Using Machine Learning to Identify Machining Parameters in Computer Numerical Control

A Dissertation
Presented to
The Academic Faculty

by

Nicholas Greenfield

In Partial Fulfillment
of the Requirements for the Degree
Masters of Science in Mechanical Engineering

Georgia Institute of Technology
August, 2022

Using Machine Learning to Identify Machining Parameters in Computer Numerical Control

Approved by:

Dr. Amit Jariwala, Advisor
School of Mechanical Engineering
Georgia Institute of Technology

Dr. Steven Biegalski
School of Mechanical Engineering
Georgia Institute of Technology

Dr. Christopher Saldana
School of Mechanical Engineering
Georgia Institute of Technology

Date Approved: August 25, 2022

ACKNOWLEDGEMENTS

I would like to give thanks to everyone who assisted with this thesis. I would like to thank Dr. Jariwala for giving me this research opportunity and guiding me through the whole process and pointing me in the right direction if I needed assistance. I would also like to thank Dr. Biegalski and Dr. Saldana for being on my reading committee and helping improve my thesis.

I would also like to thank the undergraduate researchers Kevin Wu and Neel Shah for their work on this project. Kevin's knowledge on CNC milling and Neel's experience with machine learning were invaluable. Additionally, I would like to thank Ashley Andrews at the Montgomery Machining Mall for giving me access to the CNC mill and giving me important insights from a machinist's perspective.

"This material is based upon work supported by the Department of Energy / National Nuclear Security Administration under Award Number(s) DE-NA0003921." Disclaimer: "This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

AI Artificial Intelligence

ML Machine Learning

CNC Computer Numerical Control

CAD Computer Aided Design

KNN K-Nearest Neighbors

MLP Multi-layer Perception

F_t Tool Load

σ Ultimate tensile strength

A Cross-sectional area of an uncut chip

Z_c Number of teeth engaged with workpiece

E_f Engagement factor of workpiece

T_f Tool wear factor

MI Score Mutual information scores

CAM Computer-aided machinery

NC Numerical control

MRR Material removal rate

IoT Internet of things

ROC Receiver operator characteristic

AUC Area under curve

CWT Continuous wavelet transform

PCA Principal component analysis

SUMMARY

Computer Numerical Control (CNC) milling machines are vital pieces of equipment in modern machine shops that allow a facility to quickly and accurately produce components. The ability to autonomously perform real time identification of the cutting parameters can help the machine users. In a makerspace environment, it can act as a safety check that can alert users to a potentially unsafe cut. This built-in safety check can provide quality control and lower the barrier to entry by enabling amateur users to operate the complex manufacturing equipment (like the CNC machines) in makerspaces. Additionally, in a manufacturing plant, the operator can be notified if a toolpath is being performed outside the expected operating window. Machine learning has allowed researchers to analyze data and gain new insights from these machines that were previously impossible. Researchers have used advanced sensors and machine learning in manufacturing to gather the information that can provide insight into factors such as tool behavior or material properties. However, these approaches are limited to specific tool paths and require external sensors to be mounted to the machine. Real-world machining encompasses a wide range of different tool paths with varying machining parameters and external sensors add cost and complexity to a machine. It was demonstrated that applying machine learning strategies to data collected from built-in sensors on a CNC mill can be used to detect and identify characteristics of the cutting parameters being used for a range of tool paths. Spindle power and load sensor data were extracted and used to train different machine learning algorithms to estimate machining parameters.

CHAPTER 1 INTRODUCTION

Both convention and Computer Numerical Control (CNC) milling machines are a standard feature in modern machine shops. Historians debate who invented the first true modern milling machine. Some argue it was in 1783 by Samuel Rehe (Radzevich), others give credit to American firearms manufactures, like Eli Whitney, in the early 1800s (Woodbury). Either way, these machines have changed the way manufacturers can produce parts. Milling is the process of using a rotating cutting tool to remove material from a workpiece by moving the cutting tool into the workpiece (Brown & Sharpe). This can be done in different directions over several axes allowing for complex geometry to be produced with precise tolerances. With advancements in computers, CNC milling machines started to become prevalent in the manufacturing industry in the 1970s (Noble). Conventional milling required a highly skilled operator to manually move the cutting tool through the workpiece to produce the required part. CNC milling instead using a Computer Aided Design (CAD) program to produce a 3D model of the part. This model is then converted into a file that the CNC program can read. The CNC mill then can perform cutting operations automatically at speeds unachievable with manual milling.

Artificial Intelligence (AI) is when a computer can complete tasks only thought possible by humans. As with CNC, advances in computing have allowed for significant advances in AI technologies. Machine learning (ML) is a subset of AI that uses passive observations to create predictions. Some data streams are far too large and dense to be processed and understood by humans. However, ML allows for patters and trends to found within data that humans would never be able to notice.

1.1 Motivation for Study

The Manufacturing Industry is growing to be a 2 trillion-dollar business and a vital part of any developed nation's gross domestic product (Keller). With the loss of much of its manufacturing base, nowhere is it more important than the United States to develop systems to increase productivity. With industrial machines valued at or well above $100k on average, time really is money. The time spent making a part that is incorrect costs the company and the customer in tooling, labor, and material.

Manufacturers have been pushing towards a "born-certified" holy grail, i.e., to have a strong level of confidence in the consistency of a part when it comes off a manufacturing machine. Achievements such as an *in-situ* quality certification would mean greatly reduced quality assurance (QA) cost as well as faster turnaround times. The current trend is to accomplish reliable process certification and history tracking with the use of additional sensing modules such as accelerometers and microphones with cheap computers or micro controllers such as the Raspberry Pi and Arduino. While a wonderful idea, these addons then must be managed by a host of people, from the shop foreman overlooking the team's progress to the IT professionals monitoring network security on an ever-growing array of connected devices. An area that has been largely overlooked is the use of built-in sensors to verify that the operator has loaded the correct material with the proper parameters in order to prevent a costly crash or injury. Currently, these sensors are there to provide position and horsepower utilization information for the operator but have a potential to do much more.

Additionally, makerspaces are becoming more prevalent throughout the world (Lou). Makerspaces are "a place in which people with shared interests, especially in computer or technology, can gather to work on projects while sharing ideas, equipment, and knowledge"

(Oxford Languages). By their nature, makerspaces are not operated exclusively by professional users. This limits the availability of complex machinery such as CNC mills. The purpose of this project is to harness the capabilities of sensors that are already present within CNC milling machines to autonomously characterize the cutting toolpath and determine cutting parameters in real-time. CNC milling is the focus here, as it has a great prevalence in academic makerspaces and actual manufacturing environments alike. Machinery such as CNC milling machines that rely on programming through computer-aided machinery (CAM) are inherently challenging to use and have extensively relied on experiential learning and one-on-one instruction that places a significant burden of instruction on the trainer's time (Min Jou et al.). In a manufacturing setting, if a known toolpath's parameters can be accurately characterized, operators can be clued in by real time characterization if there is a problem with the machining operation. For example, if the incorrect material is in the machine, the tool has excessive wear, or the machining parameters are incorrectly entered, then the real time cutting parameter classification would differ from the expected value. Therefore, could lead to improved safety envelopes that can ease training time and determine faults in a machining process.

1.2 Organization of the Thesis

Chapter 1 introduces the background on what CNC milling is and motivation on why the ability to characterize cutting parameters is useful.

Chapter 2 outlines the research question and existing literature on the topic.

Chapter 3 outlines the methodology that was used to set up the experiments, collect data, process data, and train machine learning algorithms. Additionally, a preliminary experiment is preformed to validate the proposed methodology using a simple unidimensional cut with constant cutting parameters.

Chapter 4 is the first study where a ML model is trained and tested.

Chapter 5 adds to the first study by increasing the variability of the data to mimic real world conditions to further test the ML models.

Chapter 6 concludes the results, examines the validity of the hypothesis, and outlines potential future work into autonomous classification based on data from CNC milling machines.

CHAPTER 2 Literature Review

This chapter examines the current state of research in CNC machining regarding using different sensors and models to gain insight into the machining process. Additional, software solutions that assist users with setting machining parameters are examined for any gaps that could lead to errors or general inaccessibility to CNC machining. Finally, the research question and hypothesis are proposed to fill the gap with regard to using sensors and models to assist CNC users with machine operation.

2.1 Uses of Sensors and Machine Learning in CNC

The current state of the art in the machining subfield of manufacturing has many different focus areas from hybrid to multi-axis and tool deflection. The references at the end of this thesis illustrate that monitoring and process control is a very active research field in manufacturing. The field has grown in countless directions with specifically relevant excerpts from the works of authors (Denkena and Boujnah 2018), Brecher, Altintas, Cho, Ferguson and Kurihar. The works of the select group described in detail below cover the current state of the art in tool deflection monitoring methodologies, process monitoring, cutting load estimation, and sensorless force detection.

As Denkena posited in the paper on *Feeling machines for online detection and compensation of tool deflection in milling*, there are great benefits to understanding the forces and reactions affecting the machining operation. Understandings, such as the ones covered in his paper illustrate the cost and time saving benefits of using advanced sensing and control techniques to drive the quality of the part being processed. However, the research here focuses on the tooling behavior, and does not attempt to identify the machining parameters. The paper assumes a significant stiffness of the tool compared to the workpiece to accurately calibrate the material, which relies on a well-developed understanding of the material properties of the calibration sample. The online analysis focusing on the tool's deflection is based on a comparison to a simulated value. This is only possible via the knowledge of what material the data is being compared to. Denka et.al. did however utilize a connection to a Siemens SINUMERIK 840D controller to provide real time communication and feed rate adjustment from an industrial grade PC connection at the machine interface. Integrations such as the one mentioned here show promise for the potential of monitoring significant events and load sensing in real time.

Brecher et al. attempt to integrate the quality control process through process monitoring and trend control. They try to identify the production of scrap early on in the process to prevent the waste of machine time and tooling life, as well as to understand the long term success rate of the machine. The research here compared the actual machined surface dimensional accuracy and finish quality to what was predicted by the models. Further discussion covers the types of sensing mechanisms adequate for the virtual workpiece development. A virtual workpiece reflects that machining process in a virtual environment using a verity of methods. The methods considered include strain gauges, dynamometer, in-spindle sensing, and Numerical Control (NC) kernel data collection, among others. The NC kernel is what processes the program and controls the position of the machine. Strain gauges were ruled out as there is little change in stiffness locally from the machining forces that could be accurately transmitted in a synchronous manner for real time comparisons. Dynamometers, both in-spindle and plate-style, were ruled out for the fact that their usage inherently changes the system. The in-spindle style dynamometers increase the tool-to-spindle nose distance, which causes a loss in machinable area. It also results in unrealistic system responses when compared to a standard tool holding setup, due to the loss in rigidity and increased mass held by the spindle. In the case of the plate-style dynamometer, the measurement equipment itself interferes with the measure of the process forces, leading to unrealistic models of the fixturing environment. This is true especially with the inertial forces inherent in moving workpieces found in 5th axis machining. NC kernel monitoring was avoided as it would take extensive modeling efforts to produce an accurate surface finish model due to the friction and backlash effects in the system. Spindle sensing showed the most promise and was therefore the focus of the analysis methodology the paper later discusses. The interesting take away here was that the NC kernel data was able to provide acceleration and motor current data. However, it was not used as it would confound the model that was driving towards a tool deflection or surface finish prediction without significant mathematical investigation into the model itself. However, this does show promise to using NC kernel data for other applications that don't require mathematical modeling.

Altintas et.al. in the paper titled *End Milling Force Algorithms for CAD Systems* investigate the first principles behind the cutting force models in use today in order to elucidate new models for CAD systems modeling cutting forces. Models such as the ones covered in this paper are the backbone of current simulation software. The paper titled *Generalized dynamic model of metal cutting operations* continues down a similar vein with in-depth description of generalized cutting force and chip formation models (Altintas and Kilic). The paper continues in its discussion of chatter stability prediction in turning and milling, but the specifics therein are outside the direct scope of the literature review.

Denkena et.al in the paper *Material Identification Based on Machine-learning Algorithms for Hybrid Workpieces During Cylindrical Operations* presents a method of identifying materials being machined on CNC turning machines using a combination of onboard sensors, external sensors, and machine-learning algorithms. The study specifically tests this capability on friction welded hybrid material shafts and proposes the use of these machine-learning algorithms for adjusting live process parameters in order to achieve better surface finishes and longer tool life. Similarly in *In-situ Identification of Material Batches Using Machine Learning for Machining Operations*, the authors used torque data from a CNC lathe, tool condition information, and various machine learning algorithms to differentiate between different batches of a material. These studies are encouraging, as they demonstrate the use of machine learning algorithms in material classification based on signal data during a CNC machining process.

Cho et.al in *Tool Breakage Detection Using Support Vector Machine Learning in a Milling Process* use load and power data from a milling machine and support vector machine learning to detect tool breakage. Ferguson et.al in *A Data Processing Pipeline for Prediction of Milling Machine Tool Condition from Raw Sensor Data* use acceleration and audio data to predict tool condition during milling. These studies help support the idea that using sensor data from a milling machine and machine learning is a viable strategy for gaining insight to the CNC milling process.

Kurihara et.al. discusses a foundational process for the sensorless data collection of cutting forces in the paper *Cutting Force Control Applying Sensorless Cutting Force Monitoring Method*. By sensorless, the authors mean without a force sensor like a dynamometer directly measuring force. Instead, the paper brings up the methodology for the detection of disturbances from the effects on linear motion table drive loads based on current and encoder position. Kurihara and team were able to observe and quantify the disturbance forces on the machine-driven table. The disturbance signal comes from the difference in the axis drive power and the expected vs actual change in the absolute encoder reading. The analysis reliant on these equations was used for process optimization and cutting force control with a control feedback loop. This was then shown to be an accurate prediction and control algorithm with proper settings for both micro and high-speed machining tool paths incorporating an accurate model of the actual cutting forces on the tool from the non-contact observer. This further validates the ability to use sensor data to predict parameters of the cutting operation.

2.2 Software usage to make CNC more Accessible

There are several different software and tools that can be used to assist users set up CNC milling operations. However, as of the publication date of this paper the author has been unable to find any mainstream computer-aided machining program designed to model the complete milling process, including the machining parameters, material, tool, and toolpath and perform a real time comparison to the actual operation. Some work was done in the early 2000's with the integrations of computer-aided process planning for tooling and fixturing selection. (Zhao et al.) Unfortunately, the work seems to rely on the user to dictate the material and does not check the validity of any of the information input to the system. It is the lack of an autonomous check on the input data that is the gap this thesis aims to close. There is a propensity for human error in determining that the correct machining parameters, correct tool, and correct material are loaded and zeroed in the correct manner. Many software options on the market try to bridge this gap, such as G-Wizard by CNCookbook and VERICUT by CGTech, which both estimate tool forces and torque cutting loads. However, these still require someone to read the values or look up the table in accordance with manufacturer specifications. Additionally, it is still reliant on all information being entered correctly and has no ability to compare the predicted values to the real cutting process; therefore, no integrated solution has been mass marketed yet.

It is only since monitoring has gone online with Industry 4.0 (smart manufacturing) that non-tradesmen are taking the birds-eye view from the office. It would not be easy to tell which machine may be processing other materials from the office, especially when the machine is not visible. VERICUT and other software applications like it can estimate cutting forces from the Gcode input, yet there are little-to-no checks on the validity of that data when it comes to actual operation. This project therefore focuses on limiting human error by avoiding reliance on self-reported data as much as possible.

A system that would verify for the operator and floor supervisors that a machine is cutting with the correct parameters for the material and experiencing loads within a reasonable deviation would do much to minimize downtime. Currently downtime generally consists of replacing broken or worn tools, maintenance on overworked or damaged components, and routine upkeep, all of which contribute to lost revenue.

2.3 Research Questions & Hypothesis

This research aims to describe a process to collect and process data from a CNC milling machine to train machine learning algorithms to classify a 2D cutting operation based on machining parameters. The machining parameters being examined are the depth of cut, width of cut, and feed rate, which is the velocity the cutting tool moves through the material. The product of these parameters, equation 1, is known as the material removal rate (MRR), which gives the volume of material removed per a unit time. Additionally, potential applications of this process will be discussed.

$$\boldsymbol{MRR = Feed\ Rate * depth * width} \tag{1}$$

The ability to autonomously characterize cutting parameters in real time has the potential to aid the educator or trainer as CAM must be customized for the individual part, taking into account material and geometry. The student's CAM program must then be reviewed to help ensure the success and safety of the program. The process of checking students CAM files is time consuming, and the machine setup must be checked to make sure it matches the assumptions that went into making the CAM file, such as correct material and tooling. This research could lead to

CNC machining becoming more accessible, because the risk from operator error would be mitigated by leveraging onboard sensors coupled with machine learning to “double check” operator inputs in real-time. Not only would this increase in accessibility benefit industry, it could also help STEM education and outreach by enabling CNC machines to be more prevalent on academic campuses and public makerspaces. A decrease in the instructor workload may then directly facilitate faster onboardings to research, and a greater number of students engaged both in the academic and industrial workflows.

The development and verification of a methodology with minimal additional instrumentation that can determine the parameters of a machining operation has other great potentials as well. There are implications for improved Internet of Things (IoT) of manufacturing, job tracking, and the aforementioned education and training. In addition to lowering manufacturing costs by decreasing errors, having an *in-situ* method to characterize and control a machining process through sensor-based closed loop control algorithms will help ensure better quality parts and faster turnaround times.

Research Question: What sensor data from a CNC milling machine is adequate to characterize cutting parameters for a 2D toolpath?

Hypothesis: The built-in axis drive and spindle load sensors in a standard CNC Mill are sufficient to reliably classify the “aggressiveness” of the cutting parameters of a 2D toolpath regardless of the cut geometry and cut parameters (depth/width of cut and feed rate).

This chapter examined current research into how using sensors data and machine learning can give further insight into CNC manufacturing. Additionally, this chapter covered some limitations in how current software tries to reduce errors and mistakes in CNC operation and proposes a hypothesis that sensor data and machine learning can help fill these limitations.

CHAPTER 3 Methodology

This chapter will outline some base assumptions being made for this research, along with explanations on how the different experiments are set up and how the recorded data was processed and used to train ML algorithms. A flowchart showing the process used to predict aggressiveness from the spindle/load data is shown in Figure 1.

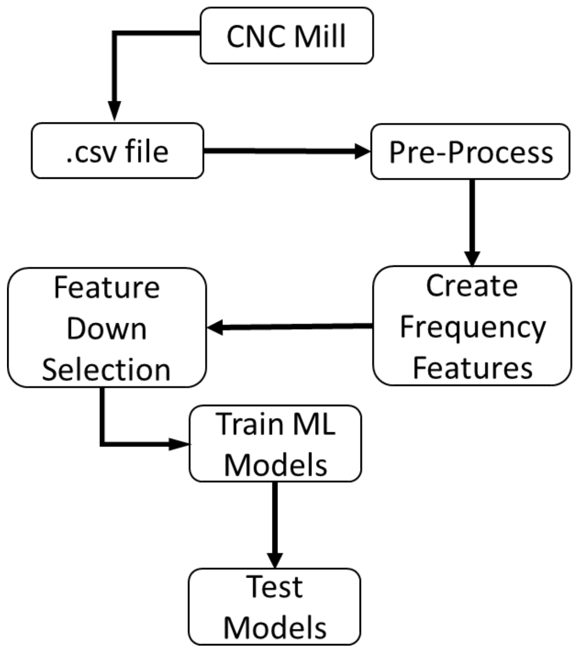


Figure 1 Flow chart for experiment process

3.1 Scope:

This research will focus on the following variation of parameters: changing width of cut, depth of cut, feed rate, and changing the toolpath strategies. There is an infinite combination of parameters and part geometries that can be used in CNC milling. To scope down the problem for this research, only 2D milling operations are examined. Also, tooling and workpiece material will be kept constant. Lastly, all experiments will be conducted on the same CNC milling machine.

3.2 Assumptions:

Several assumptions need to be made regarding the experiment and data obtained from this research. First, all materials tested are assumed to be isotropic, as it is known they generally behave in this manner unless specially altered. Secondly, tooling variation is minimal. All tools will be from the same known quality manufacturer, Accupro. All experiments were conducted with 0.25" diameter, 0.75" LOC, 2.5" OAL, four flute, solid carbide square end mill with AlTiN finish (Accupro part number: 12182467; MSC Part number: 01982420). Tool wear is not significant and does not significantly vary over the course of a part. Tool wear very well may become a factor over repeated testing of the same tool on subsequent parts; however, tooling will be replaced regularly to avoid any effects of wear. Non-isotropic materials or tooling with different wear states would produce different signals for equivalent parameters and cut geometries and would thus be an additional independent variable that is outside this scope of this research. Also, any variation among signals received from the different toolpaths must be characteristics of the path itself. That is, if two cuts are preformed that are identical except for the cutting parameters, any differences in the signals received are assumed to be because of the different cutting parameters.

3.3 General Study Plan Overview

A detailed explanation of the setup for each study can be found in subsequent sections of this thesis; however, the general experimental setup process for all the studies follows.

3.3.1 Test Piece Setup

Each study consists of a standardized and typical 2D toolpath seen in CNC machining created in Autodesk Fusion 360 CAM software. Each stock piece is a 4" long by 2.5" wide block of 6061 aluminum. Machining parameters such as spindle speed, depth of cut, width of cut, and feed-rate are set using G-Wizard: Machinist's Calculator. G-Wizard accounts for the tooling, material being cut, and machine limitations to provide a range of optimal cutting parameters. The exact process used to generate these values is proprietary; however, it is known that it estimates tool load based on the entered cutting parameters. Then, using information the user entered about the tool, it estimates tool deflection, which is used to generate the recommended feed rate for the desired "aggressiveness". Generally, tool load (F_t) is a function of ultimate tensile strength of the work material, cross-sectional area of an uncut chip (A), the number of teeth engaged with a workpiece (Zc), the engagement factor of the workpiece (E_f), and the tool wear factor (T_f) (Isakov). This information is either directly entered into Gwizard or recommended values are used based on the material, machine, and type of tool are used.

$$\boldsymbol{F_t = \sigma \times A \times Z_c \times E_f \times T_f} \tag{2}$$

The actual cutting parameters that affect the aggressiveness for a given machine/tool/material are depth of cut, width of cut, and feed rate, the product of which is material removal rate (MRR). Gwizard uses a 1% to 100% slide bar that allows users to change how aggressive/conservative the cutting parameters are. For the purpose of this thesis, we are classifying a conservative cut as 1%-33%, an optimal cut from 33%-66%, and an aggressive cut between 66% and 100%. However, 'aggressiveness' can be defined in any way based on the requirements for future work. For any given depth and width of cut, the effect of changing the slide bar is to adjust the feed rate.

3.3.2 Machine Setup

All studies were conducted on an EMCO E350 CNC milling machine with a Siemens 828D Controller. The test pieces were clamped into a vise on a pair of parallels and hammered into place with a rubber mallet. The end mills were placed into a collet with nominally 1" stick-out. The actual stick-out was manually measured using the machine. An edge-finding tool was used to determine the X and Y axis of the test piece. The Z axis was set to be the top face of the test piece. After each cut the Z axis had to be reset so that it remained on the top of the test piece.

3.3.3 Data Collection

While the machine is in operation, it is tracking all internal variables, including sensor data; however, it does not save any of them by default. A trace function can be used to select desired variables and save them to a USB drive as a .xml file after the experiment has finished. Trace function settings can be saved onto the USB drive and easily loaded back onto the machine for each experiment. The machine collects data at a rate of 166Hz. The data that were exported are spindle power and load for each axis as a percentage of max load. Documentation on how to set up a trace function and a list of pertinent variables that can be recorded can be found in Appendix A.

3.3.4 Data Pre-Processing

As previously stated, the data from the machine is a .xml file. This file was then read by a Python script (Appendix B.1) and saved as a .csv file to allow for easier data processing. However, these data still need additional processing before they can be used to train a ML model. At this stage the file was referred to as ‘dirty’. A MATLAB script was to further process this file (Appendix B.2) An example of what a ‘dirty’ data looks like for spindle power is shown in Figure 2 and will be used to explain the steps to obtain the necessary data. This data will be explained in greater detail in section 3.3.8. For now, all we need to know is that this graph can be broken up into five sections: start up (red), first pass (green), main passes (yellow), last pass (purple), and shutdown (not shown). The shutdown phase is not actually shown as data collection was stopped immediately after the last pass. If data collection were allowed to continue, shutdown would be the mirror of startup (large peak then flat line at zero). When the spindle starts up and shuts down there is a large spike in spindle power. This feature is not representative of anything pertinent and thus would only add noise to the model. The first and last passes are unique to this particular cutting path as the first pass was purposely run at a lower feed rate than the others. Also, because of the geometry of the block, the last pass has a smaller width of cut than the others. Depending on the objective of a particular experiment, these passes may not be suitable to include in the final file used with ML.

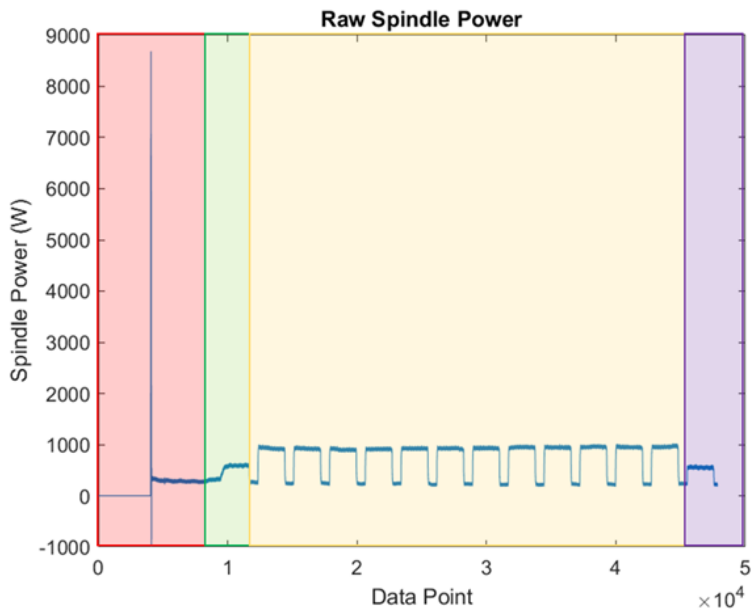


Figure 2: Spindle Power ‘Dirty’

After the sections of data that need to be removed are identified from the spindle power graph, they are removed along with the corresponding data points from the x, y, z load data. A new .csv file is generated for spindle power, x, y, and z load that only includes data from the desired section (yellow). This is repeated for each of the conservative, optimal, and aggressive files.

From here a Python script (Appendix B.3) takes all the previously made files and combines them into one .csv file. Additionally, the script also calculates the magnitude of the load forces and assigns the conservative, optimal, or aggressive label to the data for future ML algorithm training.

3.3.5 Feature Engineering

There are five time-domain features. Four are directly measured from the machine: spindle power and x, y, and z load; total load magnitude was then derived from the xyz loads. Continuous wavelet transform (CWT) was used to generate frequency features from the data (Appendix B.3.1). The complete frequency spectrum was used, resulting in an additional 125 frequency features for each of the five time-domain features. In total, this resulted in 630 different features for the ML algorithms to examine.

3.3.6 Feature Down Selection

Before training any ML algorithm, the 630 features need to be down-selected to reduce noise and increase the speed of the ML model. Pearson Correlation was used to remove features that are redundant to training. Pearson Correlation is a data reduction technique that is used in machine learning applications (Saidi). It is calculated as the normalized covariance between two variables. A Pearson Correlation of zero means there is no correlation, a value of 1 means the two features are completely correlated and thus including both features would not aid the ML algorithm (Nettleton). To perform this analysis, one of the features was taken and one at a time compared to each of the other features. If the Pearson Correlation was greater than 0.85 it was said to be redundant and removed. This was then repeated until every feature had been compared to one another.

Next, the mutual information scores (MI score) were derived for the remaining features. MI score indicated how much extra information can be obtained from one variable by examining another variable. Like Pearson Correlation, MI score is a powerful feature selection tool used in machine learning (Tourassi), (Guyon). A feature that produces extra information that is useful for classifying aggressiveness should be retained. Features with an MI score less than 0.15 were removed from the dataset.

3.3.7 Machine Learning Algorithm

Several standard supervised learning approaches were investigated for classifying aggressiveness of a milling toolpath. These include: Logistic Regression, K-Nearest Neighbors (KNN), Decision Trees, Random Forests, and Multi-layer Perception (MLP) Neural Network. All commonly used models with python libraries that have extensive documentation. Additionally, these models have been used in previous research regarding CNC machining operations (Lutz), (Cho).

Logistic regression is a statistical model that predicts an outcome based on response variables using log odds ratios (DiGangia). Logistic regression is similar to linear regression; however, the response data can be categorical and does not have to be continuous. This allows for better performance with data that are nonnormally distributed (DiGangia). K-Nearest Neighbors is a classification algorithm that predicts what group a data point will be in based on the data points around it (Joby). For example, assume there are two known groups of data, A and B, as well as an unknown data point. The algorithm will look at the labeled data closest to the unknown point and label the unknown to the group it is closest too.

Decision Trees and Random Forests are other classification algorithms that works by creating decision rules based on the training data. Somvanshi et.al. outline the general steps used to create the Decision Tree below.

1. Select attributes from decision nodes
2. Calculate information growth for each attribute
3. Choose attribute with largest information growth to be root node

4. Calculate branches of decision tree with information gain values of the node
5. Recursively build decision tree

Random Forest is then simply several randomly generated Decision Trees. Lastly the MLP Neural Network is an approximator that works for both linear and non-linear functions. It works by assigning the training data as the input layer of the neural network. An input function is then used to calculate the weighted sum of the input features. This is then passed to an activation function which produces an output (Menzies et.al.). This output is compared to the true value and the weights assigned to each node are updated to decrease error. Multiple layers and different activation functions can be used.

Each of these models have different hyperparameters that need to be optimized. Decision Trees and Random Forests have maximum depth, MLP has number of hidden layers and four different activation functions for the hidden layers, and KNN has the number of neighbors and the leaf size. These hyperparameters were optimized by iteratively changing the values and comparing the F1 scores as defined below.

$$\boldsymbol{F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}} \tag{3}$$

$$\boldsymbol{Precision = \frac{true\ positive}{true\ positive + false\ positive}} \tag{4}$$

$$\boldsymbol{Recall = \frac{true\ positive}{true\ positive + false\ negative}} \tag{5}$$

After optimizing hyperparameters, this model is finally judged by its performance classifying never-before-seen cuts as it would in a real-world scenario, not just held-out data from experiments on which it trained. For this evaluation, each model was trained using the full training set and tested on data from a testing set collected using the same common 2D milling operations, but with different part geometry. A general outline for this process is outlined below, the complete code can be found in Appendix B.3.

1. Import pre-processed data
    a. Label data and combine into single array
    b. Export to preform CWT with MATLAB
2. Import data with frequency features
3. Down select with Pearson Correlation and MI score
4. Optimize hyperparameters
    a. MLP classifier
    b. KNN
    c. Random Forest and Decision Tree
5. Train model with best hyperparameters
6. Test models
    a. For the bests model this was repeated for different feature sets, for example, comparison between the reduced feature set and just spindle power.

3.3.8 Preliminary Experiments/Methodology Validation

The focus of the preliminary experiment was to validate the above workflow and to gain insight on the physical interpretation of the data. A simple linear facing operation (Figure 3) with all cuts in the same direction, was conducted. All cuts were done with a 7000 RPM spindle speed, 0.25” depth of cut and a 0.15” stepover. The feed rates were 22.7 ipm for conservative, 44.8 ipm for optimal, and 67.0 ipm for aggressive. Each cut contains 17 passes. The first six and last six passes were done at the conservative and aggressive feed rate; the middle five passes were at the optimal feed rate. The very first and last pass were ignored, leaving five passes at each feed rate. No ML analysis was done as this was simply to gain insight to the measured data.

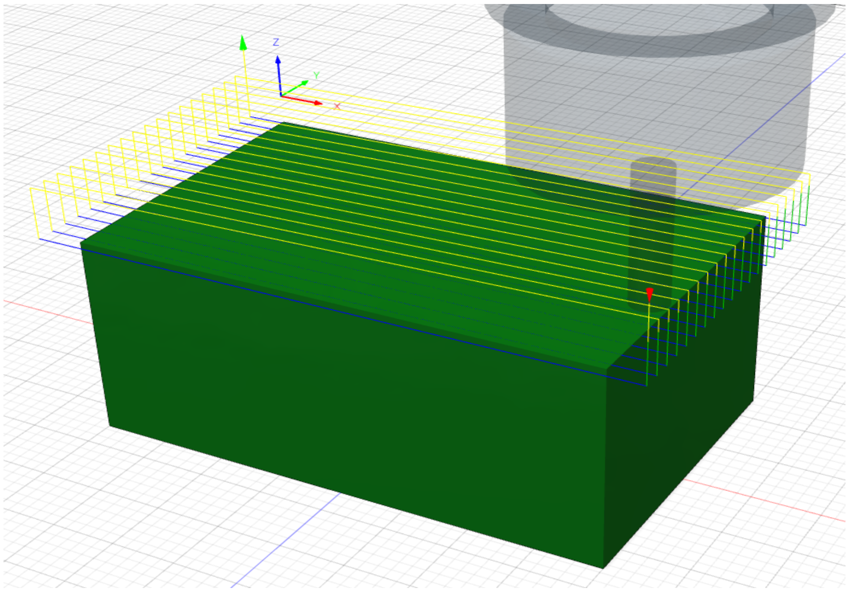


Figure 3: Toolpath for Preliminary Experiment

3.3.8.1 Preliminary Experiment Results and Discussion

Due to the very simple toolpath used in this study, different features can easily be differentiated by direct examination of the data. Figure 4 shows the spindle power and xyz axial load. Looking at the spindle power graph, it is clear that there was a steady state power level for when the tool is cutting (high values) and when it is not cutting (low values). As the feed rate increases, there is a noticeable change in the spindle power. This is much less apparent in the X and Z axis load data, where it remains relatively consistent. The Y axis load values, however, alternate from one pass to another. It is believed this is due to the direction that the cut is being performed. The cut was in the X direction, so for every pass the X location of the tool was starting and stopping in the same location. However, the Y location of the tool was at a different location for every pass.

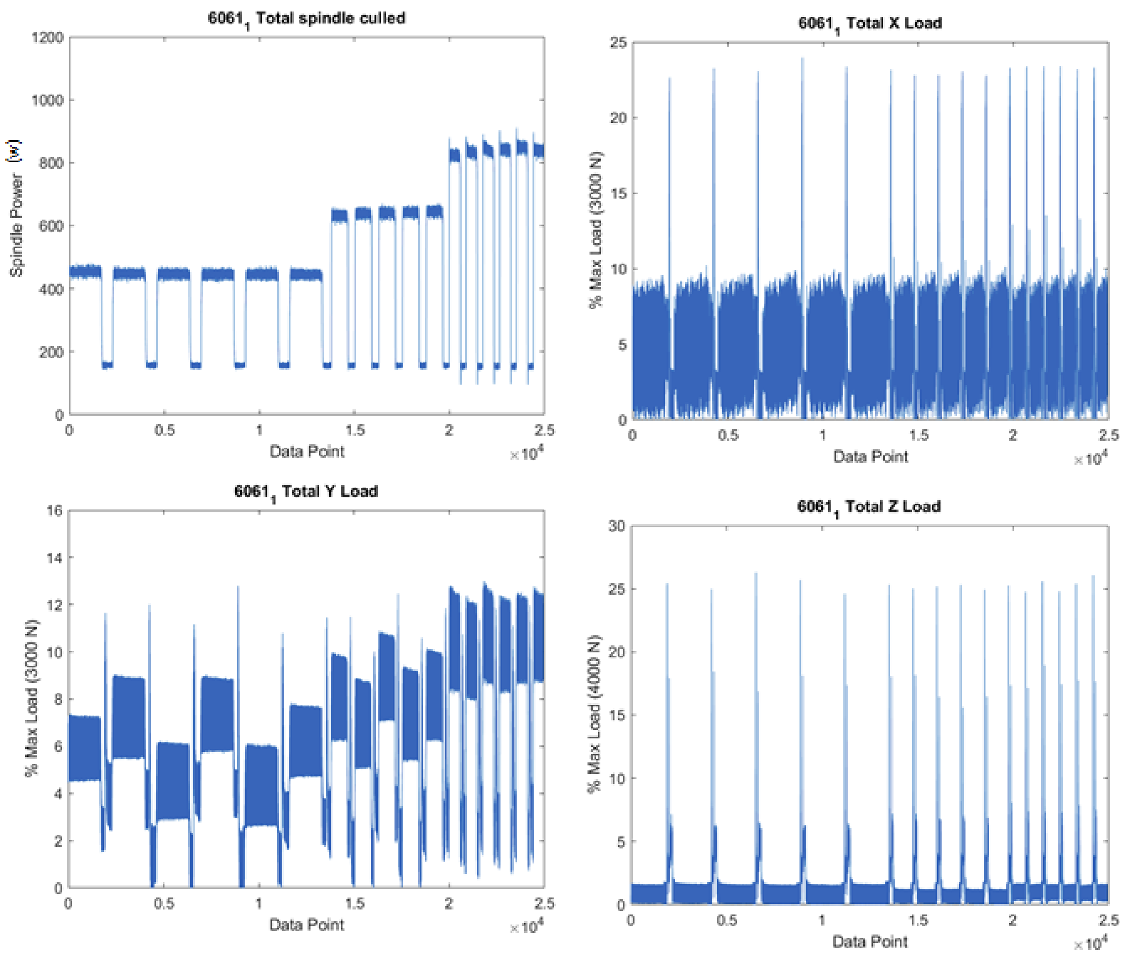


Figure 4: Raw Data from Preliminary Experiment while cutting in X direction. Increase in cut aggressiveness can be easily seen in upper left plot

By changing the direction of the cut along the Y direction, the X load data looks more like the original Y load and the new Y load is more consistent like the original X load (Figure 5). The spikes in the X, Y, and Z axis load are caused by the rapid acceleration and deceleration from the tool traversing over the part to return to the cut start point.

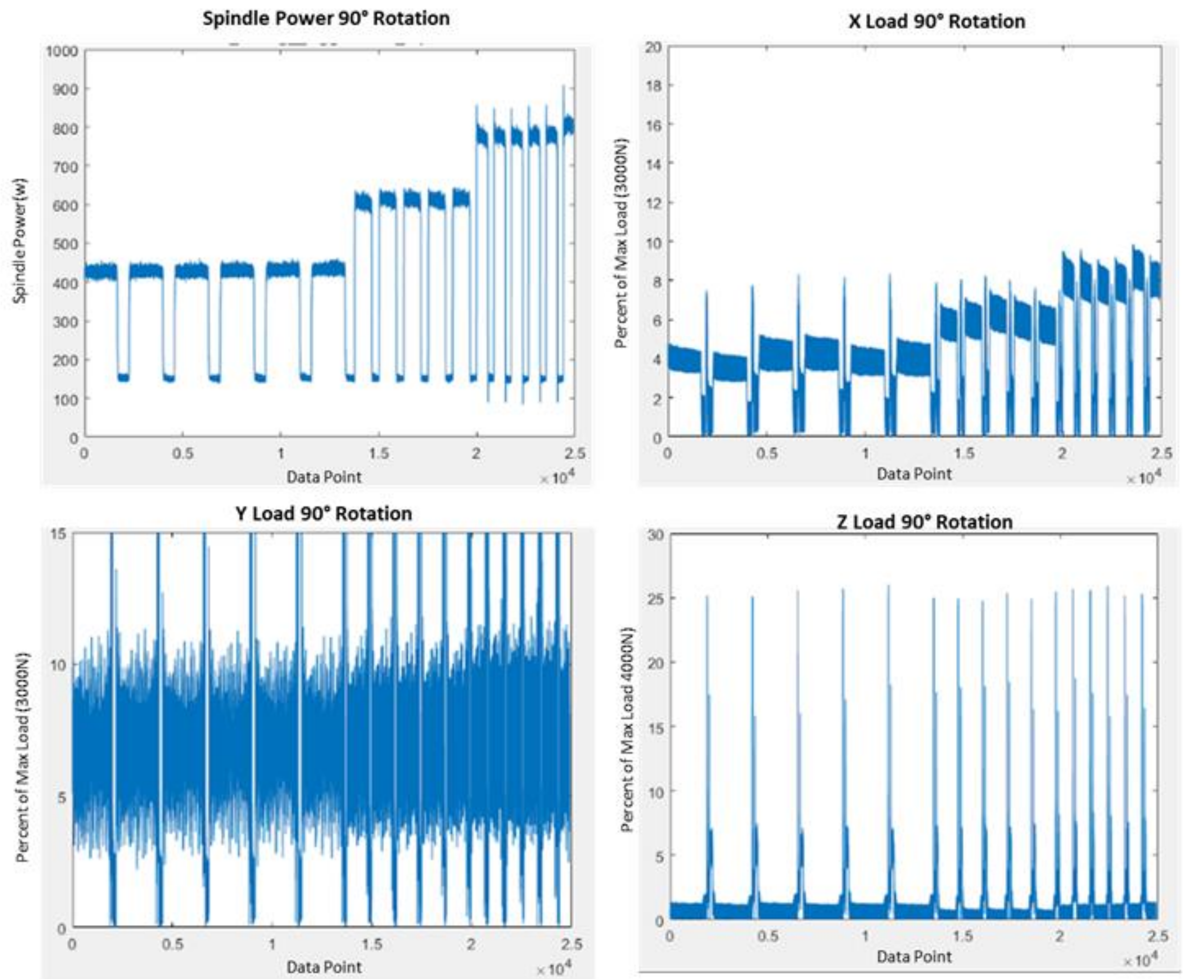


Figure 5: Data from Preliminary Experiment while cutting in Y direction. Note how Y load (bottom left) is more constant as opposed to figure 4 bottom left

It has been shown that spindle power and xyz load data can be successfully extracted from an EMCO E350 milling machine. This data can be visualized and is consistent with what should be expected. Subsequent chapters will train ML models with more complex geometries to classify aggressiveness.

CHAPTER 4 Study 1 – Complex Geometry

This chapter will analyze a complex geometry with a constant depth and width of cut to try to classify aggressiveness. Study 1 was composed of more complex toolpath that are meant to resemble a variety of real-world toolpaths such as pockets with different radii and linear operations in different directions. By varying the radii and direction throughout the cut, more variable data can be collected in a single operation.

4.1 Training Geometry

Study 1 used the same base parameters as the preliminary experiment: 7000 RPM spindle speed, 0.25” depth of cut and a 0.15” stepover. The geometry was created using Fusion 360’s Fit Point Spline feature that allows points to be added that will be connected with curves. This geometry has seven different curves that have varying radii. These curves are intentionally random as to avoid training the model to a specific type of operation. Dimensions for this geometry are in Appendix C.1. The toolpath was created using Fusion 360’s 2D adaptive clearing feature (Figure 6). To produce the inner pocket, the mill needed to ramp down into the part (pink swirl in Figure 6). This feature was easily identifiable from the spindle power in Figure 7. Data from the ramp down was removed as it was outside the scope of this research (2D toolpaths only).

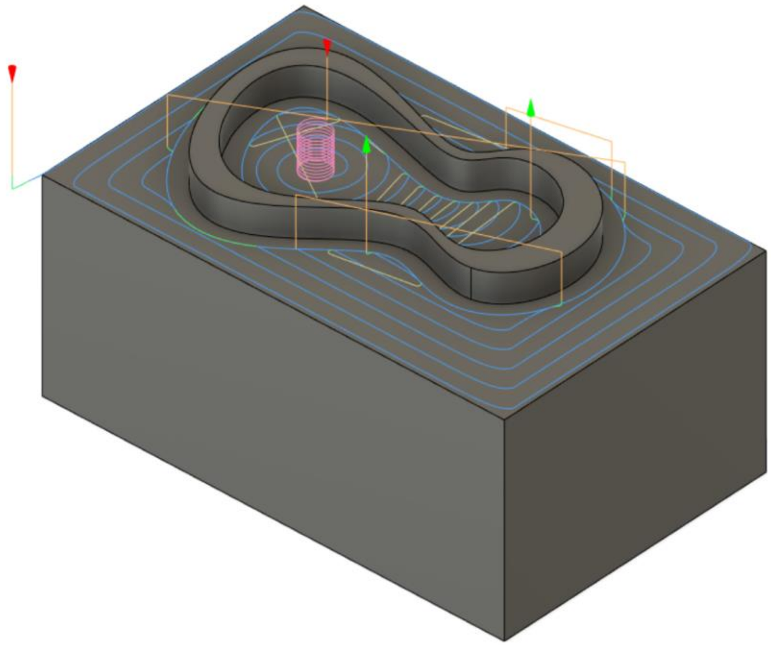


Figure 6: Study 1 Training Set Toolpath, 0.25” Depth of Cut, 0.15” Stepover. Pink spiral is ramp down

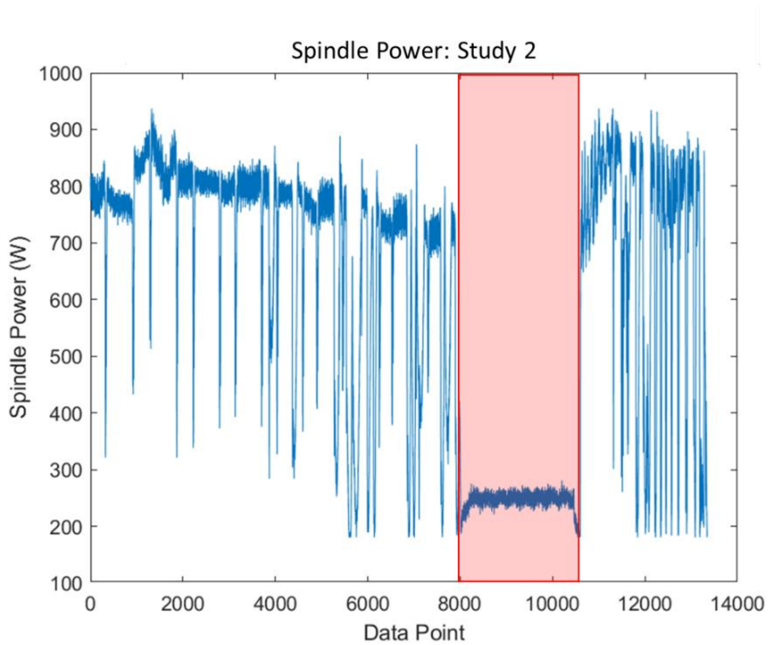


Figure 7: Spindle Power from Study 1. Red area is ramp down

4.2 Testing Geometry

In addition to the training set geometry, a different testing geometry was used to test the performance of the ML algorithms. This geometry (Figure 8) has the same parameters as the training set but has a different shape, and thus toolpath. This geometry still has seven different curves of varying radii but they have been shifted around by dragging the points that connect any two curves (Appendix C.2). This know geometry contains different radii that are cut in different directions/locations to increase variability between the training and testing set.

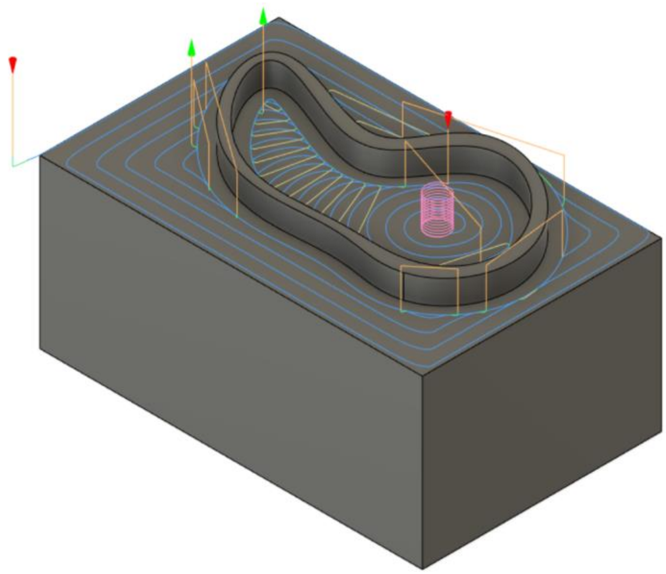


Figure 8: Study 1 Testing Set Toolpath, 0.25” Depth of Cut, 0.15” Stepover

4.3 Parameter Selection

As described in Section 3.3.1, these toolpaths were broken into three groups based on the aggressiveness of the cut: conservative, optimal, and aggressive. Within each cut aggressiveness group, the features were cut at varying feed rates to further increase variability. Feed rate was changed during the cut by using a dial on the EMCO E350 that would change the feed rate specified in the G-code on a range from 0% to 120%. By turning the dial during the cut, a range of feed rates can be collected in one cut. Table 1 summarizes the cutting parameters for all three cuts and shows the starting value the dial was set on to produce the desired feed rates as well as the feed rate from the G-code. From this starting value, the dial was turned to the full 120%.

Table 1 Study 1 Feed Rate Cutting Parameters

	Minimum Feed Rate (in/min)	Max Feed Rate (in/min)	Value in Gcode	Start Dial
Conservative	22.7	37.5	31.25	75%
Optimal	37.5	52.3	43.58	85%
Aggressive	52.3	67.1	55.92	95%

4.4 Study 1 Results and Discussion

Reducing features as described in section 3.3.6 resulted in 12 of the 630 features remaining for ML training. These features are shown in figure 9, which shows a comparison of their MI scores. Each axis on the figure is a different feature with each circle representing the MI score. Thus, the further from the center of the figure, the larger the MI score for that feature. It should be noted that the only time domain feature that remains is the spindle power, whereas all the frequency features remaining come from the load data, non from the spindle power. This is unlikely to be a coincidence. An in-depth investigation on why this is the case is outside the scope of this research, but it is hypothesized that the reasoning there are no spindle frequency features is that these are based on the frequency at which the controller drives the spindle. Thus, the spindle frequency values are not affected by the cutting process.

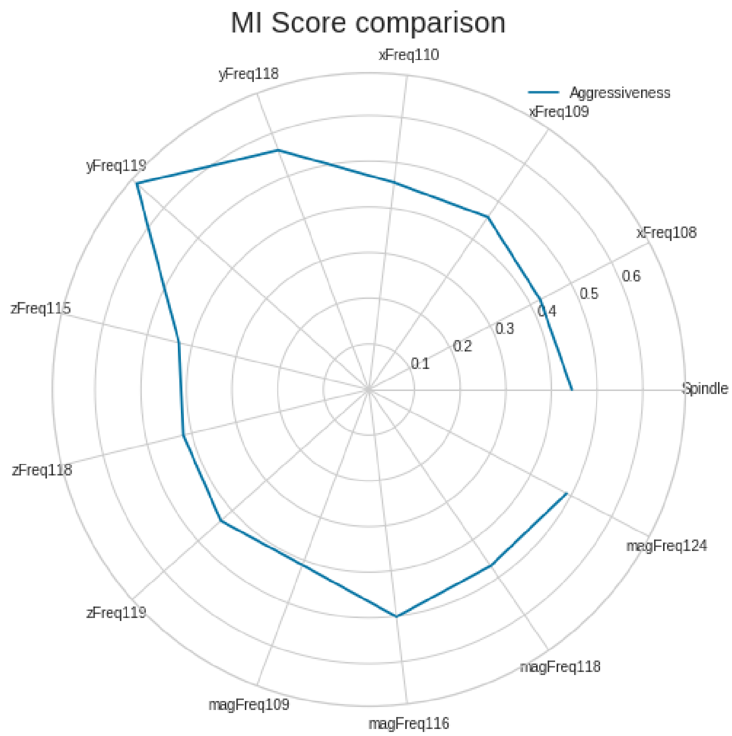


Figure 9: MI Scores for Remaining Features for Study 1. Individual spokes represent a feature. Radii magnitude is the MI score

Using the hyperparameter optimization method described in section 3.3.7 resulted in using a max iteration of 15,000 for Logistic Regression, three neighbors and a leaf size of two for KNN, a max depth of five for Decision Tree, a max depth of two for Random Forest, and a hidden layer size of 2 using the logical activation function for MLP. The Resulting scores for these models are shown for each aggressiveness classification in Table 2. Decision Tree and Random Forest provide the best results of the models tested. Conservative and aggressive classification preforms better than the optimal classification. This is to be expected as the optimal range has two ambiguous zones, one at the conservative-optimal region and one at the optimal-aggressive region. The conservative and aggressive classification only has one of the ambiguous zones. While not important to this thesis, in real world applications, the runtime for the ML models is an important factor to consider. The time the models took to fit and predict was recorded using a timer that tracks how long a block of code takes to run. Each model was in its own code block. Decision Tree and KNN both ran very quickly with Logistic Regression taking just a little longer. Random Forest and MLP where the slowest models.

Table 2: Study 1 ML Model Score Results. Higher scores are better classification results

	Logistic Regression	KNN	Decision Tree	Random Forest	MLP
Conservative	0.414	0.850	0.889	0.885	0.500
Optimal	0.042	0.634	0.833	0.823	0.500
Aggressive	0.898	0.870	0.898	0.901	0.500
Run Time (s)	3	2	1	16	25

As they performed the best, a more in-depth examination of the Decision Tree and Random Forest models were preformed to compare how different feature sets effect the results. This comparison was done by training the models for four different feature sets: spindle power only, the time domain features (xyz and load magnitude), the full 630 feature spectrum, and the reduced feature set previously discussed. Receiver operating characteristic (ROC) curves were then created to compare the different feature sets (Figure 10-11). ROC curves were first developed in the 1940s to analyze radar signals (Green) and have since seen use in the medical fields as diagnostic aids (Zweig), (Hoo) and more importantly, as a powerful machine leaning evaluation tool (Bradley), (Spackman). The curves are generated by plotting the true positive rate vs the false positive rate at different decision thresholds. Then, by taking the area under the ROC curve (AUC), a single value can be obtained that measures the model's performance regardless of decision thresholds (Bradley). The closer the area under the curve is to one, or visually, how quickly the curve rises on the y axis, the better the classifier. The AUC for each model is shown in the legend.

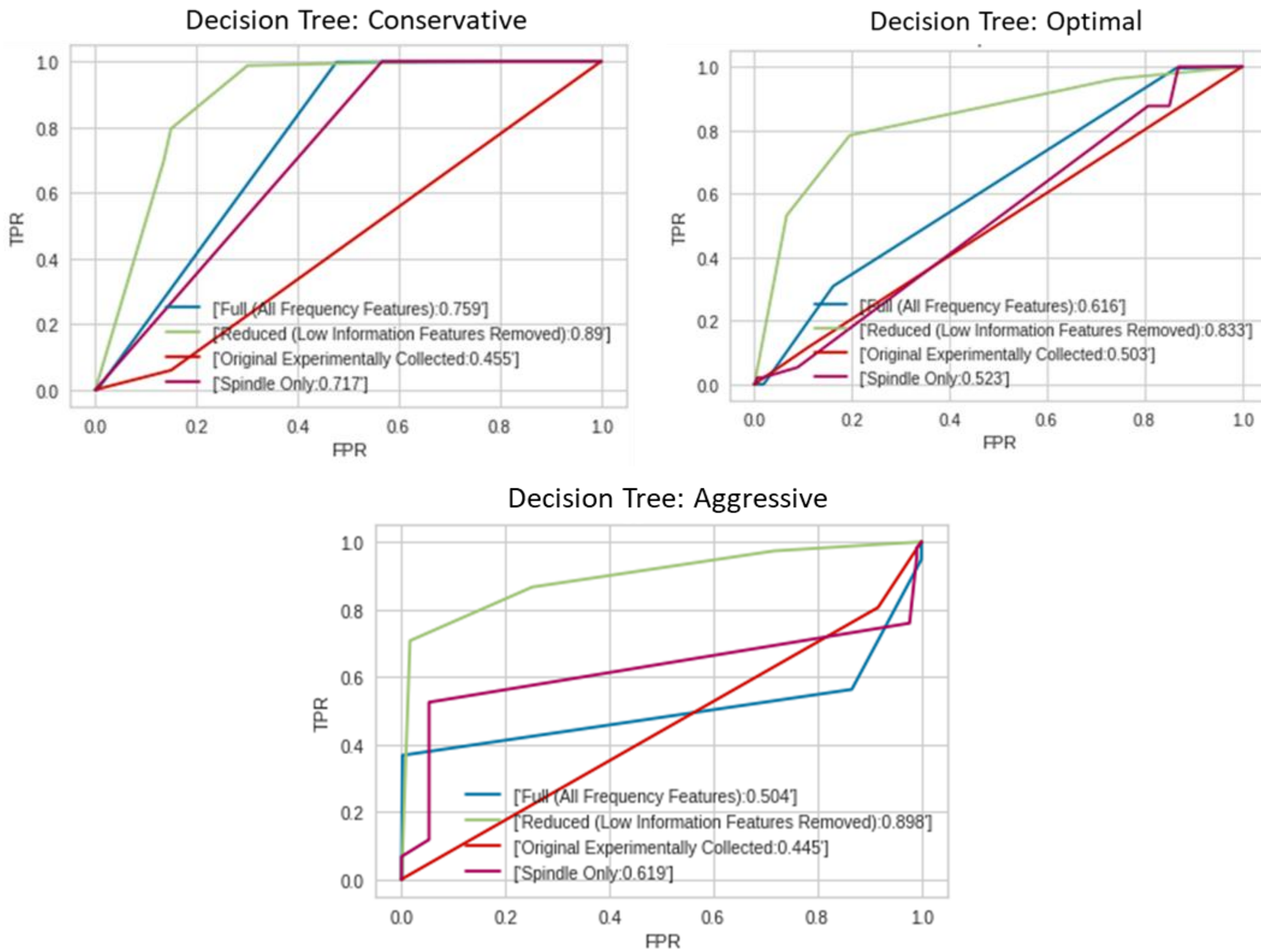


Figure 10: ROC Curves for Decision Tree. Conservative (top left), Optimal (top right), Aggressive (bottom). TPR is true positive rate. FPR is false positive rate

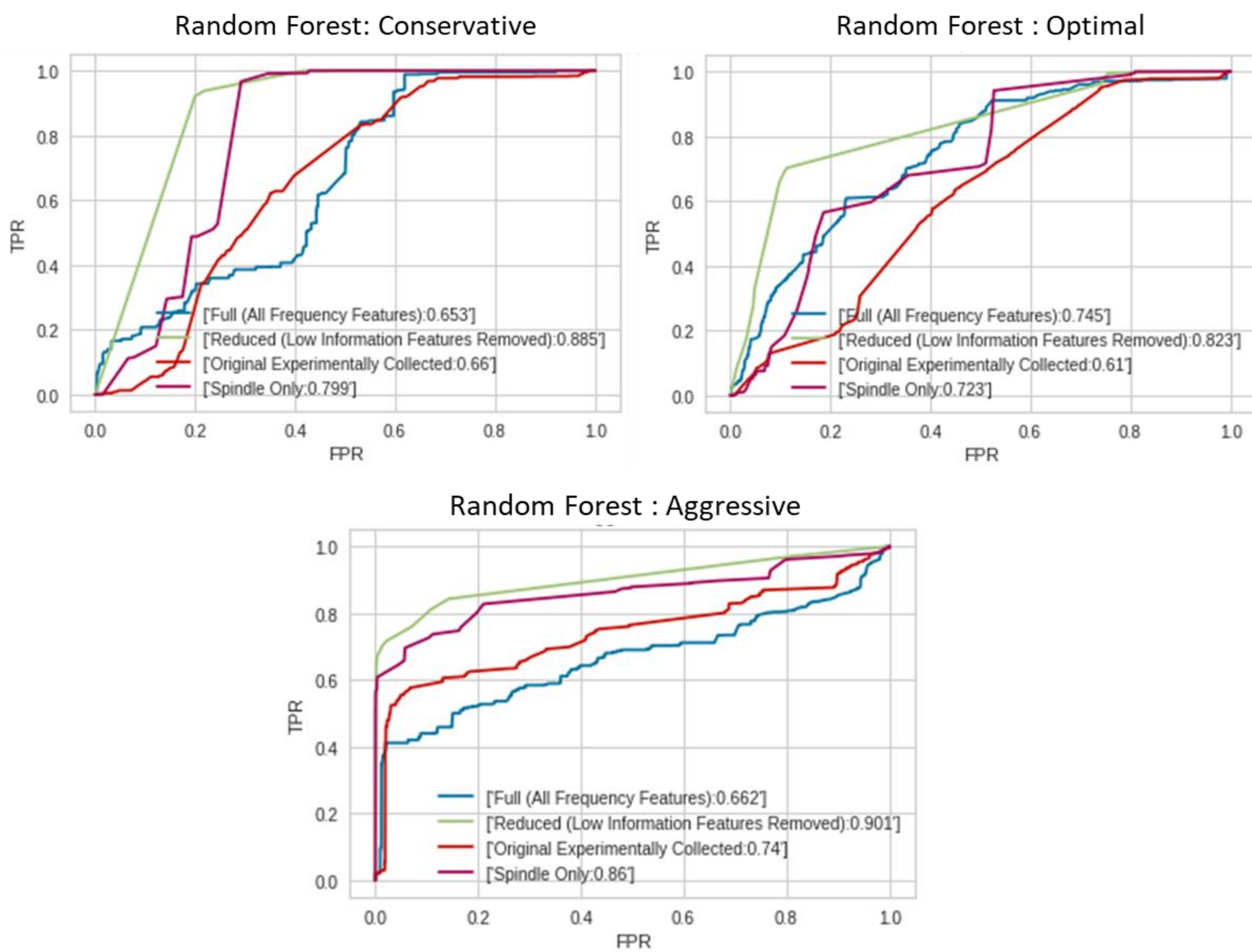


Figure 11: ROC Curves for Random Forest. Conservative (left), Optimal (Right), Aggressive (bottom)

It can be seen that for every aggressiveness, the reduced feature set (green) provides the best results, thus validating the CWT feature creation and subsequent reduction method. Spindle power (purple) also provides good results and depending on the accuracy needed, might be the better method as it is much simpler and faster. Using all time domain data (red) provides the poor results, which is expected as it was determined in the feature reduction section that the time domain load data is not useful for classification and thus would only add noise. Similarly, using all 630 features (blue) produces worse results due to the added noise from the redundant and unimportant features. For the decision tree model, using the reduced frequency features provides an increased prediction accuracy of 24.1% for conservative, 59.3% for optimal, and 45.1% over just spindle power. For the Random Forest model, the accuracy increase is 10.8%, 13.8%, and 4.77% respectively.

CHAPTER 5 Study 2 – Complex Geometry with Increased Variability

Study 1 showed that ML can perform effective aggressiveness classification for complex and varying 2D geometries with the stipulation that the cutting parameters (width and depth of cut) themselves are the same. Study 2 will focus on training a model with varying cutting parameters.

5.1 Geometry and Parameter Selection

To maximize variability in the data, and thus versatility in the model, three additional versions of the previous geometry were made with varying depth and width of cuts (Figure 12). The 2D geometry of the curves was identical to Study 1, only the height (which translates to depth of cut) of the geometry is changed. The width of cut is changed in the adaptive clearing setting as it does not require changing the geometry in any way. Similarly, the training geometries were modified in the same way (Figure 13).

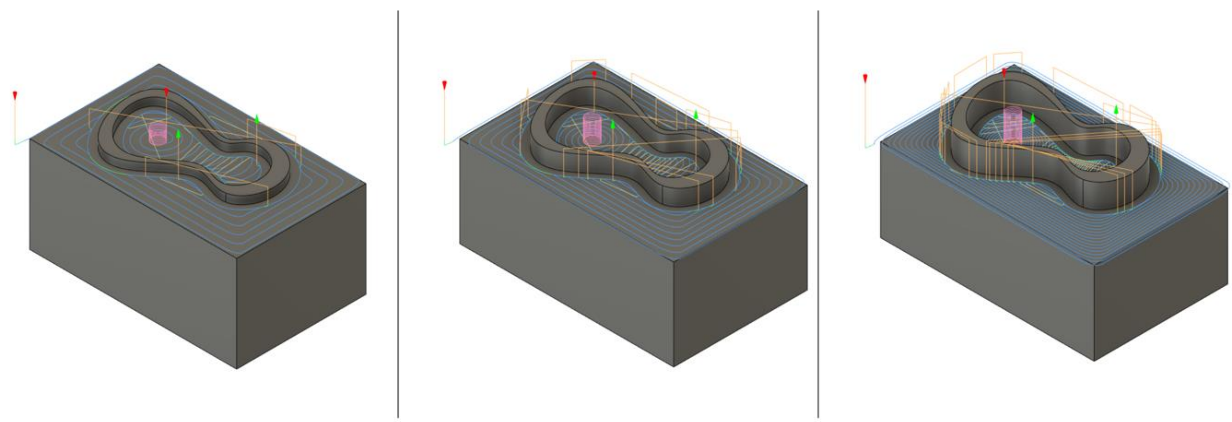


Figure 12: Study 2 Training Toolpaths

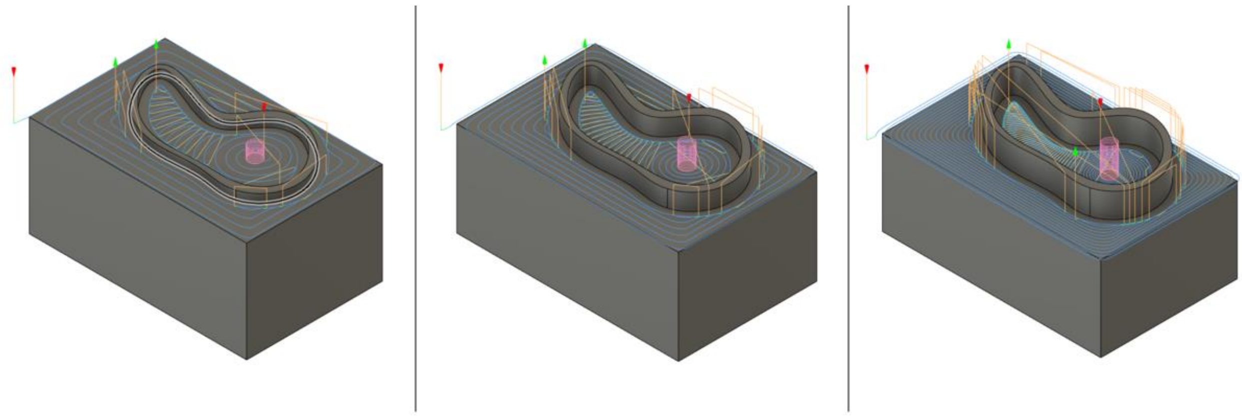


Figure 13: Study 2 Testing Geometry

5.2 Parameter Selection

The same process was used as in Study 1 where a cut was performed for each aggressiveness and the feed-rate was swept throughout the cut to gather a wider range of data. Like before, there was both a geometry for the training set and for the testing set. All in all, there were 12 different cuts performed for each geometry. Four features with different depths/widths of cut were each performed three times with feed rates corresponding to the aggressiveness scale. Table 3 summarizes the cutting parameters for all 12 cuts and shows the starting value the dial was set on to produce the desired feed rates. From this starting value, the dial was turned to the full 120%. This was then done for both the training and testing toolpaths giving a total of 24 different cuts.

Table 3: Cutting Parameters for all 12 Cuts

			Conservitive Feed Rate Range (in/min)			
	Width of Cut (in)	Depth of Cut (in)	Low	High	Value in Gcode	Start Dial
Version 1	0.05	0.45	25.5	35.3	29.4	85%
Version 2	0.1	0.3	22.6	35.9	29.9	85%
Version 3	0.15	0.15	23.3	39.8	33.2	70%
Study 1	0.15	0.25	22.7	37.5	31.25	75%
			Optimal Feed Rate Range (in/min)			
	Width of Cut (in)	Depth of Cut (in)	Low	High	Value in Gcode	Start Dial
Version 1	0.05	0.45	35.3	45	37.5	95%
Version 2	0.1	0.3	35.9	49.2	41	90%
Version 3	0.15	0.15	39.8	56.3	46.9	85%
Study 1	0.15	0.25	37.5	52.3	43.6	85%
			Aggressive Feed Rate Range (in/min)			
	Width of Cut (in)	Depth of Cut (in)	Low	High	Value in Gcode	Start Dial
Version 1	0.05	0.45	45	54.8	45.7	100%
Version 2	0.1	0.3	49.2	62.5	52.1	95%
Version 3	0.15	0.15	56.3	72.8	60.7	95%
Study 1	0.15	0.25	52.3	67.1	55.92	95%

5.3 Study 2 Results and Discussion

After pre-processing this data, and reducing the features, eight of the 630 features remain as shown in figure 12. The same pattern follows where spindle power is the only time domain feature, while the others are frequencies from the load data.

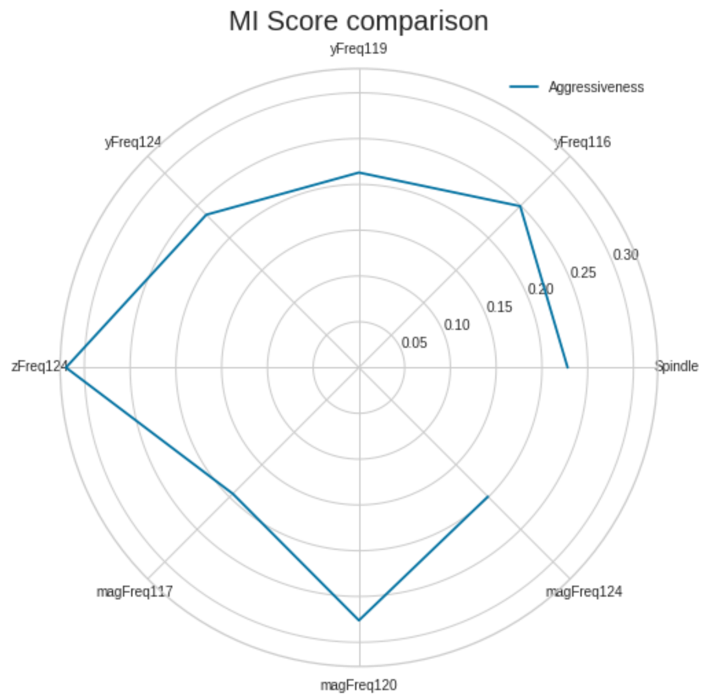


Figure 14: MI Scores for remaining features for Study 2

After hyperparameter optimization, this produced poor results that are summarized in Table 4. Clearly, adding significantly more variable data resulted in the models being unable to accurately classify aggressiveness.

Table 4: Study 2 Initial Model Score Results

	Logistic Regression	KNN	Decision Tree	Random Forest	MLP
Conservative	0.404	0.538	0.454	0.538	0.506
Optimal	0.389	0.559	0.474	0.358	0.540
Aggressive	0.588	0.621	0.602	0.704	0.437

If an effort to further reduce any noise from the data that might be negatively affecting the results, principal component analysis (PCA) was performed on the reduced data for different numbers of components kept to see if this would increase model performance. Principal component analysis reduces the dimensionality of the data. Hyperparameter optimization had to be redone for each of the PCA preformed. These parameters are summarized in Table 5.

Table 5: Hyperparameters for Study 2

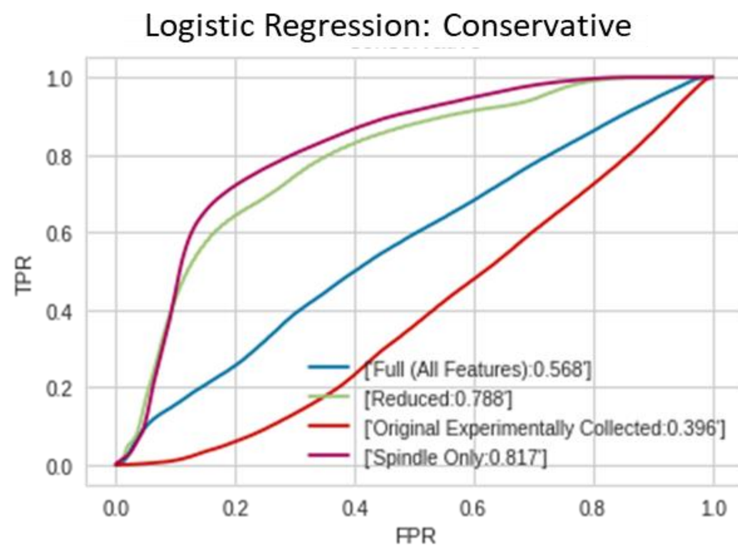
Model	Hyperparameter	PCA n=2	PCA n=3	PCA n=4	No PCA
MLP	Type	relu	relu	relu	relu
	Layer Size	4	8	11	4
Random Forest	Depth	6	4	5	2
Decision tree	Depth	5	3	4	2
KNN	Neighbors	3	6	6	4
	Leaf	2	2	2	2

Table 6 shows the model scoring results from the different principal component analyses performed with the best result for each aggressiveness level for each model highlighted in green. It can be seen that some level of PCA improves the results for all of the models other than MLP, although the results for MLP are still poor. Logistic regression using PCA with n=2 provides the best results and will now undergo further examination as in Study 2.

Table 6: Model Results for Study 2 with PCA

		Logistic Regression	KNN	Decision Tree	Random Forest	MLP
No PCA	Conservative	0.404	0.538	0.454	0.538	0.506
	Optimal	0.389	0.559	0.474	0.358	0.540
	Aggressive	0.588	0.621	0.602	0.704	0.437
PCA n=2	Conservative	0.788	0.623	0.601	0.637	0.478
	Optimal	0.573	0.479	0.490	0.500	0.497
	Aggressive	0.785	0.600	0.640	0.665	0.566
PCA n=3	Conservative	0.701	0.532	0.537	0.579	0.419
	Optimal	0.546	0.364	0.441	0.470	0.491
	Aggressive	0.726	0.634	0.776	0.745	0.404
PCA n=4	Conservative	0.627	0.543	0.577	0.557	0.415
	Optimal	0.448	0.537	0.551	0.549	0.500
	Aggressive	0.705	0.535	0.679	0.715	0.524

ROC curves for the different feature sets are shown in Figure 15. The prediction models, particularly under optimal cutting parameters are less accurate than in Study 2. Similar to Study 2, using the full 630 features and using just the original experimentally collected data provide the worst results overall. For the conservative and optimal cuts, using the spindle power alone provides a slight benefit to the reduced feature set; however, for the aggressive cut, the reduced feature set provides the best results by far. Overall, the reduced feature set still provides the most accurate results.



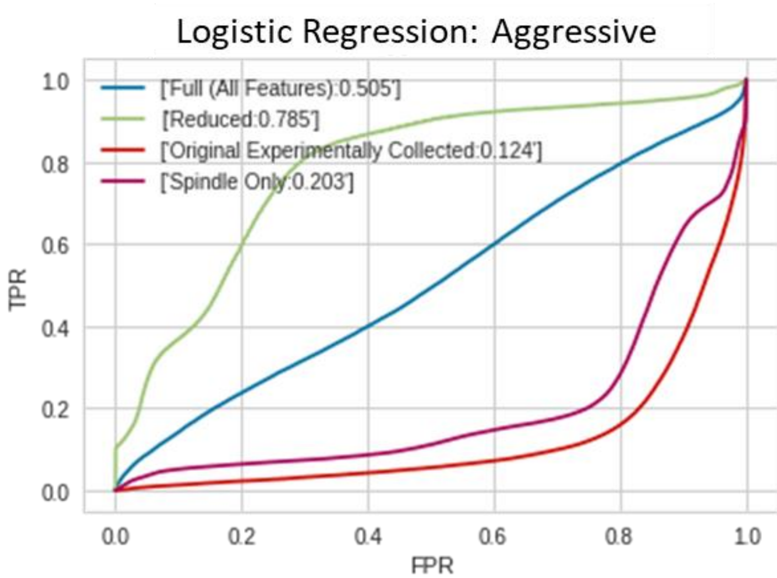


Figure 15: Study 2 ROC Curves for Logistic Regression

This chapter showed that what worked best in the Study 1 with only a single width and depth of cut combination, does not translate with a more variable set of cutting parameters. Not only does a different model preform the best, but and additional processing step (PCA) is required to obtain good results. However, this Study does confirm that the proposed process of frequency feature extraction followed by feature reduction via Person Coralation and MI Score is important to achieve the best results.

CHAPTER 6 Conclusion

CNC operations are an advanced process that requires extensive training for an operator to become proficient. Mistakes increase lead times, cost, and can even cause injury. The ability to autonomously detect potential faults before they happen is critical for both manufacturing facilities and makerspaces.

This approach of using data from sensors built into a CNC machine to train a machine learning algorithm to predict how aggressive the machine is cutting provides valuable insight into different strategies that can be used to further understand how these data streams are affected by machining parameters. This is an important building block to achieve “born-certified” parts from a manufacturing facility while also increasing safety and accessibility of CNC within makerspaces.

6.1 Evaluation of Research Question and Hypothesis

To evaluate the Research Question and Hypothesis, a process for collecting and processing data from a CNC mill to train machine learning algorithms to predict the aggressiveness of a machining operation was tested. Additionally, an examination into how different features effect model performance was conducted.

Research Question: What sensor data from a CNC milling machine is adequate to characterize cutting parameters for a 2D toolpath?

Hypothesis: The built-in axis drive and spindle load sensors in a standard CNC Mill are sufficient to reliably classify the “aggressiveness” of the cutting parameters of a 2D toolpath regardless of the cut geometry and cut parameters (depth/width of cut and feed rate).

This Hypothesis was validated through experimentation that under a variety of circumstances, ML can be used to reliably predict the aggressiveness of a cut using axial load and spindle power data. However, simply using the time domain axial load data is not sufficient as it was shown that these features are not important to aggressiveness classification. Performing frequency analysis of the time domain features and reducing them based on MI score and Pearson Correlation is necessary to provide the best results. If only using time domain data, using spindle power alone provides better results than if including axial loads.

Using a broad range of cutting parameters with complex toolpaths, logistic regression produced scores of 0.788 for conservative cuts and 0.785 for aggressive cuts. Optimal cuts produced worse results at 0.573 due to having two boundary regions instead of one. However, when using a constant width and depth of cut as in Study 1, decision tree preformed the best, with Random Forest a close second, with scores of 0.889 for conservative, 0.833 for optimal, and 0.898 for aggressive. Thus, it is not as simple as saying one of the models is the best for all application. Several different models need to be considered and tested depending on the desired application. To implement this work, the designer needs to understand the operating window for the desired application. If there is only a single or possibly vary narrow range of depth/width of cut combinations being used, such as on a manufacturing line making the same part repeatedly, then Decision Tree and Random Forest models should be considered. If operation under a wide range of depth/width of cut combinations Logistic Regression is likely going to be the best model. It is worth noting however, that accuracy of the models needs to also be balanced with how long it takes the models to run if this work is to be conducted in real time. While Decision Tree and Random Forest both provide very similar accuracy, Decision Tree runs much faster and is thus the clear choice under the conditions used in Study 1.

6.2 Engineering Applications of this Work

This aggressiveness classification has a direct use within makerspaces, where CNC operators are not professionals, and they must oversee users with little to no experience in both CNC and CAM. A facility can define an operation window as ‘too aggressive’ however they see fit and train the ML model accordingly. If a cut is being performed that is in this window, a warning can be sent to the machine operator. However, a makerspace can see an infinite combination of toolpaths and machining parameters and it was shown that increasing the range of these parameters does hurt the model performance. To produce a sufficiently accurate model to predict every conservable operation that may occur in a makerspace would likely require an extensive range of training data.

In a different scenario, such as in a manufacturing facility, where either a single part, or only a couple parts are being produced on an assembly line has the potential to leverage this workflow with much greater accuracy as the model only needs to be trained on the parameters being used by the facility. A manufacturing facility can train the model using data from a part that is being produced under ‘perfect’ conditions. If the model finds that the machine is cutting outside this window, that is a sign something might be wrong with that operation, such as incorrect raw material, overworn tool or incorrect cutting parameters.

6.3 Limitations and Future Work

While a broad range of cutting parameters is used to train the ML model, there is an infinite combination of parameters that can be used in CNC milling, including 3D operations. Additionally, this research focuses on only 6061 aluminum as the workpiece material. Further work is necessary to determine if this approach holds up when different materials are used. It is unlikely that this approach would work when training the models with only spindle power and axial load data so additional side channel data would need to be collected and analyzed to increase model performance. Also, for this research to be applied to makerspaces or manufacturing facilities, this approach of data collection and ML classification will need to be conducted in real time. Currently, the data is exported post cutting operation and processed on a computer at a later time. A process will need to be created that can send the data to a computer for processing and classification in real time. Many CNC milling machines, including the EMCO E350 used in this research, do have the ability to remotely connect to a computer so this could possibly be done without any modifications to the machine. A flowchart on how this might work is shown in Figure 16, where the collected data can be sent to a server where a pretrained ML model can predict classification. Additionally, this research was only done with a single machine. Additional testing needs to be done to determine if the models hold up when given testing data taken from a different machine than the training data.

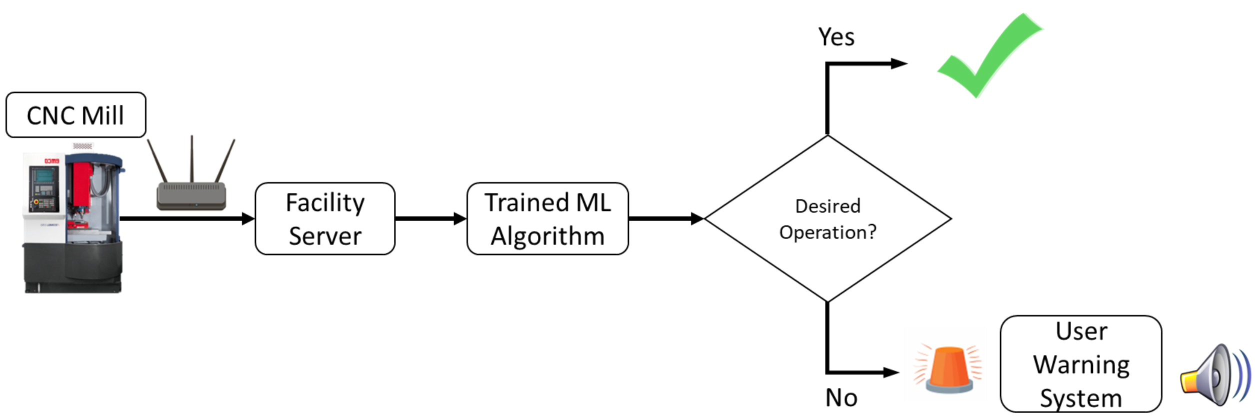


Figure 16: Flow Chart for Real Time use of Aggressiveness Classification

This thesis has shown that there is promise with using built in sensors within a CNC milling machine along with machine learning to characterize cutting parameters regardless of cutting geometry. While there is more work to be done to further increase accuracy and versatility, this work can improve safety and consistency of machining operations within both hobbyist makerspaces and professional manufacturing facilities.

Appendix A. Setup Trace Function for 840D Controller

Trace Function variables used in this research:

$/Channel/!SEMA/aaPowerSmooth [u1 4] Smoothed drive power

/Channel/!SEMA/aaLoadSmooth [u1 1] Smoothed drive load

/Channel/!SEMA/aaLoadSmooth [u1 2] Smoothed drive load

/Channel/!SEMA/aaLoadSmooth [u1 3] Smoothed drive load

Instructions for starting the trace were provided by Georgia Tech graduate student Alexanda Schueller:

Work-Instruction: Creating and Running CNC Trace File

1 Background Information

Trace programs are used to measure variables within a CNC machine while it is in operation.

All procedures mentioned in this document have not been found to stop the machine while it is in cycle. For this reason, data gathering does not require extensive planning and a shut-down.

These instructions apply to a Siemens 840D controller.

This process has been tested on Grob and Heller machines.

2 Pre-requisites

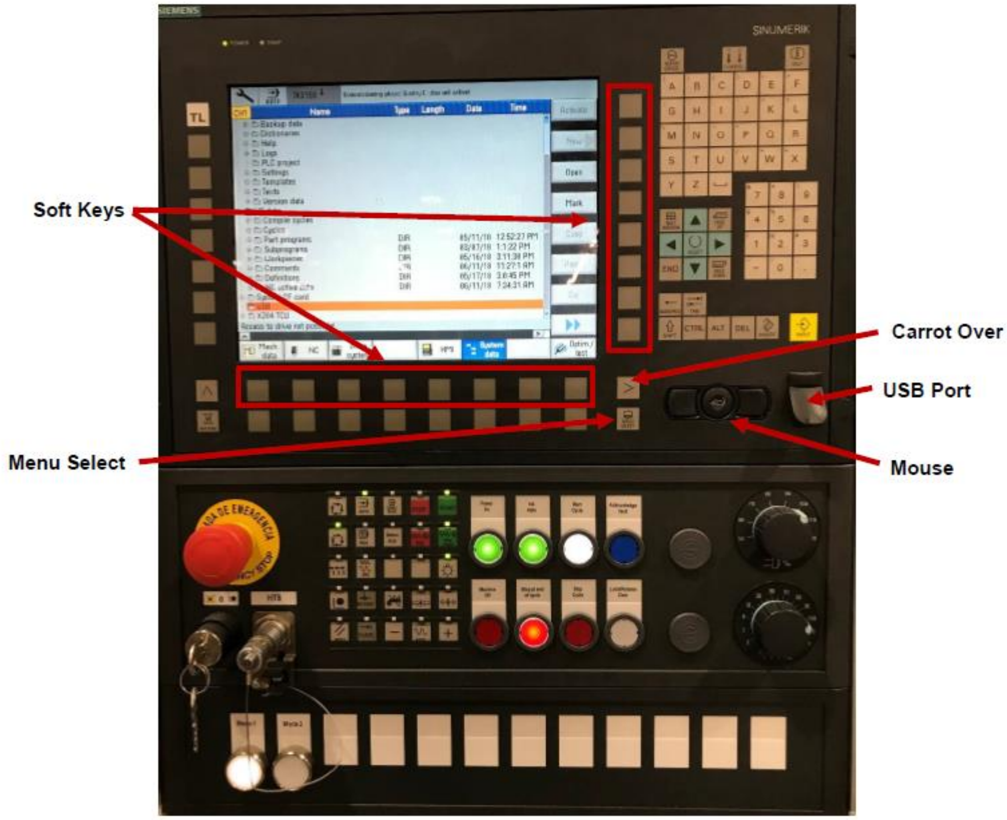
Ask for permission to run trace file from operator.

A USB stick/drive that can be used to store the trace file.

In the instance that you are running a file that has already been created in the past, you will need that trace file.

If the password needed to gain access has been changed from factory settings, you will need this.

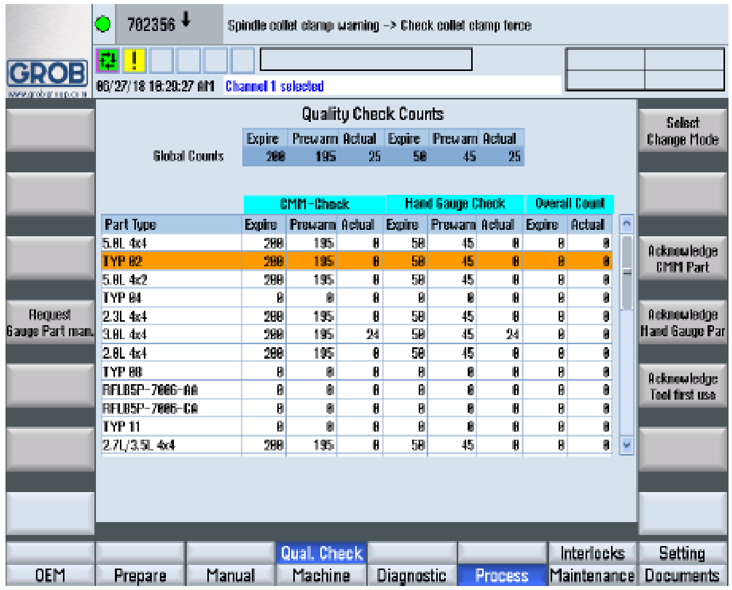
3 Machine Interface Overview



4 Gaining access to the machine (if not already unlocked)

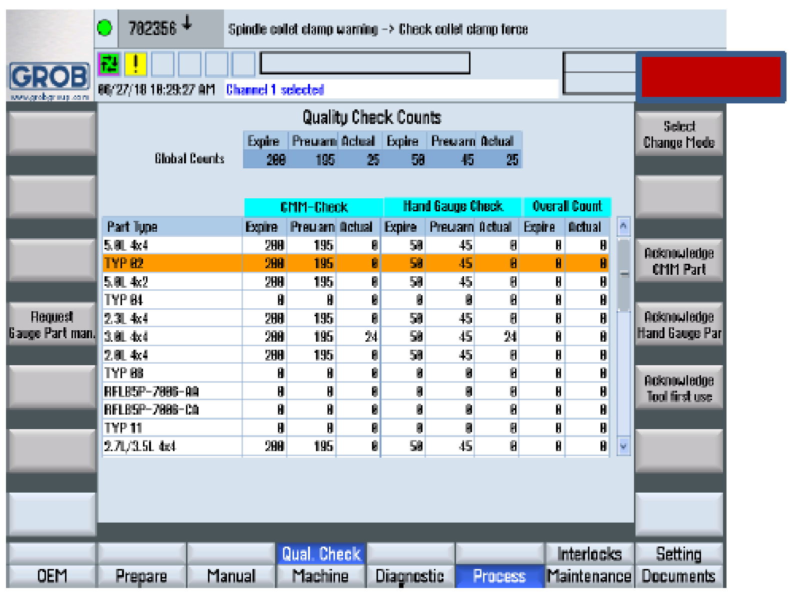
Note: These directions are derived for ATO plant. Gaining access to the machine can be different depending on the plant.

4.1 Navigate to process screen if not already present (**Menu Select – Soft Key "HMI" – Soft Key "Process"**)

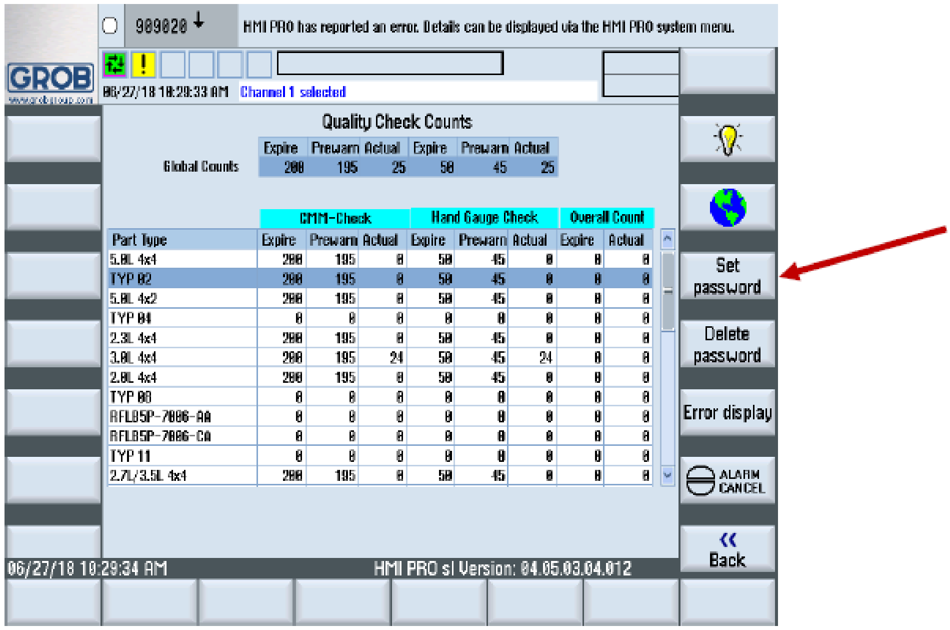


Process Screen

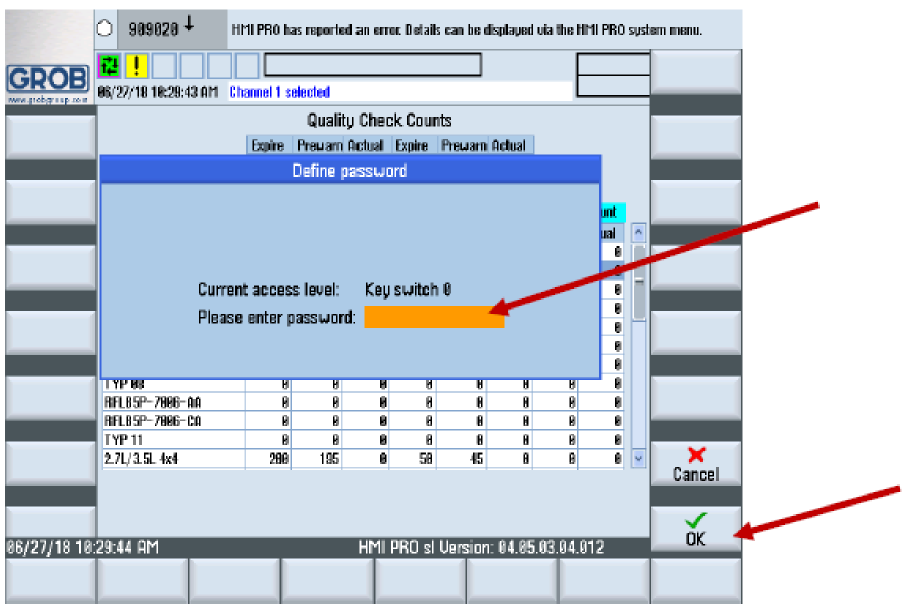
4.2 Select **Soft Key** corresponding to the red box in the figure below (this is a hidden button)



4.3 **Soft Key "Set Password"**



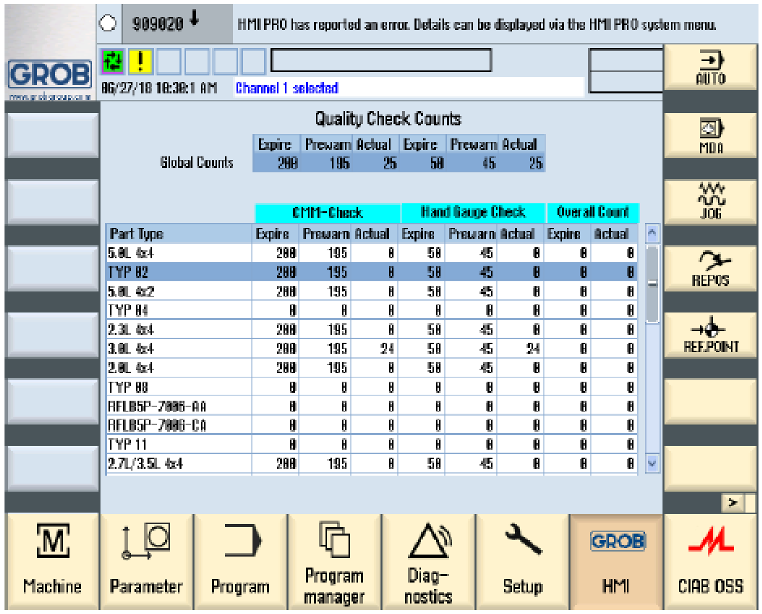
4.4 Enter **“sunrise”** – **Soft Key “OK”**



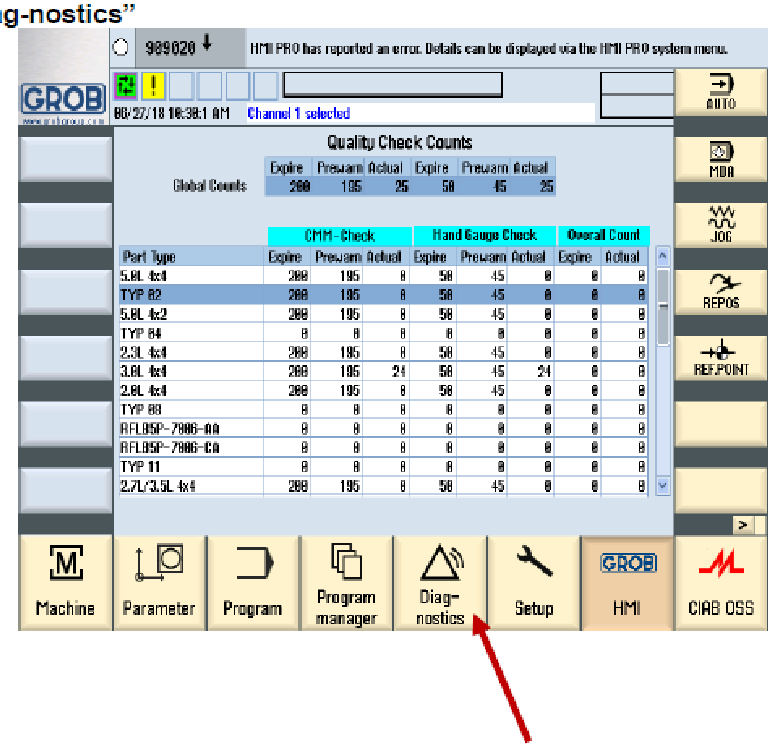
4.5 You know have write access in the machine so false moves can stop the machine operation

5 Navigate to Trace Program

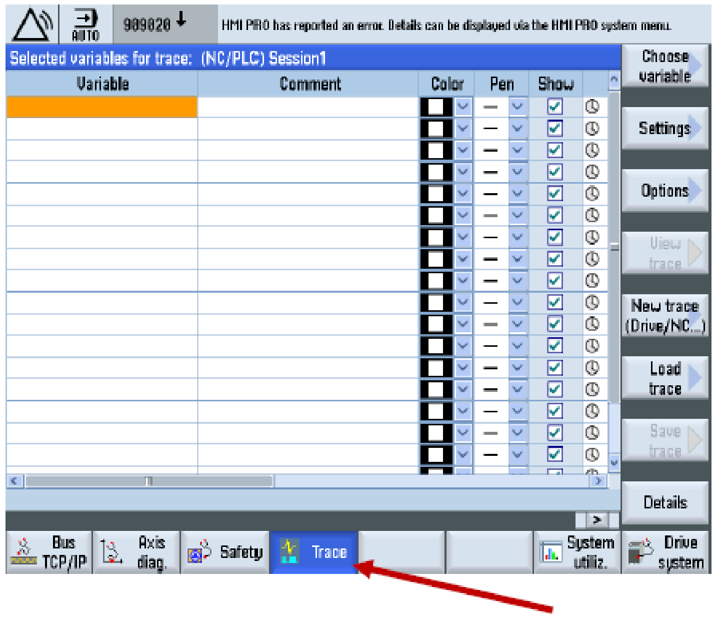
5.1 Menu Select



5.2 Soft Key "Diag-nostics"



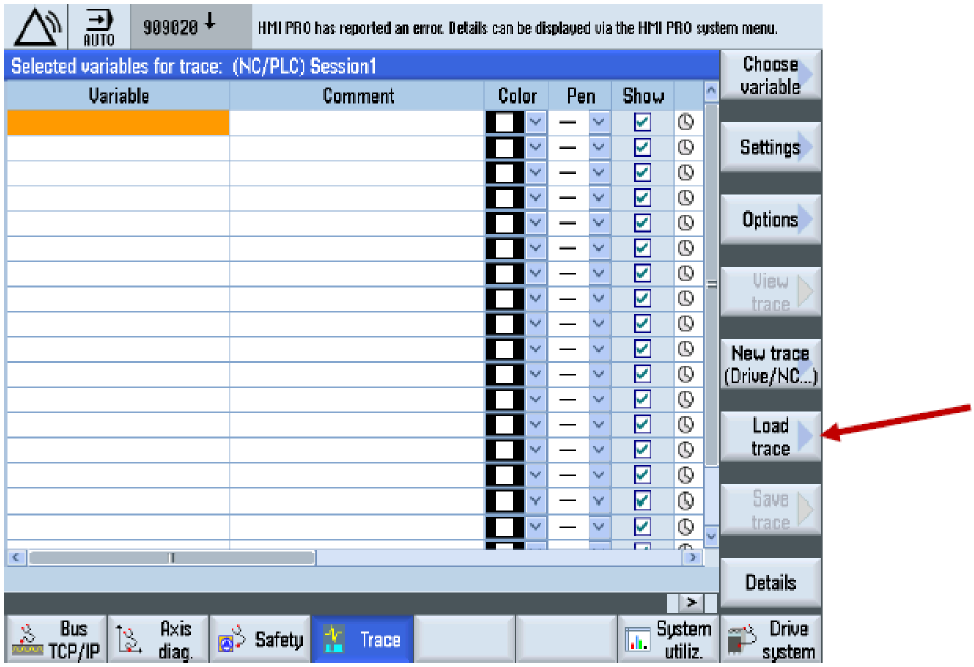
5.3 **Carrot Over** (see section 3) until you can **Soft Key "Trace"**



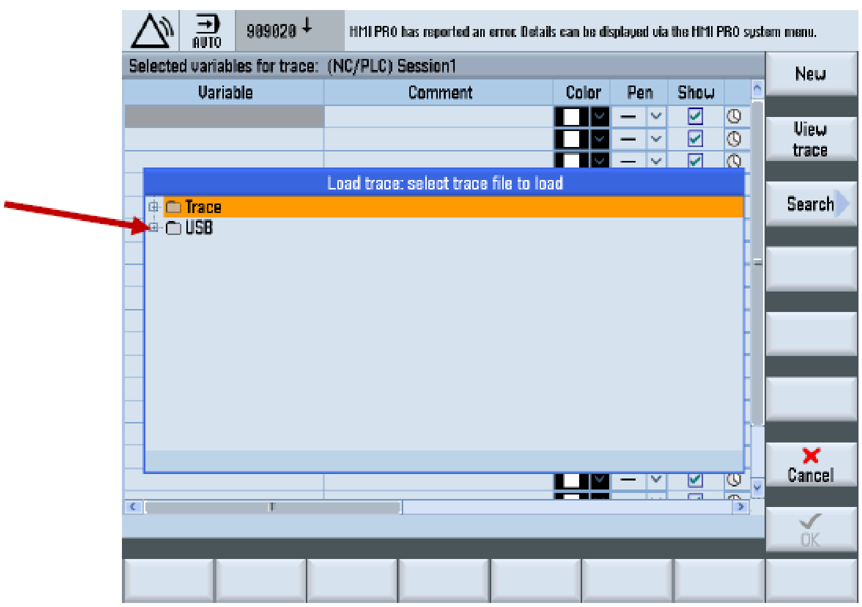
6 Set-Up Trace

If you want to start a brand new trace then select "new trace" in the soft keys. Most likely you will want to run a trace whose settings have already been established. This section will overview that process.

6.1 Soft Key "Load Trace"



6.2 **Plug in your USB**
6.3 Use arrow keys to navigate and **"Select" USB**

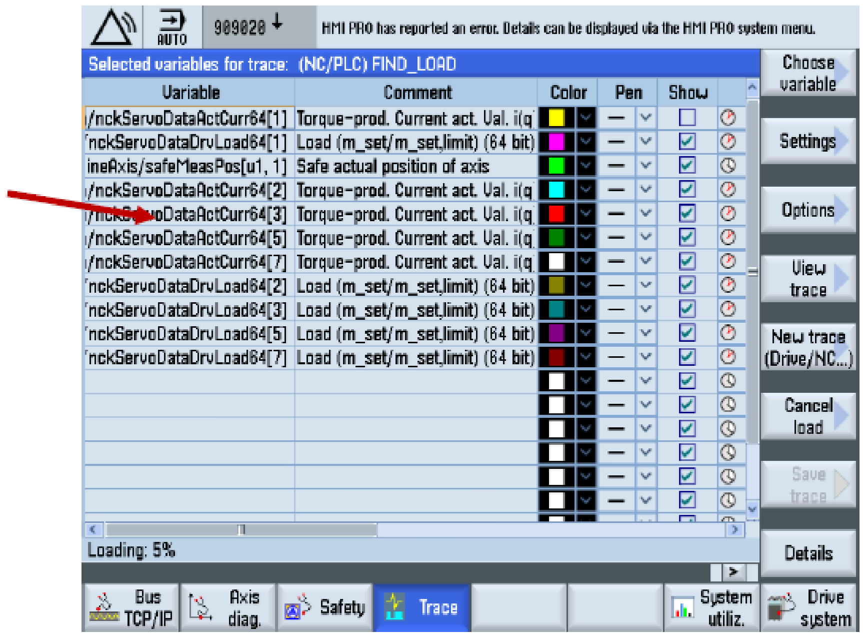


6.4 Use arrow keys to **highlight the proper trace file** you desire and **Soft Key "OK"**
6.5 **Highlight Proper option – Soft Key "OK"**

6.5.1 Choosing just variables and settings will bring in the proper variables traced in the file as well as the proper settings **(probably the best option)**

6.5.2 Choosing the second option will also import the data from that trace file if you wish to view that

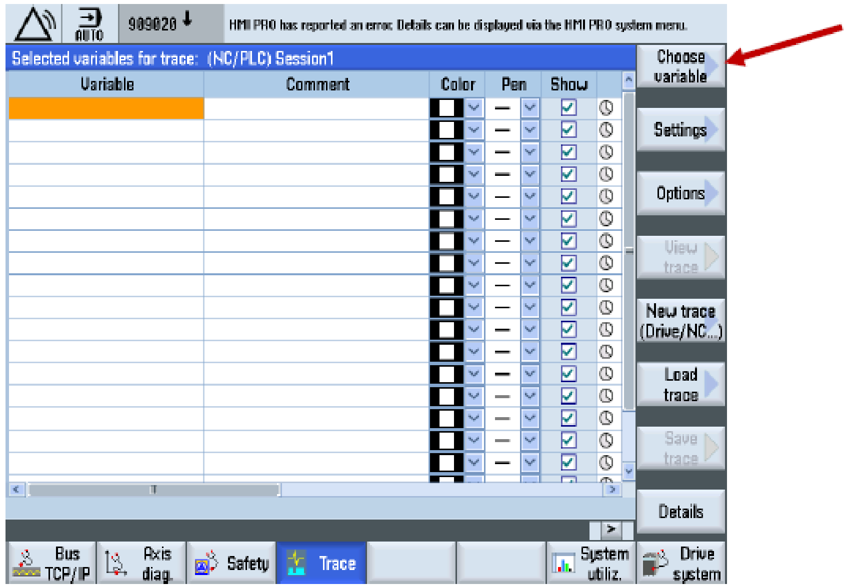
6.6 Screen should now have variables in the list



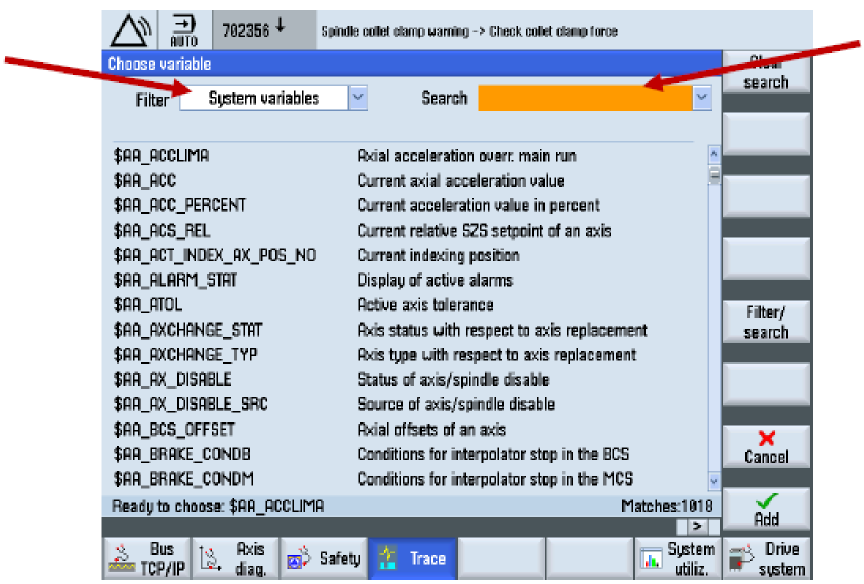
7 Choosing the Proper Variables to Trace (if desired)

Choosing the right variables is often a trial and error situation unless you are loading a previous trace file.

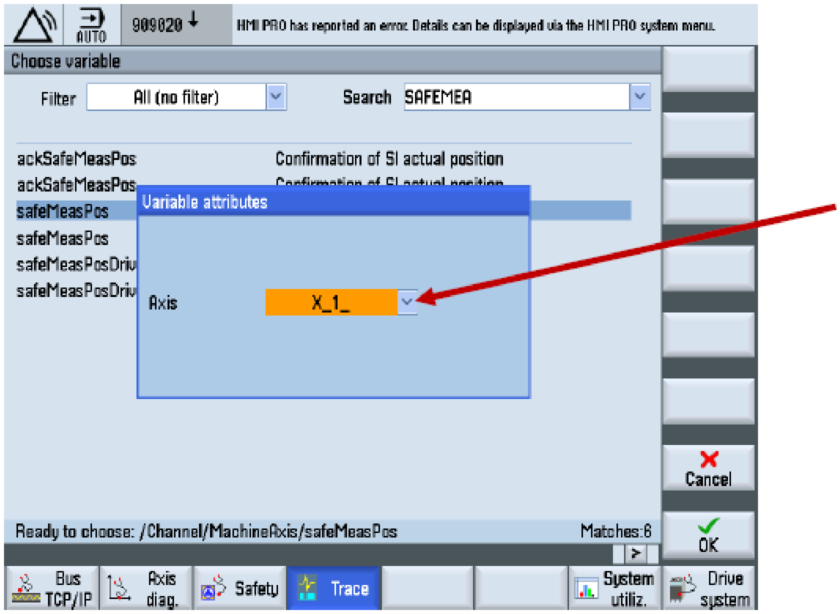
7.1 **Soft Key "Choose Variable"**



7.2 Search for variables in the search box. You can contrain the search to certain directories on the "FILTER" menu select



7.3 Depending on the variable you may have to select which axis you want to trace. Change this using the mouse or the "Select" button in the middle of the arrow keys.

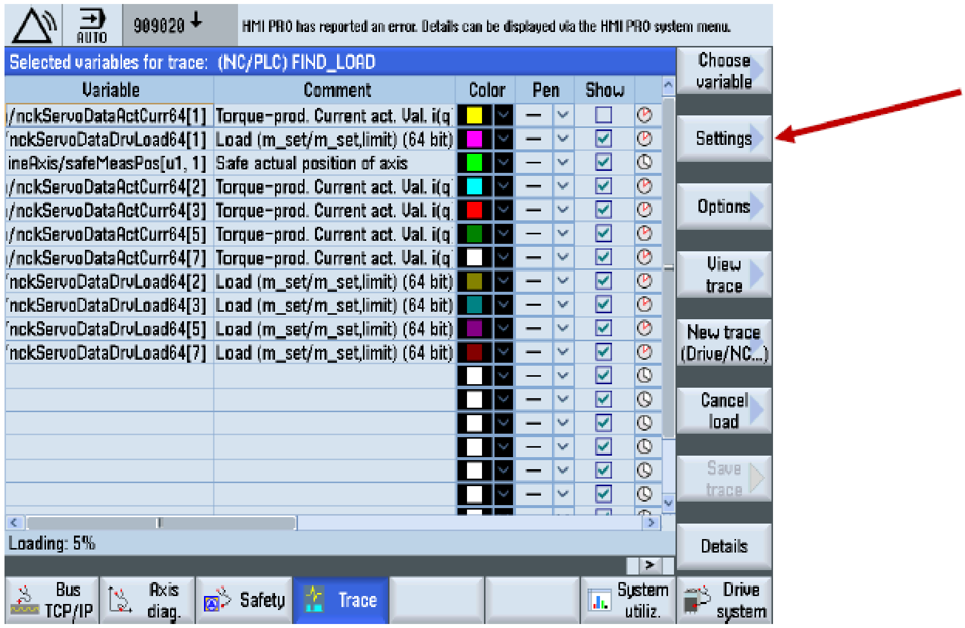


7.4 Soft Key “OK”

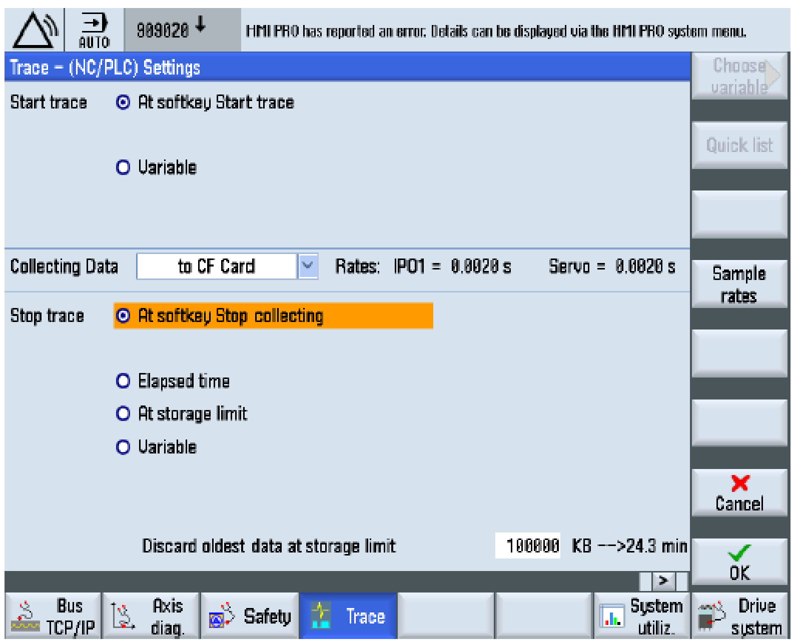
8 Trace Settings

Settings are very important to verify for correctness as they could write data to the wrong areas, causing machine shutdown.

8.1 Soft Key “Settings”



8.2 Ensure start trace is set to “At softkey start trace”



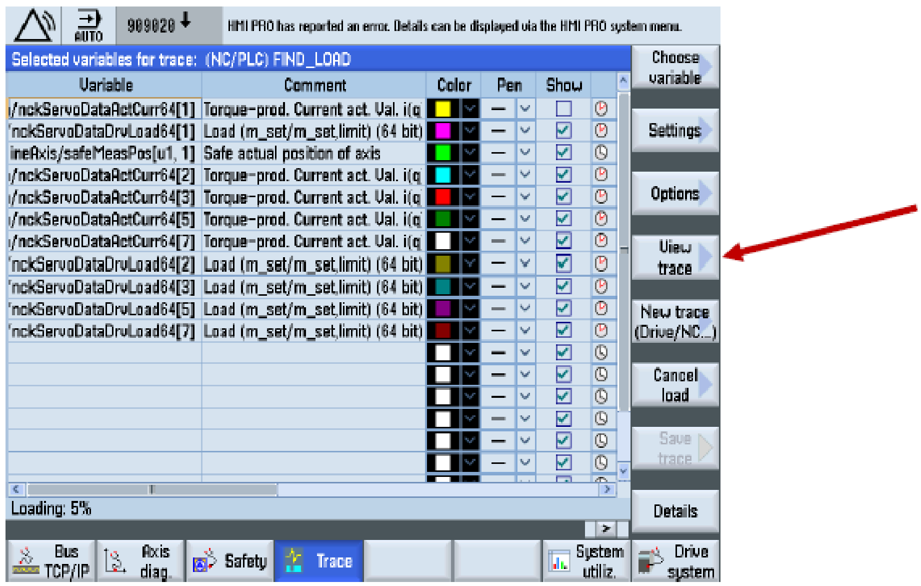
8.3 **Ensure "Collecting Data" is set "to CF Card". Failure to ensure this step could write data to the NC Kernel (NCK) causing card space to be lost and complete machine shutdown**

8.4 Ensure "Stop Trace" is set to "At softkey Stop collecting"

8.5 Start and stop parameters can be changed to start and stop based upon the chaning of variable values. Doing this is outside the scope of these work instructions.

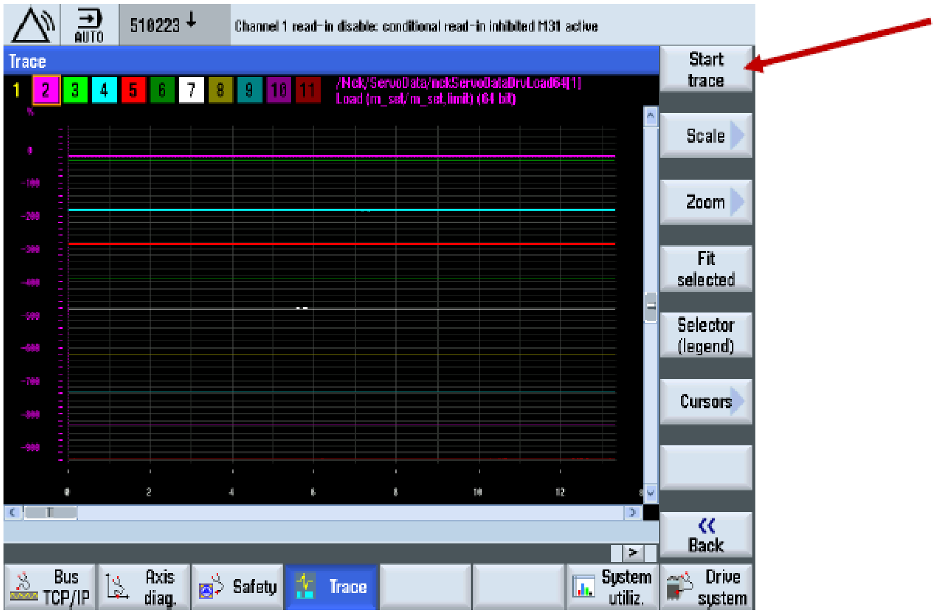
9 Executing Trace File

9.1 Soft Key "View Trace"

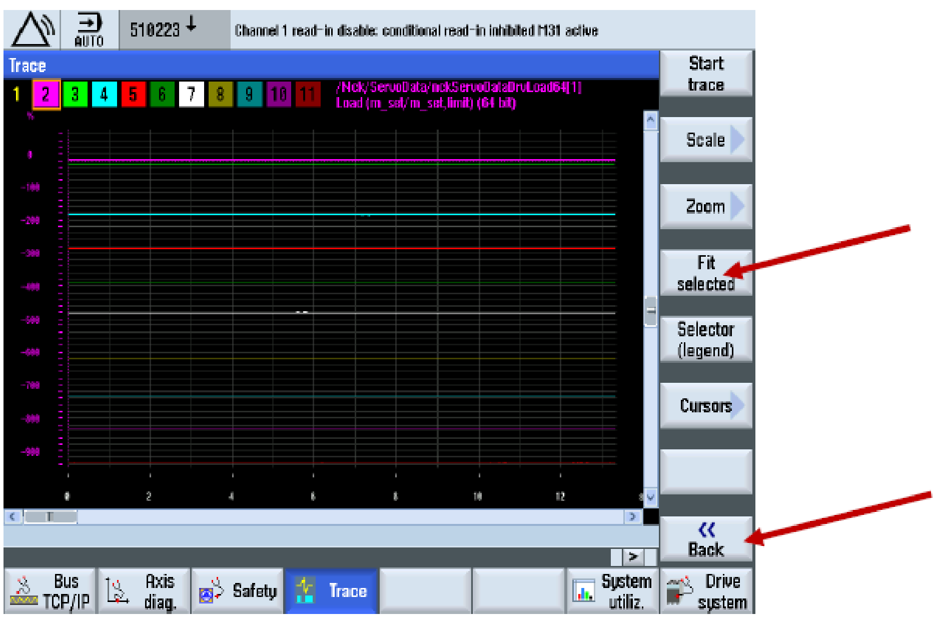


9.2 Soft Key "Start Trace"

Note: Data will start to be collected after about 2 seconds.

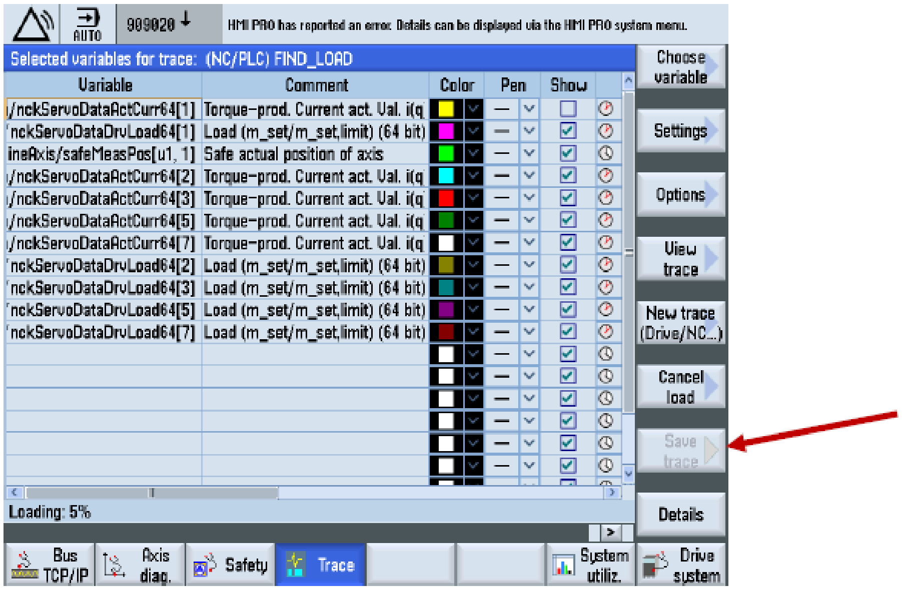


9.3 **Soft Key "Stop Trace"** to end the trace
9.4 **Soft Key "Fit Selected" or "Fit All"** to view the data, push **Back** to get to main screen

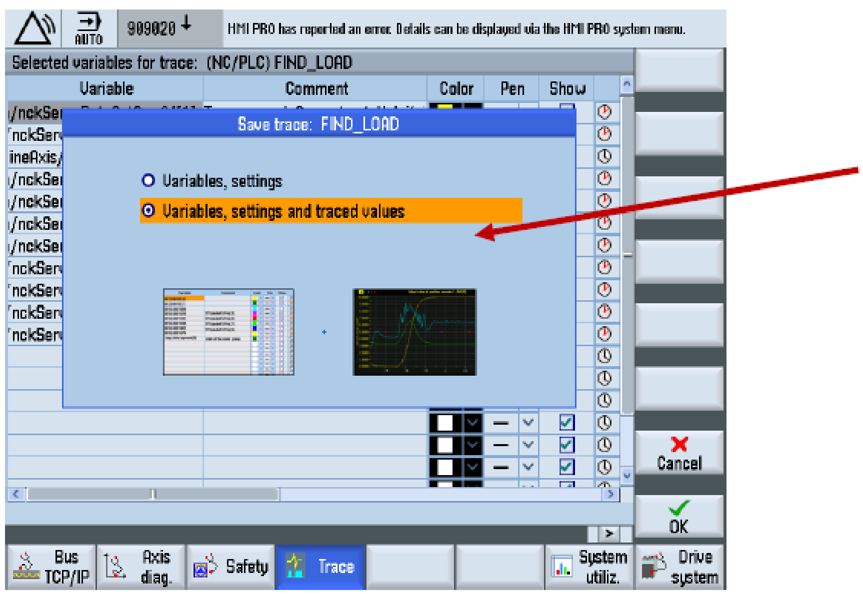


10 Save trace file

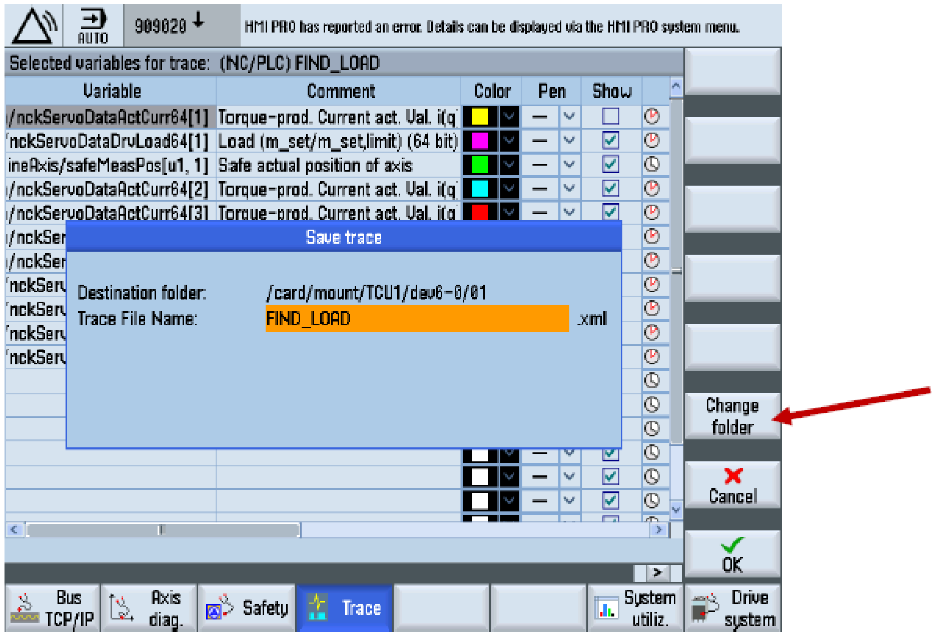
10.1 Soft Key “Save Trace”



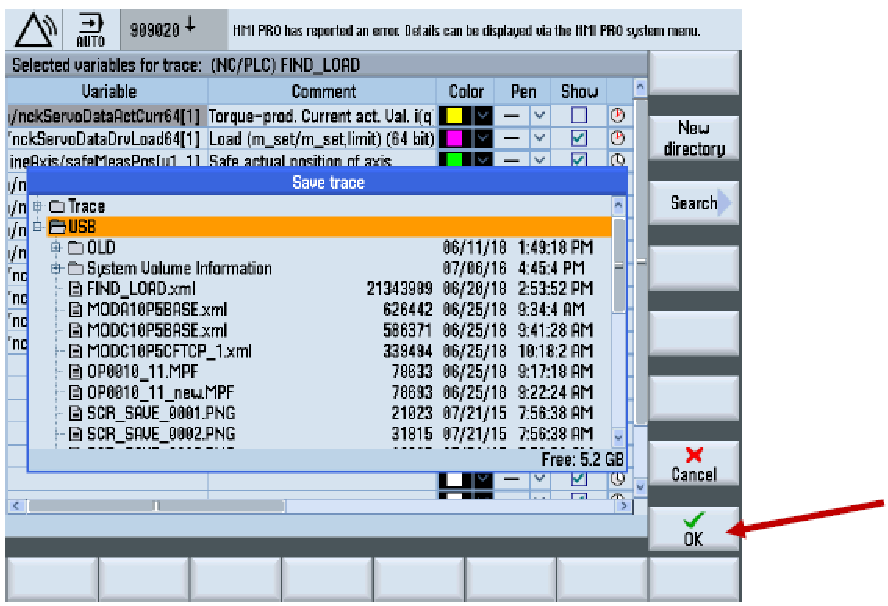
10.2 Highlight variables, settings, and traced values



10.3 **Soft Key "Change Folder"**



10.4 **Highlight USB and Soft Key "OK"**



Appendix B. Code

B.1. Python Code to Convert .xml Files to .csv for Data Processing

```
!pip install matplotlib
!pip install --upgrade sklearn

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import ShuffleSplit

import statistics as stat
import numpy as np
import matplotlib.pyplot as plt

from xml.dom import minidom

def read_file(filename):
  # Load the XML file from file filename
  # This file needs to be uploaded to Colab in the Files tab (in the left
sidebar)
  document = minidom.parse(filename)
  trace_data = document.getElementsByTagName('traceData')[0]
  trace = trace_data.getElementsByTagName('dataFrame')[0]
  data = []
  for rec in trace.getElementsByTagName('rec'):  # Iterate through each da
ta point
    # Get the values of the defined attributes,
    # or use the previous data point if undefined
    row = [float(rec.getAttribute(attr) or data[-
1][i]) for i, attr in enumerate(attributes)] #check this line if any of th
e attributes are skipped over
    data.append(row)
  return np.array(data)

######### Read data from XML file ########

# Define the attributes from the XML file that we want to read
attributes = ['f1', 'f2', 'f3', 'f4']

files = [(0, 'EXP5_6061_T_2_C_1.xml')]#, (1, 'PRL2clean.xml')]

data_sets = []
label_sets = []
for mat, filename in files:                          #for each file
  data_set = read_file(filename)                     #saves data to data_set
```

```python
  data_sets.append(data_set)                    #adds data in data_set to 
data_sets so all file are included in one variable (append vs extend?)
  labels = np.empty(data_set.shape[0]) #creates empty set the size of data
  labels.fill(mat) #labels each data_set point withb appropriate materiel 
classification (0 or 1)
  label_sets.append(labels)

data = np.vstack(data_sets)
labels = np.hstack(label_sets)

print('Read {} rows of data'.format(data.shape[0]))
dataLength = len(data[:,0])
dataPoint = range(dataLength)


power = data[:,0]
xload = data[:,1]
yload = data[:,2]
zload = data[:,3]

print(np.mean(data[:, 1]))
print(labels)

a = np.asarray( [power, xload, yload, zload])

### EXPORT AS CSV ###
a = np.asarray( [power, xload, yload, zload])
np.savetxt("EXP5_6061_T_2_C_1_Dirty.csv", a, delimiter=",")
```

B.2. MATLAB Code to Pre-Process Data

```matlab
clc
close all
clear all

%%%EDIT THESE VALUES%%%
file = 'EXP5_6061_T_2_C_1_';
aggresiveness = '_Conservative_'; % Aggressive Optimal, Conservative
% threshold = 400; %designate the value of which anything below is the spindle not 
cutting, choose something a little lower than the actual threshold and remove the 
actual values at a later time
% actualThreshold = 581; %remove spindle data the actual threshold, this is where 
you'll be removing values below you're actual threshold
exportData = 0; %if 0 it doesn't export, if 1 it does
minspindle = 235;  %Enter max falue of spindle power when cutting in air (make a 
little higher then actual
```

```
% firstramp = 14870;
% lastramp  = 18500;

firstPoint = 5929;
lastPoint = 55527;

%make sure to cut into the first and last pass a little bit
%%%

data = readmatrix(append(file,'Dirty.csv'));
lastPointFromEnd = numel(data(1, :))-lastPoint;



% Plot Before Data
figure
plot(data(1,:))
title('spindle Before')
% figure
% plot(data(2,:))
% title('X Load Before')
% figure
% plot(data(3,:))
% title('Y Load Before')
%
% figure
% plot(data(4,:))
% title('Z Load Before')

datacull = data(:,firstPoint:end); %Remove everything before a certain data point,
remove strange initial passes
datacull = datacull(:,1:end-lastPointFromEnd); %Remove everything before the last
data point, remove strange last passes

hold on;
figure
plot(datacull(1,:))
xlabel('Data Point')
ylabel('Spindle Power')

title([file, aggresiveness, ' spindle culled'])
% ylim([0 1200])

k = 1;
for i = 1:length(datacull)
    if datacull(1,i) > minspindle      %if the spindle power is above
minspinle(cutting occures) saves data in new matrix
        new_data(1,k) = datacull(1,i);
        new_data(2,k) = datacull(2,i);
        new_data(3,k) = datacull(3,i);
        new_data(4,k) = datacull(4,i);
        k = k+1;
    end
end
```

```
spindle = new_data(1,:);
xload = new_data(2,:);
yload = new_data(3,:);
zload = new_data(4,:);
data_num = [1:1:length(spindle)];
figure
plot(data_num,spindle)
title('Curvy Doughnut Splindle Power 6061 Aggressive)')
xlabel('Data Point')
ylabel('Spindle Power (W)')


if exportData == 1
    writetable(array2table(spindle),
[file,aggresiveness,'_Spindle_AutoSeperated.csv'])
    writetable(array2table(xload), [file,aggresiveness,'_xLoad_AutoSeperated.csv'])
    writetable(array2table(yload), [file,aggresiveness,'_yLoad_AutoSeperated.csv'])
    writetable(array2table(zload), [file,aggresiveness,'_zLoad_AutoSeperated.csv'])
end
```

B.3. Python Code to Preform Feature Down Selection and Train/Test Machine Learning Algorithms

```
# import tensorflow as tf
# from tensorflow import keras
!pip install yellowbrick

import pandas as pd
from numpy import genfromtxt
import numpy as np
import csv

import sklearn
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import ShuffleSplit
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import make_scorer
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.model_selection import cross_validate

from sklearn import tree
```

```python
from sklearn.manifold import TSNE
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.feature_selection import mutual_info_classif
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import StratifiedKFold
from sklearn import svm, datasets
from sklearn.metrics import auc
from sklearn.metrics import RocCurveDisplay
from sklearn.linear_model import RidgeClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OrdinalEncoder, LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier



from yellowbrick.classifier import ROCAUC
from yellowbrick.datasets import load_game

import matplotlib.pyplot as plt
import pandas_profiling

from scipy import signal
from scipy.signal import find_peaks
import pywt
from scipy.fft import fftshift
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.mlab as mlab



from matplotlib.ticker import NullFormatter
from sklearn import manifold, datasets


from __future__ import print_function
import time
from mpl_toolkits.mplot3d import Axes3D


from google.colab import drive
drive.mount('/content/drive')
from sklearn.utils import shuffle
```

```
def scores(y_true, y_predict):
  return [accuracy_score(y_true, y_predict),precision_score(y_true, y_pred
ict, average='weighted') , recall_score(y_true, y_predict, average='weight
ed'), f1_score(y_true, y_predict, average='weighted')]


from sklearn.metrics import auc, roc_curve

def ROC_OOE(y_predT, y_test):

  # classes = np.unique(y_test)
  n_classes = 3
  classes_ = ["conservative", "optimal", "aggressive"]
  colors = [range(len(classes_))]
  fpr = dict()
  tpr = dict()
  aucs = np.zeros([1, 3])
  thresholds = dict()

  ax = plt.subplots()

  for i, c in enumerate(classes_):

    fpr[i], tpr[i], thresholds[i] = roc_curve(y_test[:,i], y_predT[:, i])
    # roc_auc_score(y, y_predT[:, i])

    aucs[0, i] = auc(fpr[i], tpr[i])
    plt.plot(fpr[i],tpr[i], label = [classes_[i]])

  plt.legend()
  plt.xlabel("FPR")
  plt.ylabel("TPR")
  plt.show
  return aucs


def exportDataset(path, name, subCuts, numCuts, features, label):   #label
 these
  dataset = np.empty([1, len(features)])
  labels = np.empty((1))

  for i in range(len(subCuts)):
    subSet = np.array([])
    for j in range(len(features)):
```

```
      for k in range(len(numCuts)):
        # data = pd.read_csv('/content/drive/MyDrive/Clean data/Experiment
 4/'+name+ numCuts[k]+ subCuts[i] + features[j] +'AutoSeperated.csv', sep=
",")
        data = pd.read_csv(path+name+ numCuts[k]+ subCuts[i] + features[j]
 +'AutoSeperated.csv', sep=",")
        data = np.reshape(data.values, [-1, 1])
        data = data[~np.isnan(data)]
        data = np.reshape(data, [-1, 1])
        # print(data)
        subSet = np.hstack([subSet, data]) if subSet.size else data

    num_rows, num_cols = subSet.shape

    # print(subSet.shape)

    dataset = np.vstack([dataset, subSet])
    # default is conservative
    multiplier = 1 if subCuts[i] == '_Conservative_' or subCuts[i] == 'C_1
__Conservative_' else 0

    if subCuts[i] == '_Optimal_' or subCuts[i] == 'O_1__Optimal_':
      multiplier = 2
    elif subCuts[i] == '_Aggressive_' or subCuts[i] == 'A_1__Aggressive_':
      multiplier = 3
    # if subCuts[i] == '_Aggressive_':
    #   labels = np.concatenate((labels, np.ones([num_rows, 1])*3))
    values = np.ones([num_rows, 1])*multiplier

    if i == 0 and j == 0:
      labels[:] = values
    else:
      # print(labels.shape)
      # print(values.shape)
      labels = np.concatenate((labels, np.squeeze(values)))

    num_rows, num_cols = subSet.shape
    # if subCuts[i] == '_Conservative_':
    #   print((np.ones([num_rows, 1])*1).shape)
    #   labels = np.concatenate((labels, )
    # if subCuts[i] == '_Optimal_':
    #   labels = np.concatenate((labels, np.ones([num_rows, 1])*2))
    # if subCuts[i] == '_Aggressive_':
    #   labels = np.concatenate((labels, np.ones([num_rows, 1])*3))
```

```
  num_rows, num_cols = dataset.shape
  labels_mat = np.ones([num_rows, 1])*label
  # dataset = np.hstack([dataset, labels])if dataset.size else labels
  return dataset[1:], labels.astype(int)[1:], labels_mat[1:]


#For EXP 4
path = '/content/drive/MyDrive/Clean data/Experiment 4/'
name = 'EXP4_6061_'
numCuts = ['1_']
subCuts = ['_Conservative_', '_Optimal_', '_Aggressive_']
features = ['_Spindle_', '_xLoad_', '_yLoad_', '_zLoad_']
# print(dataset)
dataset6061, labels6061, labels_mat6061 = exportDataset(path, name, subCut
s,numCuts, features, 1)
#EXP4_6061_1__Conservative__Spindle_AutoSeperated
numCuts = ['2_']
dataset60612, labels60612, labels_mat60612 = exportDataset(path, name, sub
Cuts,numCuts, features, 1)


# For exp5

#For EXP 5
path = '/content/drive/MyDrive/Exp 5/Clean/'
name = 'EXP5_6061_'
wdcombo = ['1_'] #1, 2, 3, are different width/depth
frate = ['C_1__Conservative_', 'O_1__Optimal_', 'A_1__Aggressive_']  #only
 1 cut for each
features = ['_Spindle_', '_xLoad_', '_yLoad_', '_zLoad_']
# print(dataset)
dataset60613, labels60613, labels_mat60613 = exportDataset(path, name, fra
te, wdcombo, features, 1)
#EXP4_6061_1__Conservative__Spindle_AutoSeperated
wdcombo = ['2_']
dataset60614, labels60614, labels_mat60614 = exportDataset(path, name, fra
te, wdcombo, features, 1)
wdcombo = ['3_']
dataset60615, labels60615, labels_mat60615 = exportDataset(path, name, fra
te, wdcombo, features, 1)


trainingSet = np.vstack([dataset6061, dataset60612, dataset60613, dataset6
0614, dataset60615])
```

```
labels = np.hstack([labels6061,labels60612, labels60613, labels60614, labe
ls60615])
labels_mat = np.hstack([labels_mat6061.T, labels_mat60612.T, labels_mat606
13.T, labels_mat60614.T, labels_mat60615.T]).T

# trainingSet = np.vstack([dataset6061, dataset60612])
# labels = np.hstack([labels6061,labels60612])
# labels_mat = np.hstack([labels_mat6061.T, labels_mat60612.T]).T

magForce = ((trainingSet[:, 1])**2 +(trainingSet[:, 2])**2+(trainingSet[:,
 3])**2)**.5
trainingSet = np.c_[trainingSet, magForce]


export = pd.DataFrame(trainingSet)
export.to_csv('completeSet_04_23.csv')


spindleCWT = np.array(pd.read_csv('/content/drive/MyDrive/Exp 5/CWT/1_CWTA
llFreqs.csv', sep=",", header=None))
xCWT =   pd.read_csv('/content/drive/MyDrive/Exp 5/CWT/2_CWTAllFreqs.csv',
 sep=",", header=None)
yCWT =   pd.read_csv('/content/drive/MyDrive/Exp 5/CWT/3_CWTAllFreqs.csv',
 sep=",", header=None)
zCWT =   pd.read_csv('/content/drive/MyDrive/Exp 5/CWT/4_CWTAllFreqs.csv',
 sep=",", header=None)
magCWT = pd.read_csv('/content/drive/MyDrive/Exp 5/CWT/5_CWTAllFreqs.csv',
 sep=",", header=None)

binarize = 0;

if binarize:
  spindleCWT = 1 * (spindleCWT>30)
  xCWT = 1*(xCWT > 1)
  yCWT = 1*(yCWT>1)
  zCWT = 1*(zCWT>.1)
  magCWT = 1 * (magCWT>1)


trainingSet = np.c_[trainingSet, spindleCWT, xCWT, yCWT, zCWT, magCWT]
print(trainingSet.shape)


num_cons = np.count_nonzero(labels[:]==1)
num_optimal = np.count_nonzero(labels[:]==2)
```

```
num_agg = np.count_nonzero(labels[:]==3)
print(num_cons)
print(num_optimal)
print(num_agg)
x = labels
n, bins, patches = plt.hist(x, facecolor='blue', alpha=0.5)

plt.xlabel("Materiel")
plt.ylabel("Frequency")

plt.title('Frequency of Safety Levels')


plt.show()
print(n)
print(bins)


# Creating Feature Headers


Feat_Headers = ['Spindle', 'xLoad', 'yLoad', 'zLoad', 'magForce']
for j in range(5):
  for i in range(125): #creates list of features
    if j ==0:
      Feat_Headers = np.hstack([Feat_Headers,'sFreq'+str(i)])
    elif j ==1:
      Feat_Headers = np.hstack([Feat_Headers,'xFreq'+str(i)])
    elif j ==2:
      Feat_Headers = np.hstack([Feat_Headers,'yFreq'+str(i)])
    elif j == 3:
      Feat_Headers = np.hstack([Feat_Headers,'zFreq'+str(i)])
    elif j ==4:
      Feat_Headers = np.hstack([Feat_Headers,'magFreq'+str(i)])

Feat_Headers = np.array(Feat_Headers)
trainingsetDF = pd.DataFrame(trainingSet, columns = Feat_Headers)
cc = trainingsetDF.corr(method='pearson')
```

```
# Feature Down Selection


trainingSetDF = pd.DataFrame(trainingSet)

matrix = trainingSetDF.corr().abs()

mask = np.triu(np.ones_like(matrix, dtype=bool))

reduced_matrix = matrix.mask(mask)
to_drop = [c for c in reduced_matrix.columns if any(reduced_matrix[c] > 0.
85)]
print(to_drop)
trainingSet_Reduced = trainingSetDF.drop(to_drop, axis=1)
print(trainingSet_Reduced.shape)
Feat_Headers_Reduced = np.delete(Feat_Headers, to_drop, axis=0)
trainingSet_Reduced.columns = Feat_Headers_Reduced
MI_score = mutual_info_classif(trainingSet_Reduced, labels)
to_drop_MI = []
to_drop_MI_index = []

# to_drop_MI = [c for c in trainingSet_Reduced.columns if any(MI_score[c]
< 0.15)]
for c in range(len(trainingSet_Reduced.columns)):
  if MI_score[c] < 0.15:                              #make smaller maybe?
    to_drop_MI.append(Feat_Headers_Reduced[c])
    to_drop_MI_index.append(c)
print(to_drop_MI)
trainingSet_Reduced = trainingSet_Reduced.drop(to_drop_MI, axis=1)
print(trainingSet_Reduced.shape)
Feat_Headers_Reduced_MI = np.delete(Feat_Headers_Reduced, to_drop_MI_index
, axis=0)
trainingSet_Reduced.columns = Feat_Headers_Reduced_MI


#Injesting OOE (Out of Experiment/testing data)
path = '/content/drive/MyDrive/Clean data/Experiment 4/'
name = 'CURVY2_6061_' #Test File
numCuts = ['1_']
subCuts = ['_Conservative_', '_Optimal_', '_Aggressive_']
features = ['_Spindle_', '_xLoad_', '_yLoad_', '_zLoad_']
# print(dataset)
dataset6061OOS, labels6061OOS, labels_mat6061OOS = exportDataset(path, nam
e, subCuts,numCuts, features, 1)
#EXP4_6061_1__Conservative__Spindle_AutoSeperated
```

```
path = '/content/drive/MyDrive/Exp 5/Clean/Test/'
name = 'EXP5_6061_T_'
wdcombo = ['1_'] #1, 2, 3, are different width/depth
frate = ['C_1__Conservative_', 'O_1__Optimal_', 'A_1__Aggressive_']  #only
 1 cut for each
features = ['_Spindle_', '_xLoad_', '_yLoad_', '_zLoad_']
# print(dataset)
dataset60613OOS, labels60613OOS, labels_mat60613OOS = exportDataset(path,
name, frate, wdcombo, features, 1)
#EXP4_6061_1__Conservative__Spindle_AutoSeperated
wdcombo = ['2_']
dataset60614OOS, labels60614OOS, labels_mat60614OOS = exportDataset(path,
name, frate, wdcombo, features, 1)
wdcombo = ['3_']
dataset60615OOS, labels60615OOS, labels_mat60615OOS = exportDataset(path,
name, frate, wdcombo, features, 1)

trainingSetOOS = np.vstack([dataset6061OOS, dataset60613OOS, dataset60614O
OS, dataset60615OOS])
labelsOOS = np.hstack([labels6061OOS, labels60613OOS, labels60614OOS, labe
ls60615OOS])
labels_matOOS = np.hstack([labels_mat6061OOS.T, labels_mat60613OOS.T, labe
ls_mat60614OOS.T, labels_mat60615OOS.T]).T

magForceOOS = ((trainingSetOOS[:, 1])**2 +(trainingSetOOS[:, 2])**2+(train
ingSetOOS[:, 3])**2)**.5
trainingSetOOS = np.c_[trainingSetOOS, magForceOOS]

export = pd.DataFrame(trainingSetOOS)
export.to_csv('completeSet_04_23OOS.csv') # spits this out so you can deri
ve frequency feature from matlab

spindleCWT = pd.read_csv('/content/drive/MyDrive/Exp 5/CWT/1_CWTAllFreqsOO
S.csv', sep=",", header=None) #matlab spits out 5 files that you need to d
rag into this directory
xCWT = pd.read_csv('/content/drive/MyDrive/Exp 5/CWT/2_CWTAllFreqsOOS.csv'
, sep=",", header=None)
yCWT = pd.read_csv('/content/drive/MyDrive/Exp 5/CWT/3_CWTAllFreqsOOS.csv'
, sep=",", header=None)
zCWT = pd.read_csv('/content/drive/MyDrive/Exp 5/CWT/4_CWTAllFreqsOOS.csv'
, sep=",", header=None)
magCWT = pd.read_csv('/content/drive/MyDrive/Exp 5/CWT/5_CWTAllFreqsOOS.cs
v', sep=",", header=None)
```

```
if binarize:
  spindleCWT = 1 * (spindleCWT>30)
  xCWT = 1*(xCWT > 1)
  yCWT = 1*(yCWT>1)
  zCWT = 1*(zCWT>.1)
  magCWT = 1 * (magCWT>1)

trainingSetOOS = np.c_[trainingSetOOS, spindleCWT, xCWT, yCWT, zCWT, magCW
T]
print(trainingSetOOS.shape)
#Feature Downselecting

trainingSetOOSDF = pd.DataFrame(trainingSetOOS)

trainingSet_ReducedOOS = trainingSetOOSDF.drop(to_drop, axis=1) #filters t
esting set based on the same filters it used on the training set
trainingSet_ReducedOOS.columns = Feat_Headers_Reduced
trainingSet_ReducedOOS = trainingSet_ReducedOOS.drop(to_drop_MI,axis=1)
trainingSet_ReducedOOS.columns = Feat_Headers_Reduced_MI

print(trainingSet_ReducedOOS.shape)


MI_score = mutual_info_classif(trainingSet_Reduced, labels)
print(MI_score)
# from IPython.core.interactiveshell import InteractiveShell
# InteractiveShell.ast_node_interactivity = "all"

categories = Feat_Headers_Reduced_MI



label_loc = np.linspace(start=0, stop=2 * np.pi-
(2*np.pi)/(len(MI_score)), num=(len(MI_score)))

plt.figure(figsize=(8, 8))
plt.subplot(polar=True)
plt.plot(label_loc[0:], MI_score[0:], label='Aggressiveness')
# plt.plot(label_loc[0:], MI_score_mat[0:], label='Materiel')
plt.title('MI Score comparison', size=20)
lines, labels_rad = plt.thetagrids(np.degrees(label_loc), labels=categorie
s)
plt.legend()
plt.show()
MI_score = mutual_info_classif(trainingSet_ReducedOOS, labelsOOS)
```

```
categories = Feat_Headers_Reduced_MI



label_loc = np.linspace(start=0, stop=2 * np.pi-
(2*np.pi)/(len(MI_score)), num=(len(MI_score)))

plt.figure(figsize=(8, 8))
plt.subplot(polar=True)
plt.plot(label_loc[0:], MI_score[0:], label='Aggressiveness')
# plt.plot(label_loc[0:], MI_score_mat[0:], label='Materiel')
plt.title('MI Score comparison', size=20)
lines, labels_rad = plt.thetagrids(np.degrees(label_loc), labels=categorie
s)
plt.legend()
plt.show()


# OOE Hyperparameter Optimization MLP

from sklearn.utils import shuffle
from sklearn.neural_network import MLPClassifier
from sklearn import preprocessing


X = np.array(trainingSet_Reduced)
y = labels

X, y = shuffle(X, y, random_state=0)

X_test = np.array(trainingSet_ReducedOOS)
yOOS = labelsOOS

# pca = PCA(n_components=4) # maybe 3 to 5?
# pca.fit(X)
                                        #PCA stuff
# X = pca.transform(X)
# X_test = pca.transform(X_test)

scaler = preprocessing.StandardScaler().fit(X)
X = scaler.transform(X)

scaler = preprocessing.StandardScaler().fit(X_test)
X_test = scaler.transform(X_test)
```

```
testHP = list(range(2, 15)) # 2 to 15

min = 0
max = len(testHP)-1

hp_matrix = np.zeros((4, 4, max-min+1))


for i in list(range(min, max+1)):
# for i in testHP:
  print(testHP[i])
  MLP_model_iden = MLPClassifier(activation= 'identity',hidden_layer_sizes
=testHP[i], max_iter=400) #Instatntiate each model with different HP
  MLP_model_log = MLPClassifier(activation= 'logistic',hidden_layer_sizes=
testHP[i], max_iter=400) #best at 0.58 @15
  MLP_model_tanh = MLPClassifier(activation= 'tanh',hidden_layer_sizes=tes
tHP[i], max_iter=400)
  MLP_model_relu = MLPClassifier(activation= 'relu',hidden_layer_sizes=tes
tHP[i], max_iter=400)


  # rfc = cross_validate(rfc_model, X, y, cv=skf, scoring=('recall_weighte
d', 'f1_weighted'))

  MLP_model_iden.fit(X, y) #train and test each model
  Y_pred_iden = MLP_model_iden.predict(X_test)

  MLP_model_log.fit(X, y)
  Y_pred_log = MLP_model_log.predict(X_test)

  MLP_model_tanh.fit(X, y)
  Y_pred_tanh = MLP_model_tanh.predict(X_test)

  MLP_model_relu.fit(X, y)
  Y_pred_relu = MLP_model_relu.predict(X_test)



  # hp_matrix[0,:,i-
min] = [rfc['test_recall_weighted'].mean(), rfc['test_f1_weighted'].mean()
] #Take the mean of all of folds for one label
  hp_matrix[0,:,i-min] = scores(yOOS, Y_pred_iden)
  hp_matrix[1,:,i-min] = scores(yOOS, Y_pred_log)
```

```
  hp_matrix[2,:,i-
min] = scores(yOOS, Y_pred_tanh)    #scores different HP for each of the 4
 activation functions
  hp_matrix[3,:,i-min] = scores(yOOS, Y_pred_relu)

plt.rcParams['figure.figsize'] = [30, 15]   #plots the above scores
fig, axs = plt.subplots(4)

for j in list(range(0, 4)):
  for k in list(range(0, 4)):
    axs[j].plot(testHP, hp_matrix[j,k,:],label = "Accuracy", linestyle="-
.")
    axs[j].plot(testHP, hp_matrix[j,k,:],label = "Precision", linestyle="-
.")
    axs[j].plot(testHP, hp_matrix[j,k,:],label = "Recall", linestyle="-.")
    axs[j].plot(testHP, hp_matrix[j,k,:],label = "F1 Score", linestyle="-
.")
fig.suptitle('MLP HP')


plt.legend()
plt.xlabel('Hidden Layer Size')
plt.ylabel('Scoring')

plt.show()


best = np.amax(hp_matrix, axis=2) # maximum score for each hidden layer si
ze
best2 = hp_matrix.argmax(axis=2)


# OOE Hyperparameter Optimization KNN
from sklearn.utils import shuffle
from sklearn.neural_network import MLPClassifier
from sklearn import preprocessing


X = np.array(trainingSet_Reduced)
y = labels

X, y = shuffle(X, y, random_state=0)

X_test = np.array(trainingSet_ReducedOOS)
yOOS = labelsOOS
```

```
# pca = PCA(n_components=4) # maybe 3 to 5?
# pca.fit(X)
                                #PCA Stuff
# X = pca.transform(X)
# X_test = pca.transform(X_test)

scaler = preprocessing.StandardScaler().fit(X)
X = scaler.transform(X)

scaler = preprocessing.StandardScaler().fit(X_test)
X_test = scaler.transform(X_test)



# testHP = [4, 8, 12, 16, 20, 24, 28, 32, 36, 40]
# testHP = [2, 4]
testHP = [2, 4, 8, 16, 32, 64, 128] #, 256, 512]

# testHP = list(range(2, 100))

min = 0
max = len(testHP)-1

hp_matrix = np.zeros((7, 4, max-min+1))


for i in list(range(min, max+1)):
# for i in testHP:
  print(i)
  KNC_model_3 = KNeighborsClassifier(n_neighbors = 3, leaf_size = testHP[i
])
  KNC_model_4 = KNeighborsClassifier(n_neighbors = 4, leaf_size = testHP[i
])
  KNC_model_5 = KNeighborsClassifier(n_neighbors = 5, leaf_size = testHP[i
])
  KNC_model_6 = KNeighborsClassifier(n_neighbors = 6, leaf_size = testHP[i
])
  KNC_model_7 = KNeighborsClassifier(n_neighbors = 7, leaf_size = testHP[i
])
  KNC_model_8 = KNeighborsClassifier(n_neighbors = 8, leaf_size = testHP[i
])
  KNC_model_9 = KNeighborsClassifier(n_neighbors = 9, leaf_size = testHP[i
])
```

```
  # rfc = cross_validate(rfc_model, X, y, cv=skf, scoring=('recall_weighte
d', 'f1_weighted'))

  KNC_model_3.fit(X, y) #train and test each model
  Y_pred_3 = KNC_model_3.predict(X_test)

  KNC_model_4.fit(X, y) #train and test each model
  Y_pred_4 = KNC_model_4.predict(X_test)

  KNC_model_5.fit(X, y) #train and test each model
  Y_pred_5 = KNC_model_5.predict(X_test)

  KNC_model_6.fit(X, y) #train and test each model
  Y_pred_6 = KNC_model_6.predict(X_test)

  KNC_model_7.fit(X, y) #train and test each model
  Y_pred_7 = KNC_model_7.predict(X_test)

  KNC_model_8.fit(X, y) #train and test each model
  Y_pred_8 = KNC_model_8.predict(X_test)

  KNC_model_9.fit(X, y) #train and test each model
  Y_pred_9 = KNC_model_9.predict(X_test)




  # hp_matrix[0,:,i-
min] = [rfc['test_recall_weighted'].mean(), rfc['test_f1_weighted'].mean()
] #Take the mean of all of folds for one label
  hp_matrix[0,:,i-min] = scores(yOOS, Y_pred_3)
  hp_matrix[1,:,i-min] = scores(yOOS, Y_pred_4)
  hp_matrix[2,:,i-
min] = scores(yOOS, Y_pred_5)    #scores different HP for each of the 4 ac
tivation functions
  hp_matrix[3,:,i-min] = scores(yOOS, Y_pred_6)
  hp_matrix[4,:,i-min] = scores(yOOS, Y_pred_7)
  hp_matrix[5,:,i-min] = scores(yOOS, Y_pred_8)
  hp_matrix[6,:,i-
min] = scores(yOOS, Y_pred_9)    #scores different HP for each of the 4 ac
tivation functions
```

```
best = np.amax(hp_matrix, axis=2) # maximum score for each hidden layer si
ze
best2 = hp_matrix.argmax(axis=2)


# OOE Hyperparameter Optimization Decistion tree and random forest (one at
a time)


from sklearn.utils import shuffle
# skf = StratifiedKFold(n_splits=5, shuffle = True)

X = np.array(trainingSet_Reduced)
y = labels

X, y = shuffle(X, y, random_state=0)

X_test = np.array(trainingSet_ReducedOOS)
yOOS = labelsOOS

# pca = PCA(n_components=4) # maybe 3 to 5?
# pca.fit(X)
                    #PCA stuff
# X = pca.transform(X)
# X_test = pca.transform(X_test)

min = 2
max = 20

hp_matrix = np.zeros((2, 4, max-min))

for i in list(range(min, max)):
  print(i)
  # rfc_model = MLPClassifier(activation = 'relu', hidden_layer_sizes = (i
,), max_iter = 500)
  rfc_model = RandomForestClassifier(random_state=0, max_depth=i, class_we
ight = 'balanced')
  # rfc_model = DecisionTreeClassifier(max_depth=i, class_weight = 'balanc
ed')

  rfc_model.fit(X, y)
  Y_pred = rfc_model.predict(X_test)
```

```
  # hp_matrix[0,:,i-
min] = [rfc['test_recall_weighted'].mean(), rfc['test_f1_weighted'].mean()
] #Take the mean of all of folds for one label
  hp_matrix[1,:,i-
min] = scores(yOOS, Y_pred) #Take the mean of all of folds for one label

# plt.plot(list(range(min, max)), hp_matrix[0,0,:], label = "Accuracy", li
nestyle="-.")
plt.plot(list(range(min, max)), hp_matrix[0,0,:],label = "Recall", linesty
le="-.")
plt.plot(list(range(min, max)), hp_matrix[0,1,:],label = "F1 Score", lines
tyle="-.")

plt.legend()
plt.xlabel('Max Depth')
plt.ylabel('Scoring')

plt.show()

plt.plot(list(range(min, max)), hp_matrix[1,0,:],label = "Accuracy", lines
tyle="-.")
plt.plot(list(range(min, max)), hp_matrix[1,1,:],label = "Precision", line
style="-.")
plt.plot(list(range(min, max)), hp_matrix[1,2,:],label = "Recall", linesty
le="-.")
plt.plot(list(range(min, max)), hp_matrix[1,3,:],label = "F1 Score", lines
tyle="-.")

plt.legend()
plt.xlabel('Max Depth')
plt.ylabel('Scoring')

plt.show()

print(hp_matrix[1,3,:])
best = np.amax(hp_matrix, axis=2) # maximum score for each hidden layer si
ze
best2 = hp_matrix.argmax(axis=2)
```

```
# Fit models with hyperparameters
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC

from sklearn import preprocessing
from sklearn.preprocessing import label_binarize

from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

# X = np.array(trainingSet_Reduced)
X = np.array(trainingSet_Reduced)

y_true = labelsOOS
y = labels


pca = PCA(n_components=2)
pca.fit(X)                  #PCA stuff

X = pca.transform(X)

X, y = shuffle(X, y, random_state=0)



# X_test = trainingSet_ReducedOOS
X_test = trainingSet_ReducedOOS

X_test = pca.transform(X_test)      #PCA stuff

y_test = label_binarize(labelsOOS, classes=[1, 2, 3])



# X = OrdinalEncoder().fit_transform(X)
# y = LabelEncoder().fit_transform(y)
# y = label_binarize(y, classes=[0, 1, 2])
```

```
# Instantiate the machine learning classifiers
log_model = LogisticRegression(max_iter=15000)
KNC_model = KNeighborsClassifier(n_neighbors = 3, leaf_size = 2)
dtr_model = DecisionTreeClassifier(max_depth=5, class_weight = 'balanced')
rfc_model = RandomForestClassifier(random_state=0, max_depth=2, class_weig
ht='balanced')
MLP_model = MLPClassifier(activation = 'relu', hidden_layer_sizes = (2,))
#should be max iter?
# SVC_model = SVC(kernal = 'linear')

# Define the models evaluation function

scaler = preprocessing.StandardScaler().fit(X)

metric_matrix = np.zeros((4, 5))
'''
X : data set features
y : data set target
folds : number of cross-validation folds

'''

log_model.fit(scaler.transform(X), y)  #scales data between 0 and 1
log_pred = log_model.predict_proba(scaler.transform(X_test))

auc_log = ROC_OOE(log_pred, y_test)

KNC_model.fit(X, y)                              #might need to scale
KNC_pred = KNC_model.predict_proba(X_test)

auc_knc = ROC_OOE(KNC_pred, y_test)

dtr_model.fit(X, y)
dtr_pred = dtr_model.predict_proba(X_test)

auc_dtr = ROC_OOE(dtr_pred, y_test)

rfc_model.fit(X, y)
rfc_pred = rfc_model.predict_proba(X_test)

auc_rfc = ROC_OOE(rfc_pred, y_test)

MLP_model.fit(scaler.transform(X), y)
MLP_pred = MLP_model.predict_proba(scaler.transform(X_test))
```

```
auc_mlp = ROC_OOE(MLP_pred, y_test)
matrix = np.vstack([auc_log, auc_knc, auc_dtr, auc_rfc, auc_mlp])  # puts
ROC curve data in a table

models_scores_table = pd.DataFrame({
                                  'Logistic Regression':matrix[0, :],

                                  'K Nearest Neighbors':matrix[1, :],

                                  'Decision Tree':matrix[2, :],

                                  'Random Forest':matrix[3, :],

                                  'MLP Neural Network':matrix[4, :]

                                  # 'Macro Average':matrix.mean(axis=0)

                                  },

                                  index=['Conservative AUC', 'Optimal AUC'
, 'Aggressive AUC'])
# models_scores_table['Macro Average'] = models_scores_table.mean(axis=1)
models_scores_table['Best Score'] = models_scores_table.idxmax(axis=1)
models_scores_table


def ROC_OOE_perlabel(y_predT, y_test):   #makes nice ROC curves to compare
 models

  # classes = np.unique(y_test)
  n_classifiers = 5
  classifiers = ['Logistic Regression', 'K Nearest Neighbors',  'Decision
Tree',  'Random Forest',  'MLP Neural Network']
  classes_ = ["conservative", "optimal", "aggressive"]
  colors = [range(len(classes_))]
  fpr = dict()
  tpr = dict()
  thresholds = dict()

  # ax = plt.subplots()



  for i, c in enumerate(classes_):
```

```
    for j, c in enumerate(classifiers):

      fpr[j], tpr[j], thresholds[j] = roc_curve(y_test[:,i], y_predT[j][:,
 i])
      # roc_auc_score(y, y_predT[:, i])
      plt.figure(i)

      plt.plot(fpr[j],tpr[j], label = [classifiers[j]])
    plt.legend(loc = 'lower right')
    plt.title(classes_[i])
    plt.xlabel("FPR")
    plt.ylabel("TPR")
    plt.show


dict_of_pred = dict() #make dictionary of all outputs
dict_of_pred[0]=log_pred
dict_of_pred[1]=KNC_pred
dict_of_pred[2]=dtr_pred
dict_of_pred[3]=rfc_pred
dict_of_pred[4]=MLP_pred
ROC_OOE_perlabel(dict_of_pred, y_test)


from sklearn.metrics import roc_auc_score
def ROC_OOE_featexplore(y_predT, y_test):     # takes bests models

  # classes = np.unique(y_test)
  n_classifiers = 5
  sets = ['Full (All Features)','Reduced', 'Original Experimentally Collec
ted','Spindle Only']
  classes_ = ["Conservative", "Optimal", "Aggressive"]
  colors = [range(len(classes_))]
  fpr = dict()
  tpr = dict()
  thresholds = dict()

  # ax = plt.subplots()



  for i, c in enumerate(classes_):
    for j, c in enumerate(sets):
```

```
      fpr[j], tpr[j], thresholds[j] = roc_curve(y_test[:,i], y_predT[j][:,
 i])
      AUC = round(roc_auc_score(y_test[:,i], y_predT[j][:, i]), 3)
      # roc_auc_score(y, y_predT[:, i])
      plt.figure(i)

      plt.plot(fpr[j],tpr[j], label = [sets[j]+':'+str(AUC)])
    plt.legend(loc = 'best')
    plt.title(classes_[i])
    plt.xlabel("FPR")
    plt.ylabel("TPR")
    plt.show


from sklearn.utils import shuffle

def scores(y_true, y_predict):
  return [accuracy_score(y_true, y_predict),precision_score(y_true, y_pred
ict, average='weighted') , recall_score(y_true, y_predict, average='weight
ed'), f1_score(y_true, y_predict, average='weighted')]


# Find New hyperparameters for best models for different feature spaces
(spindle only vs reduced vs time domaine ect)


from sklearn.utils import shuffle
# skf = StratifiedKFold(n_splits=5, shuffle = True)
trainingsetDF = pd.DataFrame(trainingSet, columns = Feat_Headers)
trainingSetOOSDF = pd.DataFrame(trainingSetOOS, columns = Feat_Headers)

X = np.array(trainingSet_Reduced) #train
y = labels

X_spindle = trainingsetDF['Spindle']
X_org = trainingsetDF[['Spindle', 'xLoad','yLoad','zLoad']]  #might need P
CA for all but spindle
X_full = trainingSet


y_true = labelsOOS #test

#### Shuffling sets
X, y = shuffle(X, y, random_state=0)
```

```
X_spindle, y_spindle = shuffle(X_spindle, y, random_state=0)
X_spindle = np.array(X_spindle).reshape(-1, 1)

X_org, y_org = shuffle(X_org, y, random_state=0)
X_full, y_full = shuffle(X_full, y, random_state=0)




min = 2 #lower end of range for values to test 2
max = 10 #higher end of range for values to test 20

#Use best models to find parameters for each feature set (full, org, spind
le, reduced)
hp_matrix = np.zeros((1, 4, max-min))
for i in list(range(min, max)):

  print(i)
  #Random Forest
  # rfc_model = RandomForestClassifier(random_state=0, max_depth=i, class_
weight='balanced')
  # rfc_model.fit(X_full, y_full)  # change with different (X_org, y_org e
ct)
  # # Y_pred = rfc_model.predict(trainingSetOOSDF[['Spindle','xLoad','yLoa
d','zLoad']])
  # Y_pred = rfc_model.predict(trainingSetOOS)  #for full

  #Dicision Tree
  rfc_model = DecisionTreeClassifier(max_depth=i, class_weight = 'balanced
')
  rfc_model.fit(X_full, y_full)
  # Y_pred = rfc_model.predict(trainingSetOOSDF[['Spindle','xLoad','yLoad'
,'zLoad']])
  Y_pred = rfc_model.predict(trainingSetOOS)  #for full



  # print(scores(labelsOOS, Y_pred))
  # hp_matrix[0,:,i-
min] = [rfc['test_recall_weighted'].mean(), rfc['test_f1_weighted'].mean()
]
  print(scores(labelsOOS, Y_pred))

  hp_matrix[0,:,i-min] = scores(labelsOOS, Y_pred)
```

```
# plt.plot(list(range(min, max)), hp_matrix[0,0,:], label = "Accuracy", li
nestyle="-.")
# plt.plot(list(range(min, max)), hp_matrix[0,0,:],label = "Accuracy", lin
estyle="-.")
plt.plot(list(range(min, max)), hp_matrix[0,3,:],label = "F1 Score", lines
tyle="-.")

plt.legend()
plt.xlabel('Max Depth')
plt.ylabel('Scoring')

plt.show()


      # Test best models with new hyperparameters with different feature

spaces. Need to change for different models. Some models commented out

from sklearn.ensemble import RandomForestClassifier

from sklearn.svm import SVC

from sklearn import preprocessing
from sklearn.preprocessing import label_binarize

from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

trainingsetDF = pd.DataFrame(trainingSet, columns = Feat_Headers)
trainingSetOOSDF = pd.DataFrame(trainingSetOOS, columns = Feat_Headers)

X = np.array(trainingSet_Reduced) #train      Real OG data (reduced freque
ncy)
y = labels

X_spindle = trainingsetDF['Spindle']                               #spindle pow
er only
X_org = trainingsetDF[['Spindle', 'xLoad','yLoad','zLoad']]   #just record
ed data
X_full = trainingSet                                              #all 80 feat
ures

  #y_test gives 3 label sets, 0 or 1 for if conservative, 0 or 1 if optima
l, 0 or 1 if aggressive
```

```
y_test = label_binarize(labelsOOS, classes=[1, 2, 3])

y_true = labelsOOS #test    for hyperparameters

##PCA
pca = PCA(n_components=2)
# pca_org = PCA(n_components=2)
# pca_full = PCA(n_components=2)

pca.fit(X)
X = pca.transform(X)      #PCA stuff

# pca_org.fit(X_org)        dont do this????
# X_org = pca_org.transform(X_org)

# pca_full.fit(X_full)
# X_full = pca_full.transform(X_full)

#### Shuffling sets
X, y = shuffle(X, y, random_state=0)

X_spindle, y_spindle = shuffle(X_spindle, y, random_state=0)
X_spindle = np.array(X_spindle).reshape(-1, 1)

X_org, y_org = shuffle(X_org, y, random_state=0)
X_full, y_full = shuffle(X_full, y, random_state=0)

# X = OrdinalEncoder().fit_transform(X)
# y = LabelEncoder().fit_transform(y)
# y = label_binarize(y, classes=[0, 1, 2])

# # Instantiate the machine learning classifiers
# KNC_model = KNeighborsClassifier(n_neighbors = 3, leaf_size = 20) #Reduc
ed
# rfc_model = RandomForestClassifier(random_state=0, max_depth=2, class_we
ight='balanced')
# rfc_model = DecisionTreeClassifier(max_depth=4, class_weight = 'balanced
')
rfc_model = LogisticRegression(max_iter=15000)

# KNC_model_full = KNeighborsClassifier(n_neighbors = 3, leaf_size = 20)
# rfc_model_full = RandomForestClassifier(random_state=0, max_depth=8, cla
ss_weight='balanced')
# rfc_model_full = DecisionTreeClassifier(max_depth=3, class_weight = 'bal
anced')
```

```
rfc_model_full = LogisticRegression(max_iter=15000)

# KNC_model_org = KNeighborsClassifier(n_neighbors = 3, leaf_size = 20)
# rfc_model_org = RandomForestClassifier(random_state=0, max_depth=20, cla
ss_weight='balanced')
# rfc_model_org = DecisionTreeClassifier(max_depth=16, class_weight = 'bal
anced')
rfc_model_org = LogisticRegression(max_iter=15000)

# KNC_model_spindle = KNeighborsClassifier(n_neighbors = 3, leaf_size = 20
)
# rfc_model_spindle = RandomForestClassifier(random_state=0, max_depth=14,
 class_weight='balanced')
# rfc_model_spindle = DecisionTreeClassifier(max_depth=15, class_weight =
'balanced')
rfc_model_spindle = LogisticRegression(max_iter=15000)

# SVC_model = SVC(kernal = 'linear')

# Define the models evaluation function



metric_matrix = np.zeros((4, 5))
'''
X : data set features
y : data set target
'''

# print('KNN Reduced')
# KNC_model.fit(X, y) #knn models to try
# KNC_pred = KNC_model.predict_proba(pca.transform(trainingSet_ReducedOOS)
)
# auc_knc = ROC_OOE(KNC_pred, y_test)

# print('KNN Spindle')

# KNC_model_spindle.fit(X_spindle, y_spindle)
# KNC_pred_spindle = KNC_model_spindle.predict_proba(np.array(trainingSet_
ReducedOOS['Spindle']).reshape(-1, 1)) #Spindle only dataset
# auc_knc_spindle = ROC_OOE(KNC_pred_spindle, y_test)


# print('KNN org')
```

```
# KNC_model_org.fit(X_org, y_org)
# KNC_pred_org = KNC_model_org.predict_proba(trainingSetOOSDF[['Spindle',
'xLoad','yLoad','zLoad']])
# auc_knc_org = ROC_OOE(KNC_pred_org, y_test)

# print('KNN full')

# KNC_model_full.fit(X_full, y_full)
# KNC_pred_full = KNC_model_full.predict_proba(trainingSetOOS)
# auc_knc_full = ROC_OOE(KNC_pred_full, y_test)

# RF Models to try

# print('RF Reduced')

# rfc_model.fit(X, y)
# # rfc_pred = rfc_model.predict_proba(pca.transform(trainingSet_ReducedOO
S)) #with PCA
# rfc_pred = rfc_model.predict_proba(trainingSet_ReducedOOS)
    #no PCA
# auc_rfc = ROC_OOE(rfc_pred, y_test)

# print('RF spindle')

# rfc_model_spindle.fit(X_spindle, y_spindle)
# rfc_pred_spindle = rfc_model_spindle.predict_proba(np.array(trainingSetO
OSDF['Spindle']).reshape(-1, 1))
# auc_rfc_spindle = ROC_OOE(rfc_pred_spindle, y_test)

# print('RF org')

# rfc_model_org.fit(X_org, y_org)
# rfc_pred_org = rfc_model_org.predict_proba((trainingSetOOSDF[['Spindle',
 'xLoad','yLoad','zLoad']]))
# auc_rfc_org = ROC_OOE(rfc_pred_org, y_test)
# print('RF full')

# rfc_model_full.fit(X_full, y_full)
# rfc_pred_full = rfc_model_full.predict_proba((trainingSetOOS))
# auc_rfc = ROC_OOE(rfc_pred_full, y_test)


print('Logistic Regression Reduced')

rfc_model.fit(X, y)
```

```
rfc_pred = rfc_model.predict_proba(pca.transform(trainingSet_ReducedOOS))
#with PCA
# rfc_pred = rfc_model.predict_proba(trainingSet_ReducedOOS)
    #no PCA
auc_rfc = ROC_OOE(rfc_pred, y_test)

print('Logistic Regression spindle')

rfc_model_spindle.fit(X_spindle, y_spindle)
rfc_pred_spindle = rfc_model_spindle.predict_proba(np.array(trainingSetOOS
DF['Spindle']).reshape(-1, 1))
auc_rfc_spindle = ROC_OOE(rfc_pred_spindle, y_test)

print('Logistic Regression org')

rfc_model_org.fit(X_org, y_org)
rfc_pred_org = rfc_model_org.predict_proba((trainingSetOOSDF[['Spindle', '
xLoad','yLoad','zLoad']]))
auc_rfc_org = ROC_OOE(rfc_pred_org, y_test)
print('Logistic Regression full')

rfc_model_full.fit(X_full, y_full)
rfc_pred_full = rfc_model_full.predict_proba((trainingSetOOS))
auc_rfc = ROC_OOE(rfc_pred_full, y_test)
dict_of_pred = dict() #make dictionary of all outputs #  sets = ['Full','R
educed', 'Original Collected','Spindle Only']
dict_of_pred[0]=KNC_pred_full
dict_of_pred[1]=KNC_pred
dict_of_pred[2]=KNC_pred_org
dict_of_pred[3]=KNC_pred_spindle
ROC_OOE_featexplore(dict_of_pred, y_test)
#important plots
dict_of_pred = dict() #make dictionary of all outputs #  sets = ['Full','R
educed', 'Original Collected','Spindle Only']
dict_of_pred[0]=rfc_pred_full
dict_of_pred[1]=rfc_pred
dict_of_pred[2]=rfc_pred_org
dict_of_pred[3]=rfc_pred_spindle
ROC_OOE_featexplore(dict_of_pred, y_test)
```

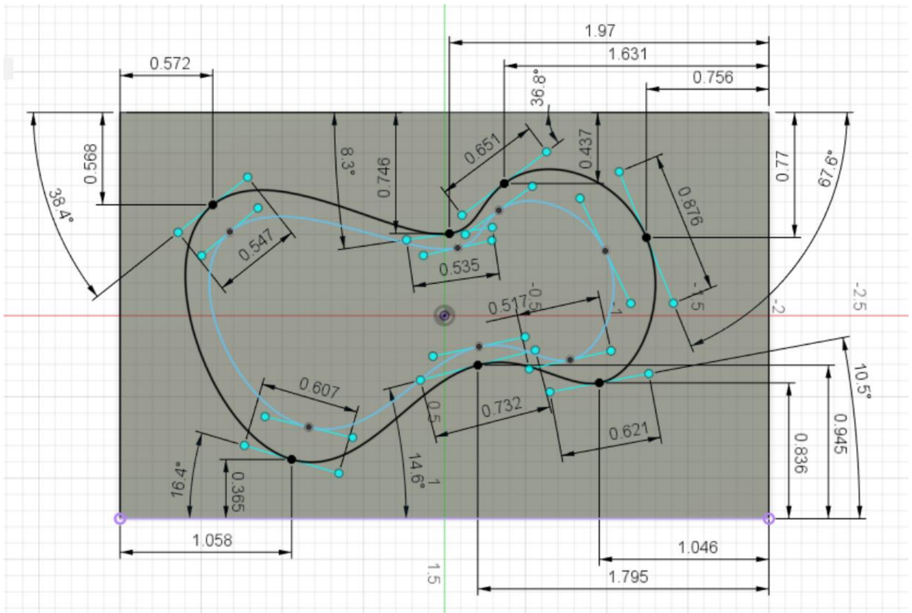
B.3.1. MATLAB Code to Preform Continuous Wavelet Transform

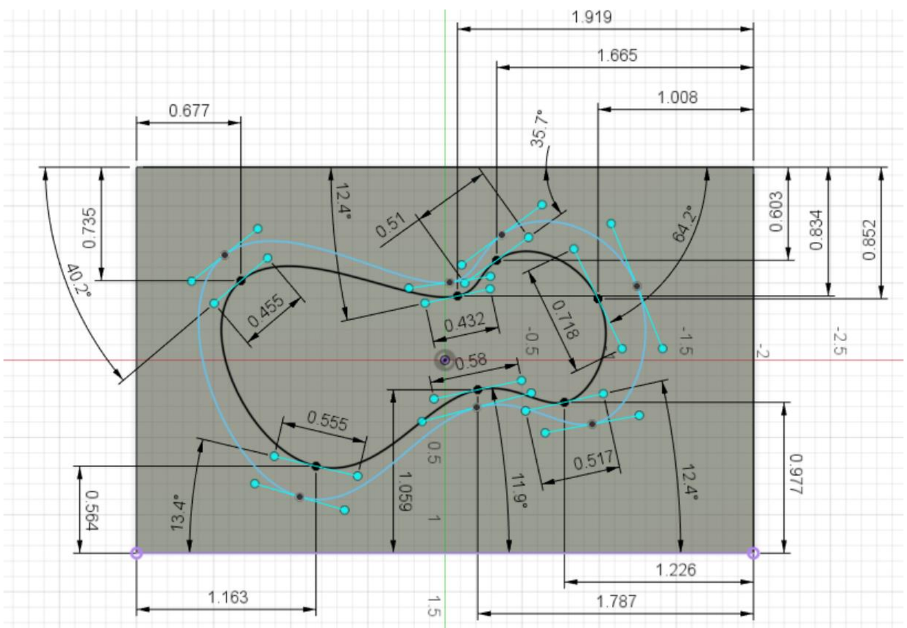
```
close all
figure(1)
data = readmatrix('completeSet_04_23OOS.csv');
```

```
[s, w, t] = cwt(data(:, 1), 166);
% cwt(data(:, 1), 'morse', 166);
s = abs(s);

shape = size(s);

sbinarize = zeros(shape(1), shape(2));

for i = 1:5
    [s, w, t] = cwt(data(:, i), 'morse', 166);
    s = abs(s(1:125, :));
    numel(w(1:125))
    strans = s';
    size(strans)
    writematrix(strans, append(string(i), '_CWTAllFreqsOOS.csv')) %add OOS to end if
needed
%     figure
%     plot(1:42164, strans(:, 121))
end

titles = [{'Spindle'}, {'x'}, {'y'}, {'z'}, {'mag'}];
for i =1:5
    figure
    cwt(data(:, i), 'morse', 166);
    title(titles{i})
end
```

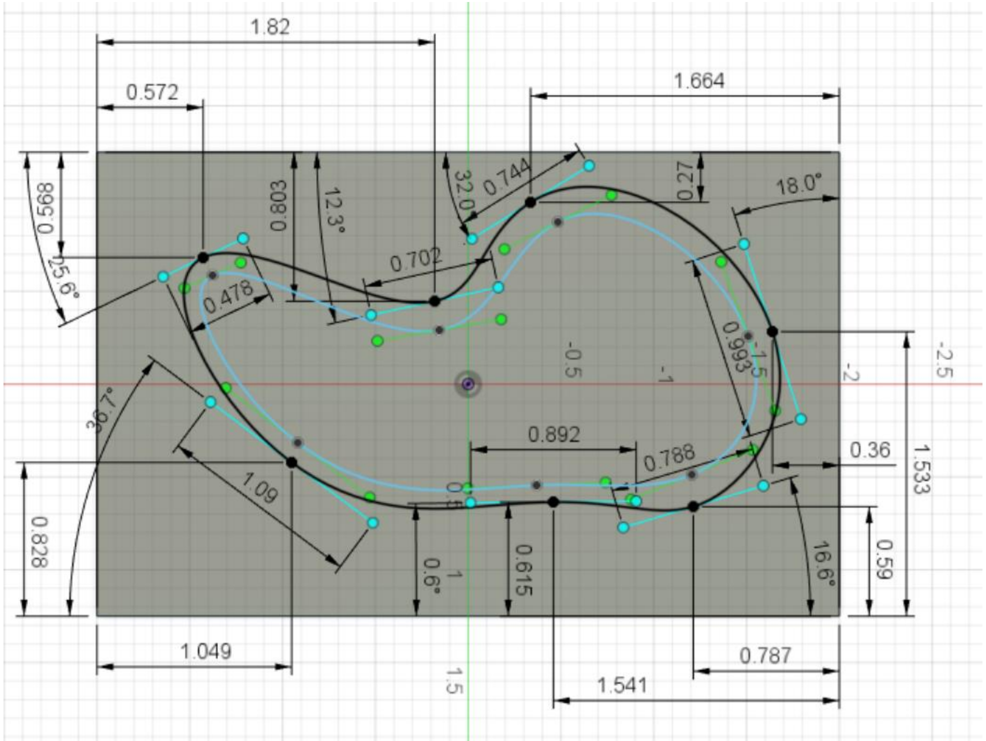
Appendix C

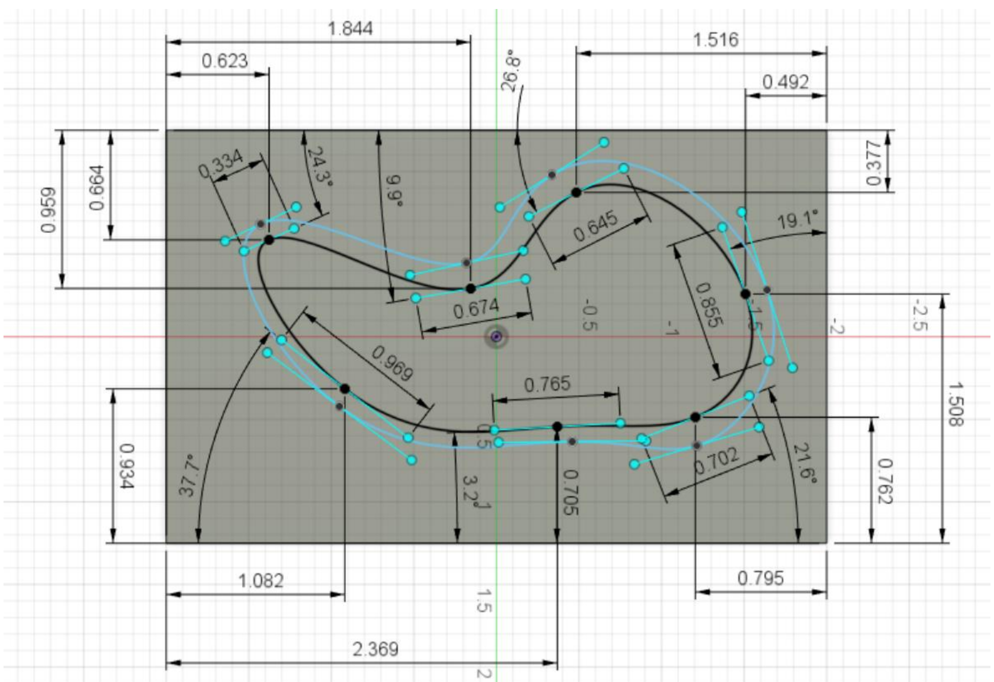
C.1 Dimensions for Training Geometry





C.2 Dimensions for Testing Geometry





References

"About Vericut." *CGTech,* https://cgtech.com/products/about-vericut.html.

Albert, Mark. "MT Connect Is For Real." Modern Machine Shop, vol. 82, no. 7, 2009, pp. 60–68.

Altintas, Y. Spence, A. Tlusty, J. "End Milling Force Algorithms for CAD Systems." CIRP Annals - Manufacturing Technology, vol. 40, no. 1, 1991, pp. 31–34.

Altintas, Y., and Z. M. Kilic. "Generalized Dynamic Model of Metal Cutting Operations." *CIRP Annals*, vol. 62, no. 1, Jan. 2013, pp. 47–50.

Bradley, Andrew P. "The Use of the Area under the ROC Curve in the Evaluation of Machine Learning Algorithms." *Pattern Recognition*, vol. 30, no. 7, 1997, pp. 1145–1159., https://doi.org/10.1016/s0031-3203(96)00142-2.

Brecher, C., Eckel, H. M., Motschke, T., Fey, M., & Epple, A. (2019). Estimation of the virtual workpiece quality by the use of a spindle-integrated process force measurement. *CIRP Annals*, *68*(1), 381-384. DiGangi, Elizabeth A., and Megan K. Moore. *Research Methods in Human Skeletal Biology*. Elsevier, 2013.

Chalmers, Ray. "High-Speed Machining Means More than Quick Spindles." Automotive Design & Production, vol. 131, no. 10, 2019, pp. 34-36. ProQuest,

Cho, S., Asfour, S., Onar, A., & Kaundinya, N. (2005). Tool breakage detection using support vector machine learning in a milling process. *International Journal of Machine Tools and Manufacture*, *45*(3), 241-249.

Denkena, B., Bergmann, B., & Witt, M. (2019). Material identification based on machine-learning algorithms for hybrid workpieces during cylindrical operations. *Journal of Intelligent Manufacturing*, *30*(6), 2449-2456.

Denkena, Berend, and Haythem Boujnah. "Feeling Machines for Online Detection and Compensation of Tool Deflection in Milling." CIRP Annals - Manufacturing Technology, vol. 67, no. 1, 2018, pp. 423–426.

Ertekin, Yalcin M, et al. "Identification of Common Sensory Features for the Control of CNC Milling Operations under Varying Cutting Conditions." International Journal of Machine Tools and Manufacture, vol. 43, no. 9, 2003, pp. 897–904.

Ferguson, M., Bhinge, R., Park, J., Lee, Y. T., & Law, K. H. (2018). A data processing pipeline for prediction of milling machine tool condition from raw sensor data. *Smart and sustainable manufacturing systems*, *2*.

Green, David M.; Swets, John A. (1966). *Signal detection theory and psychophysics.* New York, NY: John Wiley and Sons Inc. *ISBN 978-0-471-32420-1*.

Guyon, I. and Elisseeff, A. "An introduction to variable and feature selection," *Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003.

"G-Wizard CNC Speeds and Feeds Calculator for Milling Machines." *CNCCookbook,* 16 June 2022, https://www.cnccookbook.com/g-wizard-feeds-speeds-calculator-mill/.

Hoo ZH, Candlish J, Teare D. "What is an ROC Curve" Emerg Med J2017;34:357–359.

Isakov Edmund Isakov, Edmund. "Understanding Tangential Cutting Force When Milling." *Cutting Tool Engineering,* Cutting Tool Engineering , 1 May 2012, https://www.ctemag.com/news/articles/understanding-tangential-cutting-force-when-milling.

Joby, Amal. "What Is K-Nearest Neighbor? an ML Algorithm to Classify Data." *Learn Hub,* 19 July 2021, https://learn.g2.com/k-nearest-neighbor.

Keller, Ralph. "What's the Future of U.S. Manufacturing?" Industry Week, vol. 258, no. 10, 2009, p. 10.

Koike, Ryo, et al. "A Sensorless Approach for Tool Fracture Detection in Milling by Integrating Multi-Axial Servo Information." CIRP Annals - Manufacturing Technology, vol. 65, no. 1, 2016, pp. 385–388.

Kurihara, Daisuke, et al. "Cutting Force Control Applying Sensorless Cutting Force Monitoring Method." *Journal of Advanced Mechanical Design, Systems, and Manufacturing*, vol. 4, no. 5, 2010, pp. 955–65.

Liu, Ziye, and Yuebin Guo. "A Hybrid Approach to Integrate Machine Learning and Process Mechanics for the Prediction of Specific Cutting Energy." CIRP Annals - Manufacturing Technology, vol. 67, no. 1, 2018, pp. 57–60.

Lou, Nicole, and Katie Peek Data. *Visualization Forensic Science Hackerspaces . "By the Numbers: The Rise of the Makerspace."* Popular Science, 23 Feb. 2016,

Lutz, B., Kisskalt, D., Mayr, A. *et al.* In-situ identification of material batches using machine learning for machining operations. *J Intell Manuf* **32,** 1485–1495 (2021). https://doi.org/10.1007/s10845-020-01718-3

Maurel, A, et al. "Milling Process FEM Simulation for Identification of Material Parameters Directly from Experiments." AIP Conference Proceedings, vol. 908, no. 1, 2007, pp. 251–256.

Menzies, Tim. Kocagüneli, Ekrem. Minku, Leandro. Peters, Fayola. Turhan, Burak. Chapter 24 - Using Goals in Model-Based Reasoning, Sharing Data and Models in Software Engineering, Morgan Kaufmann, 2015, Pages 321-353, ISBN 9780124172951,

Min Jou, et al. "Development of an Interactive E-Learning System to Improve Manufacturing Technology Education." *Fifth IEEE International Conference on Advanced Learning Technologies (ICALT'05)*, 2005, pp. 359–60.

Nettleton, David. "Selection of Variables and Factor Derivation." *Commercial Data Mining,* 2014, pp. 79–104., https://doi.org/10.1016/b978-0-12-416602-8.00006-6.

Noble, David F. (1984), *Forces of Production: A Social History of Industrial Automation,* New York, New York, USA: Knopf, ISBN 978-0-394-51262-4, LCCN 83048867.

NSF. “Kennametal ISO9001 Quality Certification.” *Https://www.kennametal.com/us/en/about-Us/certificates.html*, 6 Dec. 2018,

Olson, David L.; and Delen, Dursun (2008); *Advanced Data Mining Techniques*, Springer, 1st edition (February 1, 2008), page 138, ISBN 3-540-76916-1

Practical treatise on milling and milling machines. Brown & Sharpe Manufacturing Company. 1914. Retrieved 2013-01-*28.*

Radzevich, Stephen P. (2012-04-02). *Dudley's Handbook of Practical Gear Design and Manufacture, Second Edition*. CRC Press. p. 694. ISBN 978-1-4398-6601-6.

Saidi, Rania, et al. “Hybrid Feature Selection Method Based on the Genetic Algorithm and Pearson Correlation Coefficient.” *Machine Learning Paradigms: Theory and Application*, 2018, pp. 3–24., https://doi.org/10.1007/978-3-030-02357-7_1.

Somvanshi, M., Chavan, P., Tambade, S., & Shinde, S. V. (2016, August). A review of machine learning techniques using decision tree and support vector machine. In *2016 international conference on computing communication control and automation (ICCUBEA)* (pp. 1-7). IEEE.

Spackman, Kent A. (1989). *"Signal detection theory: Valuable tools for evaluating inductive learning"*. Proceedings of the Sixth International Workshop on Machine Learning. San Mateo, CA: *Morgan Kaufmann*. pp. 160–163.

Teti, R. & Jemielniak, Krzysztof & O’Donnell, G.E. & Dornfeld, David. (2010). Advanced monitoring of machining operations. CIRP Annals - Manufacturing Technology. 59. 717-739. 10.1016/j.cirp.2010.05.010.

Tourassi, Georgia. Frederick, Erik. Markey, Mia. and Carey, Floyd Jr., “Application of the mutual information criterion for feature selection in computer-aided diagnosis,” *Medical Physics*, vol. 28, no. 12, pp. 2394–2402, 2001.

U.S. Bureau of Economic Analysis, Gross Domestic Product by Industry: Private Industries: Manufacturing for United States [USMANNQGSP], retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/USMANNQGSP, May 5, 2020.

Vincent, Hannah, et al. “Trojan Detection and Side-Channel Analyses for Cyber-Security in Cyber-Physical Manufacturing Systems.” *Procedia Manufacturing*, Elsevier, 21 Oct. 2015

Wang, Lihui, et al. “Remote Real-Time CNC Machining for Web-Based Manufacturing.” Robotics and Computer Integrated Manufacturing, vol. 20, no. 6, 2004, pp. 563–571.

Woodbury, Robert S. (1972) [1960], *History of the Milling Machine. In Studies in the History of Machine Tools*, Cambridge, Massachusetts, USA, and London, England: MIT Press, ISBN 978-0-262-73033-4, LCCN 72006354. First published alone as a monograph in 1960.

Zhao, Y., et al. “Integration of CAD and a Cutting Tool Selection System.” *Computers & Industrial Engineering*, vol. 42, no. 1, Apr. 2002, pp. 17–34.

Zweig, Mark H.; Campbell, Gregory (1993). *"Receiver-operating characteristic (ROC) plots: a fundamental evaluation tool in clinical medicine"*. Clinical Chemistry. **39** (8): 561–577. *doi*:*10.1093/clinchem/39.4.561*. *PMID 8472349*.