queue->max_cells; // Fixed
return -1;  // Queue overflow.
}
//check if tail is running off the end
if((which_queue->tail) > (which_queue->end)){
  //wrap around if it is
  which_queue->tail = which_queue->beg;
}
// enqueue
*(which_queue->tail) = ptr;  // Store the pointer on the queue
(which_queue->tail)++;       // Point to next free cell
(which_queue->cells_used)++; //increment cellsused
return 0;  // Success
}

int forceEnqueue(Queue *which_queue, void *ptr){
//check if the queue is full
if(isFull(which_queue)){
  //full, so dequeue
  free((float*)dequeue(which_queue));
  //then enqueue
  return enqueue(which_queue,ptr);
}
else{
  //queue not full, so simply enqueue
  return enqueue(which_queue,ptr);
}
}

void* dequeue(Queue *which_queue) {
// Check if queue is empty
if ((which_queue->cells_used) <= 0) {
  which_queue->cells_used = 0; //insurance
  return NULL;  // Queue empty
}
//check if head is running off the end
if((which_queue->head) > (which_queue->end)){
  //wrap head around to beg
  which_queue->head = which_queue->beg;
}
// dequeue.
void* R = *(which_queue->head);//remove the head pointer
```

```
(which_queue->head)++;         // Remember, this points
(which_queue->cells_used)--; //decrement the cellsused
return R;//return the head pointer
}

void printQueue(Queue *which_queue) {
int i;  //loop counter
//loop to print the queue
for (i = 0; i<which_queue->cells_used;i++){
  //pointer to a float
  float* tempfloat;
  //This will only print the queue if it has floats
  tempfloat=(float*)malloc(sizeof(float));
  //dequeue one from queue
  tempfloat=(float *) dequeue(which_queue);
  //print the value
  printf("Queue Entry %d: %f", i, *tempfloat);
  //reenqueue
  enqueue(which_queue, (void*)tempfloat);
}
return;
}

void* peek(Queue *which_queue){
//check if empty
if ((which_queue->cells_used) <= 0) {
  which_queue->cells_used = 0; // insurance
  return NULL;  // Queue empty
}
void* ret = *(which_queue->head);
return ret;
}

void* peekTail(Queue *which_queue){
//check if empty
if ((which_queue->cells_used) <= 0) {
  which_queue->cells_used = 0; // Fix
  return NULL;  // Queue empty
}
//check to make sure we will not look before the beg of the queue
else if((which_queue->tail) == (which_queue->beg)){
  void* ret = *(which_queue->end);
  return ret;
}
//tail peek
else{
  (which_queue->tail)--;
  void* ret = *(which_queue->tail);
  (which_queue->tail)++;
  return ret;
}
}

int isFull(Queue *which_queue){
//compare max cells to cells used
if(which_queue->cells_used>=which_queue->max_cells){
  //return true
  return 1;
}
else{
  //return false
  return 0;
}
}



#ifdef __cplusplus
 extern "C" {
 #endif
```

```
#ifndef TOOL_H
#define TOOL_H

/** Struct to define a tool and related functions
 */
typedef struct tool {
  char *toolName;  //!The name of the tool
  float maxAllowableForce; //!The maximum allowable cutting force for the tool
  unsigned short number; //for easy ID
  }Tool;
```

Tool

```
/** \brief Creates a tool by allocating mem for the struct and its members and initializes it.
  * \param t- (float) the Controller gain of the new Controller
  * \param mat - (float) The controller signal
  * \param maxForce - (float) Gain used when adjusting the feedrate
  * \param refForce -(float) The previous signal sent by the controller
  * \return - (Controller *) returns a pointer to the newly allocated Controller
  */
Tool* create_Tool(char *name, float maxForce, unsigned short num);

#endif

#ifdef __cplusplus
 }
 #endif



#include <stdlib.h>
#include <string.h>
#include "tool.h"

Tool* create_Tool(char* name, float maxForce, unsigned short num){
    //make a pointer, allocate memory for the Tool
  Tool* new_tool;
  new_tool = (Tool *) malloc(sizeof(Tool));
  //set the properties
  new_tool->toolName = name;
  new_tool->maxAllowableForce = maxForce;
  new_tool->number = num;
  //return
  return new_tool;
}
```

Qt UI file

```
/********************************************************************************
/** ui.h extension file, included from the uic-generated form implementation.
*********************************************************************************/
//length of queues and arrays
#define dataSize 500
//timestep of display and control (in seconds)
#define timeStep .1
res
//neccessary imports
#include "queue.h"
#include "fileIO.h"
#include "Controller.h"
#include "DataCollector.h"
#include "tool.h"
#include "NCCommunicator.h"
#include "MemberFunction.h"
#include "mFD.h"
#include "FuzzyController.h"
#include "cuttingoperation.h"
#include "ParamTable.h"

#include <qstring.h>
#include <qwidget.h>
```

```
#include <qapplication.h>
#include <qlayout.h>
#include <qwt_plot.h>
#include <qwt_plot_canvas.h>
#include <qwt_scale.h>
#include <qwt_scldraw.h>
#include <qwt_math.h>
#include <qvalidator.h>
#include <time.h>
#include <string.h>
#include <qframe.h>
#include <qpainter.h>
#include <qcolor.h>

/** Create a queue by allocating a queue structure, initializing it,
 *  and allocating memory to hold the queue entries.
 * @param max_cells Maximum entries in the queue
 * @return Pointer to newly-allocated Queue structure, NULL if error.
 */
void Form1::init()
{
  //Alarm/Alarm Frame initialization
  alarmFlag = 0;
  alarmCounter = 0;
  alarmFrame->hide();

  //gui stuff
  controllerTypes->hide();
  synchPIDFrame->hide();
  asynchPIDFrame->hide();
  fuzzyFrame->hide();

  //init the controller
  theController = create_Controller();
  //Tools and Operations Inits
  newToolFrame->hide();
  toolIndex = 0;
  maxToolsReachedLabel->hide();
  newOperationFrame->hide();
  operationIndex = 0;
  maxOperationsReachedLabel->hide();

  //Param table init
  the_Ptable = create_paramTable();

  //this will be used for the control and data saving speed
  dataCounterTop =(int) (timeStep*1000)/10;
  dataCounter = 0;

  //create queues that will be used for graphs
  spindleQueue = create_queue(dataSize);
  forceQueue = create_queue(dataSize);
  feedQueue = create_queue(dataSize);

  //create the controller and the datacollector
  //theController = init_Controller();
  theDataCollector = createDataCollector(dataSize, 50, 1);

  //create the NcCommunicator
  the_NcCommunicator = create_NcCommunicator();

  //MAKE THE FUZZY CONTROLLER
  memberFunction* mf1;
  memberFunction* mf2;
  memberFunction* mf3;
  memberFunction* mf4;
  memberFunction* mf5;
  mFD* mFD1;
  mFD* mFD2;
  mFD* mFD3;
  mFD* mFD4;
```

```
//input error
mf1=createMemberFunction(-130,-70,-100);
mf2=createMemberFunction(-80,-20,-50);
mf3=createMemberFunction(-30,30,0);
mf4=createMemberFunction(20,80,50);
mf5=createMemberFunction(70,130,100);
mFD1 = createMFD(0,mf1,mf2,mf3,mf4,mf5);
//input delta error
mf1=createMemberFunction(-125,-75,-100);
mf2=createMemberFunction(-75,-25,-50);
mf3=createMemberFunction(-35,35,0);
mf4=createMemberFunction(25,75,50);
mf5=createMemberFunction(75,125,100);
mFD2 = createMFD(0,mf1,mf2,mf3,mf4,mf5);
//output spinlde
mf1=createMemberFunction(.15,.51,.33);
mf2=createMemberFunction(.48,.84,.66);
mf3=createMemberFunction(.75,1.25,1);
mf4=createMemberFunction(1.15,1.51,1.33);
mf5=createMemberFunction(1.48,1.84,1.66);
mFD3 = createMFD(1,mf1,mf2,mf3,mf4,mf5);
  //output feed
mf1=createMemberFunction(.15,.51,.33);
mf2=createMemberFunction(.48,.84,.66);
mf3=createMemberFunction(.82,1.18,1);
mf4=createMemberFunction(1.15,1.51,1.33);
mf5=createMemberFunction(1.48,1.84,1.66);
mFD4 = createMFD(1,mf1,mf2,mf3,mf4,mf5);

//timerCounter = (timestep*1000)
//the_fuzzyController = createfuzzyController(mFD1, mFD2, mFD3, mFD4);

//create the time axis that will be used in all the plots
//600 tenths of a second = 1 minute of data displayed
//also fill dummy array that will be used for plotting
for (int i = 0; i< dataSize; i++)
  {
      t[i] = -timeStep * double(i);     // time axis
      y[i] = 0;                        // dummy array
      force[i] = 0;
      feed[i] = 0;
      spindle[i] = 0;
  }

//set up the force plot and curve
forcePlot->setTitle("Cutting Force");
forcePlot->setAxisTitle(QwtPlot::xBottom, "Time(s)");
forcePlot->setAxisTitle(QwtPlot::yLeft, "Newtons");
long forceCurve = forcePlot->insertCurve("Force");
forcePlot->setCurvePen(forceCurve, QPen(Qt::red));
forcePlot->setCurveRawData(forceCurve, t, force, dataSize);
forcePlot->replot();
forcePlot->show();

//set up the spindle plot and curve
spindlePlot->setTitle("Spindle Speed");
spindlePlot->setAxisTitle(QwtPlot::xBottom, "Time(s)");
spindlePlot->setAxisTitle(QwtPlot::yLeft, "RPM");
long spindleCurve = spindlePlot->insertCurve("Spindle");
spindlePlot->setCurvePen(spindleCurve, QPen(Qt::blue));
spindlePlot->setCurveRawData(spindleCurve, t, spindle, dataSize);
spindlePlot->replot();
spindlePlot->show();

//set up the feed plot and curve
feedPlot->setTitle("Feed Rate");
feedPlot->setAxisTitle(QwtPlot::xBottom, "Time(s)");
feedPlot->setAxisTitle(QwtPlot::yLeft, "mm/min");
long feedCurve = feedPlot->insertCurve("Feed");
feedPlot->setCurvePen(feedCurve, QPen(Qt::green));
feedPlot->setCurveRawData(feedCurve, t, feed, dataSize);
```

```
  feedPlot->replot();
  feedPlot->show();

  //Tools

  //toolComboBox->insertItem("Tool 2",-1);
  //toolComboBox->insertItem("Tool 3",-1);
  //toolComboBox->insertItem("Tool 4",-1);

  //Init the array of tools
  //This would have to be done with a database if a lot of tools were used
  //toolArray[0] =

  currTool = create_Tool( "Carbide", 50000, 1);


}

//write the dataCollector's queues to text file
void Form1::saveData()
{
   //check to make sure the queues arent empty
    if(forceQueue->cells_used>0){
       time_t now;
       char the_date[50];
       the_date[0] = '\0';
       now = time(NULL);
       if(now !=-1)
       {
        strftime(the_date, 50, "%c.txt", localtime(&now));
       }
       FILE *file = fopen(the_date , "w");
       printf("OPENED THE FILE\n");


       writeAllTheQueuesFormatted(file,forceQueue, feedQueue, spindleQueue, theDataCollector-
>xForceQueue, theDataCollector->yForceQueue, theDataCollector->zForceQueue, timeStep);
   }

}


//set controller parameters
void Form1::setParams()
{/*
   the_fuzzyController->idealSpindleSpeed = spindleEdit->text().toFloat();
   //the_fuzzyController->currentSpindleSpeed = the_fuzzyController->idealSpindleSpeed;
   the_fuzzyController->idealFeedRate = feedEdit->text().toFloat();
   //the_fuzzyController->currentFeedRate = the_fuzzyController->idealFeedRate;
   the_fuzzyController->idealCuttingForce = forceEdit->text().toFloat();
    /*
   theController->desiredSpindleSpeed = spindleEdit->text().toFloat();
   theController->currFeedRate = feedEdit->text().toFloat();
   theController->desiredCuttingForce = forceEdit->text().toFloat();
   theController->controllerGain = .6;
   theController->feedRateGain = .5;
   theController->previousCommandSignal = 0;*/
   /*********************************************
     get the current tool index from the tool index QComboBox
     theController->setTool
     *********************************************/

}

//clears the dataCollector's queues
//clears the form's queues
//sets control parameters to 0
//clears curves
void Form1::reset()
{
```

```
    //reset the force, feed, and spindle arraysex
   for (int i = 0; i< dataSize; i++)
    {
        t[i] = -0.1 * double(i);     // time axis
        y[i] = 0;                       // dummy array
        force[i] = 0;
        feed[i] = 0;
        spindle[i] = 0;
    }

   //if there is a controller, reset the prev error and sum error

   clearOldData(theController);


   //reset the controller parameters
   //->desiredSpindleSpeed = 0;
   //theController->currFeedRate = 0;
   //theController->desiredCuttingForce = 0;

   //the_fuzzyController->idealSpindleSpeed = 0;
   //the_fuzzyController->idealFeedRate = 0;
   //the_fuzzyController->idealCuttingForce = 0;
   //the_fuzzyController->currentFeedRate = 1;
   //the_fuzzyController->currentSpindleSpeed = 1;

   //clear the queues
   for (int i = 0; i< dataSize; i++)
    {
       free(dequeue(forceQueue));
       free(dequeue(feedQueue));
       free(dequeue(spindleQueue));
       free(dequeue(theDataCollector->xForceQueue));
       free(dequeue(theDataCollector->yForceQueue));
       free(dequeue(theDataCollector->zForceQueue));
    }
   return;
}

//stops the timer events
void Form1::stop()
{
printf("Stopping Timer\n");
  killTimer(tehtimer);
}

//initialize the timer!
void Form1::start()
{
    //currTool=

     printf("Starting Timer\n");
     tehtimer = startTimer((int)(10));

}

//take in data, impliment control algorithm, communicate with NC
//plots
void Form1::timerEvent(QTimerEvent *)
{
    //flash the alarm if it is on
if(alarmFlag==1){
        alarmCounter++;
               alarmFrame->show();
    if(alarmCounter>=20){
        if(alarmCounter>=40){
            alarmCounter = 0;
            //change to red
            alarmFrame->setPaletteBackgroundColor(Qt::red);
            alarmFrame->repaint(true);
            alarmLabel2->setPaletteBackgroundColor(Qt::red);
```

```
            alarmLabel2->repaint(true);

        }
        else{
            alarmFrame->setPaletteBackgroundColor(Qt::black);
            alarmFrame->repaint(true);
            alarmLabel2->setPaletteBackgroundColor(Qt::black);
            alarmLabel2->repaint(true);
        }
    }
}

//Alarm wasnt currently acitvated, so proceed

    else{
    //if too high of force!  Alarm!!!!
    if(sqrt(getCurrForceSquared(theDataCollector))>=currTool->maxAllowableForce){
               //ALARM
        //stop the timer
        //killTimer(tehtimer);
        //perform the alarmRoutine (aka spindle off, back off the piece)
        alarmGUI();
        //now display the alarm in GUI
        //alarmFrame->show();
        //alarmFrame->showFullScreen();
    }
    //alarm not active and the force is acceptable, so store, gui, and control
    else{
        //because of the 10ms alarm, 100ms control
        dataCounter++;
        if(dataCounter >= dataCounterTop){
            //get the ADC force values
    getAndStoreForces(theDataCollector);
   // printf("GOT AND STORED FORCES\n");


    //this might not be right, but calculate the cutting force(adam method)
    currForce = forceFloat(theDataCollector);
    currForce = sqrt(currForce);
    //printf(" Current Force: %f\n", currForce);

    //Control
    multiControl(theController, currForce, .1);

    //update the forceQueue
    float* fp1;
    fp1 = (float *) malloc(sizeof(float));
    *fp1 = currForce;
    forceEnqueue(forceQueue,(void *)fp1);

    forceShow->setText( QString::number( currForce, 'f', 3 ) );
    qwtShiftArray(force, dataSize, 1);
    force[0] = currForce;
    forcePlot->replot();


    //update spindleSpeed and store in queue
    float* fp2;
    fp2 = (float *) malloc(sizeof(float));
    *fp2 = theController->currSpindleSpeed;
     //*fp2 = spindleControlFunction(theController, currForce);
     forceEnqueue(spindleQueue,(void *)fp2);
     spindleShow->setText( QString::number( *fp2, 'f', 3 ) );
    qwtShiftArray(spindle, dataSize, 1);
    spindle[0] = *fp2;
    spindlePlot->replot();

    //update spindleSpeed and store in queue
    float* fp3;
    fp3 = (float *) malloc(sizeof(float));
    *fp3 = theController->currFeedRate;
```

```
     //*fp3 = feedControlFunction(theController, currForce);
     forceEnqueue(feedQueue,(void *)fp3);
     feedShow->setText( QString::number( *fp3, 'f', 3 ) );
    qwtShiftArray(feed, dataSize, 1);
    feed[0] = *fp3;
    feedPlot->replot();

    //CONTROL


     //free the memory
    //free(fp1);

    ///float a[600];


    //make the
    /*
    static int zeta = 0;

    if( zeta == 100) zeta = 0;
    qwtShiftArray(y, 600, 1);
    y[0] = zeta;
    forcePlot->replot();
    zeta++;
  */
            dataCounter=0;
        }

    }
}
}


void Form1::test()
{

}


void Form1::unAlarmGUI()
{
    //turn off alarmflag

    //hide the alarm frame
    alarmFrame->hide();
    alarmFlag = 0;
    stop();
    reset();


}




void Form1::alarmGUI()
{
    //killTimer(tehtimer);
    //perform the alarmRoutine (aka spindle off, back off the piece)
    alarmRoutine(the_NcCommunicator);
    //now display the alarm in GUI
    alarmFrame->raise();
    alarmFrame->show();
    //already a timer in use, so this works well
    alarmFlag=1;
    //save the Data, to provide insight as to what went wrong
    saveData();
}
```

```
void Form1::popNewToolFrame()
{
    cancelNewOperation();
    //newToolFrame->setIsTopLevel(true);
    //newToolFrame->resize();
    newToolFrame->raise();
    newToolFrame->show();
    newToolFrame->repaint();
    disableBackground();

}


void Form1::popNewControllerSelect()
{
    //need to ensure that both a tool and an operation are available
    if(toolComboBox->count()<=0){
        //TODO pop up a warning that a tool must be selected
        printf("A tool must be selected\n");
    }
    else if(opComboBox->count()<=0){
        //TODO pop up a warning that an operation must be selected
        printf("an op must be selected\n");
    }
    //need to ensure that for the specified combo, a Controller doesnt already exist
    else if(confirmEntry(the_Ptable, toolComboBox->currentItem(), opComboBox->currentItem())==0){
        controllerTypes->raise();
        controllerTypes->show();
        controllerTypes->repaint();
        disableBackground();
    }
    //the combo already existed
    else{
        //TODO pop a window that a Controller has already been made
        printf("A controller already exists\n");
    }
}

void Form1::popNewOperationFrame()
{
    cancelNewTool();
    //opComboBox->setEnabled(false);
    //opComboBox->clearFocus();
  //newToolFrame->setIsPopup(true);
  newOperationFrame->raise();
  newOperationFrame->show();
  newOperationFrame->repaint();
  disableBackground();
}



void Form1::cancelNewTool()
{
    toolNameEdit->clear();
    maxForceEdit->clear();
    newToolFrame->hide();
    //toolComboBox->insertItem("Tool 1",-1);
    enableBackground();
}

void Form1::cancelNewOperation()
{
    operationNameEdit->clear();
    newOperationFrame->hide();
    enableBackground();
}

void Form1::saveTool()
{
QString str1 = toolNameEdit->text();
```

```
// copy QString to char*
char* cstr;
std::string fname;
fname = str1.ascii();
cstr = new char [fname.size()+1];
strcpy( cstr, fname.c_str() );
Tool* aNewTool = create_Tool(cstr,maxForceEdit->text().toFloat(), toolIndex);
toolArray[toolIndex] = aNewTool;

//add tool to tool array
    //Tool* newTool=create_Tool(toolNameEdit->text(),maxForceEdit->text().toFloat(),toolIndex);
    //toolArray[toolIndex]=newTool;
    toolIndex++;
    //if too many tools, then say it
    if(toolIndex>=25){
        maxToolsReachedLabel->show();
        newToolButton->setEnabled(false);
    }
     //newToolFrame->setIsPopup(false);
     toolComboBox->insertItem(aNewTool->toolName,-1);
  cancelNewTool();
  changeTool();
 //setEnabled(true);
}

void Form1::saveOperation()
{
QString str1 = operationNameEdit->text();
// copy QString to char*
char* cstr;
std::string fname;
fname = str1.ascii();
cstr = new char [fname.size()+1];
strcpy( cstr, fname.c_str() );
cuttingOperation* aNewOperation = create_cuttingOperation(cstr, operationIndex);
operationArray[operationIndex] = aNewOperation;

    operationIndex++;
    //if too many tools, then say it
    if(operationIndex>=25){
        maxOperationsReachedLabel->show();
        newOperationButton->setEnabled(false);
    }
     //newToolFrame->setIsPopup(false);
    opComboBox->insertItem(aNewOperation->operationName,-1);

  cancelNewOperation();
  changeOperation();
 //setEnabled(true);
}

void Form1::popSynchPIDFrame()
{
  controllerTypes->hide();
  synchPIDFrame->raise();
  synchPIDFrame->show();
  synchPIDFrame->repaint();
}

void Form1::saveSynchPID()
{
    // make a new controller and pointer to the controller
    Controller *newCont = create_Controller();
    //init the controller as a SynchPID with inputted params
    init_as_synchPID(newCont,
    newRecForceEdit->text().toFloat(),
    newSynchfKpEdit->text().toFloat(),
    newSynchfKdEdit->text().toFloat(),
    newSynchfKiEdit->text().toFloat(),
    newSynchsKpEdit->text().toFloat(),
    newSynchsKdEdit->text().toFloat(),
```

```
    newSynchsKiEdit->text().toFloat()
    );
    //now save the controller to the_Ptable
        addSet(the_Ptable,newCont,toolComboBox->currentItem(),opComboBox->currentItem());
        closeSynchPID();
        clearOldData(theController);
        theController=newCont;
        //enable the start button
        startButton->setEnabled(true);

}

void Form1::closeSynchPID()
{
    newSynchfKpEdit->clear();
    newSynchfKdEdit->clear();
    newSynchfKiEdit->clear();
    newSynchsKpEdit->clear();
    newSynchsKdEdit->clear();
    newSynchsKiEdit->clear();
    newRecForceEdit->clear();
    synchPIDFrame->hide();
    enableBackground();
}

void Form1::popAsynchPIDFrame()
{
  controllerTypes->hide();
  asynchPIDFrame->raise();
  asynchPIDFrame->show();
  asynchPIDFrame->repaint();
}

void Form1::saveAsynchPID()
{
    // make a new controller and pointer to the controller
    Controller *newCont = create_Controller();
    //init the controller as an AsynchPID with inputted params
    init_as_asynchPID(newCont,
    newAsynchRecForceEdit->text().toFloat(),
    newAsynchfKpEdit->text().toFloat(),
    newAsynchfKdEdit->text().toFloat(),
    newAsynchfKiEdit->text().toFloat(),
    newAsynchsKpEdit->text().toFloat(),
    newAsynchsKdEdit->text().toFloat(),
    newAsynchsKiEdit->text().toFloat(),
    AsynchCutoffForce->text().toFloat()
    );
    //now save the controller to the_Ptable
        addSet(the_Ptable,newCont,toolComboBox->currentItem(),opComboBox->currentItem());
        clearOldData(theController);
        theController=newCont;
        closeAsynchPID();
        //enable the start button
        startButton->setEnabled(true);
}

void Form1::closeAsynchPID()
{
    newAsynchRecForceEdit->clear();
    newAsynchfKpEdit->clear();
    newAsynchfKdEdit->clear();
    newAsynchfKiEdit->clear();
    newAsynchsKpEdit->clear();
    newAsynchsKdEdit->clear();
    newAsynchsKiEdit->clear();
    AsynchCutoffForce->clear();
    asynchPIDFrame->hide();
    enableBackground();
}
```

```
void Form1::popFuzzyFrame()
{
  controllerTypes->hide();
  fuzzyFrame->raise();
  fuzzyFrame->show();
  fuzzyFrame->repaint();
}

void Form1::saveFuzzy()
{
    // make a new controller and pointer to the controller
    Controller *newCont = create_Controller();
    //init the controller as an AsynchPID with inputted params
    init_as_Fuzzy(newCont,
    fuzzyENegEdit->text().toFloat(),
    fuzzyEZeroEdit->text().toFloat(),
    fuzzyEPosEdit->text().toFloat(),
    fuzzyDENegEdit->text().toFloat(),
    fuzzyDEZeroEdit->text().toFloat(),
    fuzzyDEPosEdit->text().toFloat(),
    fuzzyRecFeedEdit->text().toFloat(),
    fuzzyRecSpindleEdit->text().toFloat()
    );
    //now save the controller to the_Ptable
        addSet(the_Ptable,newCont,toolComboBox->currentItem(),opComboBox->currentItem());
        closeFuzzy();
        clearOldData(theController);
        theController=newCont;
        //enable the start button
        startButton->setEnabled(true);
}

void Form1::closeFuzzy()
{
    fuzzyENegEdit->clear();
    fuzzyEZeroEdit->clear();
    fuzzyEPosEdit->clear();
    fuzzyDENegEdit->clear();
    fuzzyDEZeroEdit->clear();
    fuzzyDEPosEdit->clear();
    fuzzyRecFeedEdit->clear();
    fuzzyRecSpindleEdit->clear();
    fuzzyFrame->hide();
    enableBackground();
}

void Form1::disableBackground()
{
    textLabel7->setEnabled(false);
    feedShow->setEnabled(false);
    newtonLabel->setEnabled(false);
    mmMinLabel->setEnabled(false);
    pushButton8->setEnabled(false);
    spindleShow->setEnabled(false);
    rpmLabel->setEnabled(false);
    textLabel6->setEnabled(false);
    textLabel5->setEnabled(false);
    forceShow->setEnabled(false);
    feedPlot->setEnabled(false);
    forcePlot->setEnabled(false);
    spindlePlot->setEnabled(false);
    toolSelectionLabel->setEnabled(false);
    opselectLabel->setEnabled(false);
    buttonsTabs->setEnabled(false);
    opComboBox->setEnabled(false);
    toolComboBox->setEnabled(false);
    forcePlot->removeMarkers();
}

void Form1::enableBackground()
{
```

```
    textLabel7->setEnabled(true);
    feedShow->setEnabled(true);
    newtonLabel->setEnabled(true);
    mmMinLabel->setEnabled(true);
    pushButton8->setEnabled(true);
    spindleShow->setEnabled(true);
    rpmLabel->setEnabled(true);
    textLabel6->setEnabled(true);
    textLabel5->setEnabled(true);
    forceShow->setEnabled(true);
    feedPlot->setEnabled(true);
    forcePlot->setEnabled(true);
    spindlePlot->setEnabled(true);
    toolSelectionLabel->setEnabled(true);
    opselectLabel->setEnabled(true);
    buttonsTabs->setEnabled(true);
    opComboBox->setEnabled(true);
    toolComboBox->setEnabled(true);
}

//upon tool change, will check if the current tool/op combo has a controller
//if not, disable start button
void Form1::changeTool()
{
    //if(toolComboBox->count()>0){
        //sets the current tool to the corresponding one in the toolArray
    printf("LOLOLOLOLOLOLOLOL Changing the currTool");
    currTool=toolArray[toolComboBox->currentItem()];//}
    //make sure that there is a tool and an operation
    if(opComboBox->count()>0&&toolComboBox->count()>0){
        printf("both an op and tool are present");
        //if there is no controller, disable start button
        if(confirmEntry(the_Ptable, toolComboBox->currentItem(), opComboBox->currentItem())==0){
            startButton->setEnabled(false);
        }
        //else there is a controller, so enable start button
        else{
        startButton->setEnabled(true);
        //set theController to the one from the ParamTable
        clearOldData(theController);
        theController = getAtIndex(the_Ptable, toolComboBox->currentItem(), opComboBox-
>currentItem());
    }
    }
    else{
        startButton->setEnabled(false);
    }
}

void Form1::changeOperation()
{
    //make sure that there is a tool and an operation
    if(opComboBox->count()>0&&toolComboBox->count()>0){
        printf("both an op and tool are present");
        //if there is no controller, disable start button
        if(confirmEntry(the_Ptable, toolComboBox->currentItem(), opComboBox->currentItem())==0){
            startButton->setEnabled(false);
        }

        //else there is a controller, so enable the start button
        else{
        startButton->setEnabled(true);
        clearOldData(theController);
        //set theController to the one from the ParamTable
        theController = getAtIndex(the_Ptable, toolComboBox->currentItem(), opComboBox-
>currentItem());
    }

    }
    else{
        startButton->setEnabled(false);
```

```
    }
}
```

Main Program

```
#include <qapplication.h>
#include "form1.h"
#include "qwt_plot.h"

int main( int argc, char ** argv )
{
    QApplication a( argc, argv );
    Form1 w;
    w.show();
    a.connect( &a, SIGNAL( lastWindowClosed() ), &a, SLOT( quit() ) );
    return a.exec();
}
```