WPI

Impact of Heat-Treatment on Carbon Steel Alloys

A study on the microstructure and mechanical properties of heat-treated steel

4/27/2022

By Students:

Wenlan Fan

Samuel Furman

Dawson Scheid

Daniel Tengdin

Advisors:

Professor Brajendra Mishra

Professor Jianyu Liang

Professor Walter Thomas Towner

Professor Stephen John Bitar

A Major Qualifying Project Proposal submitted to the faculty of
Worcester Polytechnic Institute in partial fulfillment of the requirements of the
Degree of Bachelor of Science

This report represents the work of WPI undergraduate students submitted to the faculty as evidence of completion of the degree requirement. WPI routinely publishes these reports on the web without editorial or peer review. For more information about the project's program at WPI, please see http://wpi.edu/academics/ugradstudies/project-learning.html

Abstract

This MQP is part of a larger project for the Department of Defense's Strategic Environmental Research and Development Program. This project seeks to reduce metal waste generated at US military forward operating bases. In partnership with other research laboratories, WPI's Center for Heat Treating Excellence is developing a 3D-printing-enabled investment casting manufacturing process to allow soldiers to make replacement parts. To support this initiative, our MQP team was tasked with evaluating the metallurgical properties of heat-treated carbon steels. We conducted multiple heat treatment operations on two steel alloys, which provided a set of samples with known hardness and grain structure data. Our team was also tasked with creating an improved furnace control system. This was accomplished by prototyping a custom circuit board and control interface. Additionally, our team conducted an axiomatic design decomposition, which broke down those two tasks into their respective design matrices, resulting in improved task efficiency.

Acknowledgments

We would like to thank Professor Brajendra Mishra and Professor Jianyu Liang for providing this project for the team to participate in. We would also like to thank them for guiding us to be on track with all processes. We are also grateful to have Professor Walter Thomas Towner and Professor Stephen John Bitar as co-advisors for this project. We would also like to thank our teaching assistant Yutao Wang for laboratory and technical support. Finally, we would like to thank Michael Collins for granting us access to the facilities and tools in the Washburn shops.

Table of Contents

Table of Figures

Table of Tables

Executive Summary

As a sub-project under a larger initiative for the Department of Defense's Strategic Environmental Research and Development Program (DoD SERDP), our goal was to increase the utility of furnace equipment on forward operating bases, and the experience of the individuals using them. In order to achieve this, the team divided the project into three objectives to accomplish. The first objective was to heat treat two carbon steel alloys, 8630 and 1018, in order to collect more imagery data for a deep learning system which then will help to guide soldiers through heat treatment operations. The second objective was to improve the furnace control system by implementing a custom circuit and program, increasing accuracy and usability. The third objective was to utilize axiomatic design to optimize the processes of the other two objectives.

In pursuit of these objectives, the team performed quenching and tempering on samples of 8630 steel alloy. For 1018 steel, the team performed normalizing and annealing. These samples were mounted in polyurethane pucks, hardness tested, etched, and image analyzed. These images and data were uploaded to the deep learning model, increasing both its accuracy in prescribing heat treatment procedures, and the range of steel alloys it can prescribe operations for. The furnace control system was improved by replacing the current controller module with a microcontroller, custom circuit, and laptop. Then an axiomatic design analysis was conducted by creating a process flow chart and economic analysis for both of the other two objectives.

1 Introduction

This MQP is a component part of a larger project for the Department of Defense's Strategic Environmental Research and Development Program (DoD SERDP), led in part by our project advisors, Professor Brajendra Mishra, and Professor Jianyu Liang. The SERDP initiative seeks to bring metalworking capabilities to soldiers in forward operating bases (FOBs), allowing them to manufacture replacement parts for equipment that breaks in the field. This initiative addresses two DoD objectives simultaneously by developing a means of recycling scrap steel on military installations, as well as alleviating logistics problems on both a tactical and strategic level. At the tactical level, soldiers can manufacture "good enough" replacement parts for their equipment and facilities, reducing the impact of faulty equipment on missing effectiveness. At the strategic level, it saves significant resources from being squandered on shipping new equipment up to the front.

To guide soldiers in selecting the proper heat treatment procedures (temperatures, times, etc.), a convolutional neural network (CNN) model is being developed that is able to calculate and prescribe heat treatment procedures to the customer based on what the part's function will be, and the chemical composition of the available steel. The CNN model is a deep learning system, which means that when given the heat treatment cycle performed on a particular piece (ex. tempered at 400℃ for 2 hours), the chemical composition of that steel, and an image of the steel's crystal microstructure, it can predict the hardness of the steel. When presented with these data points, the CNN model will learn and adjust its future predictions to be more in-tune with the reality of what it has been presented. In basic terms, the more data that is given to the model, the more accurate the model's predictions will be.

1.1 Problem Statement and Objectives

That is where our MQP comes in. We were seeking to improve the experience of the individuals using the foundries and furnaces, and the utility of the equipment. In pursuit of this goal, we developed two technical objectives for our project: to provide quality data points to the existing CNN model, and to create a user-friendly furnace control interface. Additionally, we pursued a third, overarching objective of utilizing axiomatic design to streamline and optimize the processes of the two technical objectives. See below in Figure 1 a flowchart of the three objectives of our project, and how they interact.

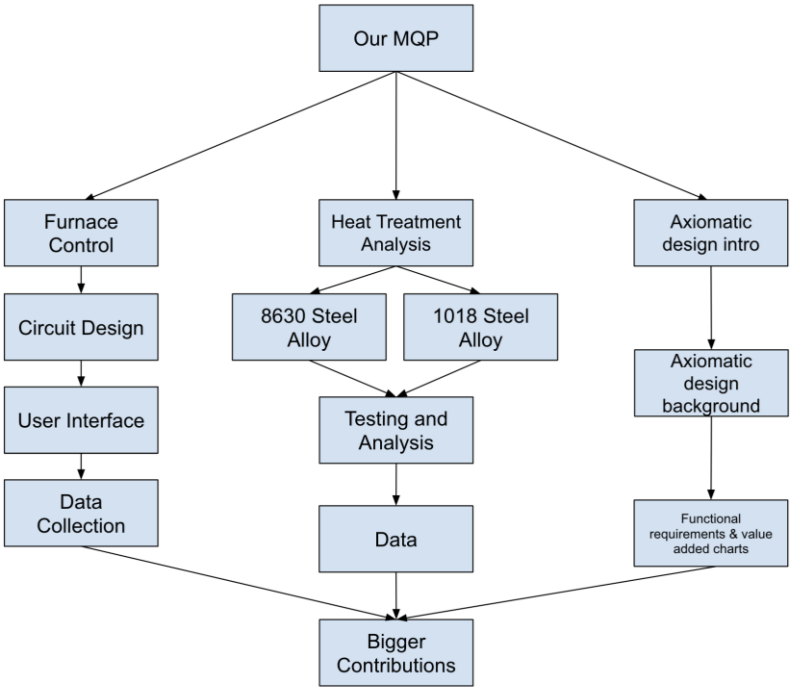

Figure 1. Overall flow chart of the whole project.

1.1.1 Acquisition of Data Points for CNN Model

This objective was to provide the CNN model with numerous solid data points for several different compositions of steel after having undergone various heat treatment operations. Our team's efforts towards this started with background research on various heat treatment processes, formulas, and diagrams, in order to better understand what goes into heat treatment operations. Next, we developed an experimental procedure to perform those heat treatment operations on steel samples, which we then executed. Afterward, we evaluated the hardness of the samples, and captured images of their microstructures. These images and data were then uploaded to the CNN model, becoming part of its database for future use. The desired end-state for the CNN model is that a soldier will be able to enter the steel composition of their manufactured replacement part, and what mechanical properties it will need to accomplish its job, and the CNN model will tell them what heat treatment cycle(s) to subject it to. This will allow soldiers to innovate and fix problems that have arisen on the front lines with self-made, heat-treated steel products.

1.1.2 Improved Furnace Control

One of the key aspects of any heat treatment process is maintaining an accurate temperature in the oven in which the sample is being heated. The user interface panel that came with the furnace consisted of only four buttons and two seven-segment display modules to both program the furnace and display the current temperature. This configuration, though relatively simple, was unintuitive to use and did not allow for a friendly and efficient use experience. To fix these issues, we prototyped an alternative control system utilizing an Arduino microcontroller and a laptop as an interface to allow for accurate, real-time temperature analysis and to provide a visually appealing and easy-to-use control panel to the furnace operator.

1.1.3 Axiomatic Design

To better achieve the aforementioned objectives, we used the principles of axiomatic design to perform a process flow analysis and an economic analysis. Process flow helped our project by showing us the most efficient manner to complete these objectives. We performed this by setting out the different steps of the heat treatment process and comparing how the steps of this process interact. Then we organized the process so that we completed steps that interact before they affect future steps. Our economic analysis justifies this project by showing how our different objectives as a whole adds value. We also look at how this project could have been done but show that our project was more effective in the manner that we completed it.

1.2 Report Overview

In pursuit of the objectives outlined above, we first conducted extensive background research on the larger DoD initiative our project was a part of, the history and nature of heat treatment operations, furnace control systems and interfaces, and axiomatic design. We then developed experimental procedures to guide our efforts toward collecting steel samples and developing a better user interface. Our results section highlights the data gathered from our experimental procedures, including steel hardness data and images, the effectiveness of the CNN model, the improved furnace user interface, and an economic analysis of our efforts as a whole. Finally, we summarize our findings, discuss the project completion, make recommendations to groups going forward, and reflect on our group experience within this MQP as a whole.

2 Background

2.1 The SERDP Initiative

When a piece of a soldier's equipment gets broken on the front lines, it takes an inordinate amount of work hours, money, and planning to get replacement parts to them. This has both tactical and strategic level consequences by blunting mission effectiveness of the unit and wasting significant DoD resources, respectively. In an effort to minimize the logistics involved with replacing and/or fixing equipment, the DoD's SERDP funded a project to provide front line soldiers with the foundry and furnace equipment necessary to fix and create items on their own. Our MQP advisors, Professor Jianyu Liang, and Professor Brajendra Mishra, serve in leadership roles for that project (8). If properly implemented, the addition of this metalworking equipment will alleviate a massive chunk of the front lines' logistics backup. In addition, this will enable soldiers to solve problems stemming from faulty/broken parts of their own initiative, instead of relying on logistics to replace it, making units at the front more self-reliant.

This project also simultaneously addresses another issue SERDP has identified, which pertains to improper waste material disposal in forward areas. While things like wood and paper can biodegrade, there is currently no program in place to reuse/recycle spent metal products, specifically ferrous metals (steel, iron, etc.). This problem of metal recycling can be addressed while also providing soldiers the opportunity to fix their equipment and parts.

However, the average soldier is not an expert in material science. While heat treatment procedures and the formulas and the science behind them are common knowledge to a student or professor who studies these topics, or to an industry expert who collaborates with them daily, they are not readily accessible to soldiers in the field. This is the main purpose of the CNN model discussed in the intro; to take on the role of a technical subject matter expert and do the research

and calculations for heat treatment procedures on behalf of the individual utilizing it. With this approach, soldiers gain the benefit of access to metalworking facilities while mitigating the drawback of lacking technical knowledge.

2.2 The Heat-Treating Process

The reason the heat-treating process is important for metals is that it is able to increase the strength of metals or achieve other desired characteristics such as improve its ductility (5). Heat treating takes a metal and changes the molecular structure by heating the material to a critical point and then cooling it with various cooling speeds. This takes advantage of the fact that metals and alloys have a more fluid molecular structure when they are heated. The common methods of heat treating are quenching, tempering, normalizing, and annealing.

The quenching process starts off by heating an alloy to its critical temperature and then abruptly cooling it down in either water or oil. The critical temperature is the point at which the microstructure of the alloy becomes uniform and miscible in all proportions. This produces a steel with the smallest possible microstructure crystals and is therefore its hardest possible state. The tempering process follows after a material is quenched and pulls back the hardness level while increasing the material's ductility. This occurs when a quenched sample is reheated and held at a less-than-critical temperature, allowing the crystalline structure to relax. Normalizing is the process of cooling a heated sample—either at the critical temperature, or at a lower one, in the case of tempering—in the open air, coming to a stop when the sample reaches room temperature. This produces softer steel than quenching, as it allows the crystalline microstructure to develop for a longer period of time, resulting in larger crystals. Annealing is similar to normalizing, but instead of cooling in the open air, you allow the sample to cool inside of the

furnace. This allows the microstructure crystals to grow for the longest possible time, producing the softest possible steel.

Through studying the effects of heat treatment on the properties of steel, methods have been developed for predicting the outcome of a heat-treating cycle. One such method is the Hollomon-Jaffe parameter, an equation relating temperature and time to a specific parameter for the given steel. One can use this parameter to decide if there is a different temperature their specific heat-treating operation can be conducted that results in a quicker treatment time. Below is a screenshot of the equation, where H is the Hollomon-Jaffe parameter, T is the temperature in degrees Kelvin, t is the time in hours, and C is a constant, determined by the carbon content of the steel (21).

$$H = T[C + log(t)] \,/\, 1000$$

In addition to the Holloman-Jaffe parameter, there are several types of phase diagrams that can be used to predict the composition and properties of a steel sample being subjected to heat treatment. The continuous cooling transformation (CCT) diagram is one such phase diagram. By measuring the starting temperature and the steel piece's rate of cooling, one can determine what its phase composition will be, and therefore its physical properties, such as hardness, ductility, etc. Below is an example of a CCT diagram for an unspecified steel:

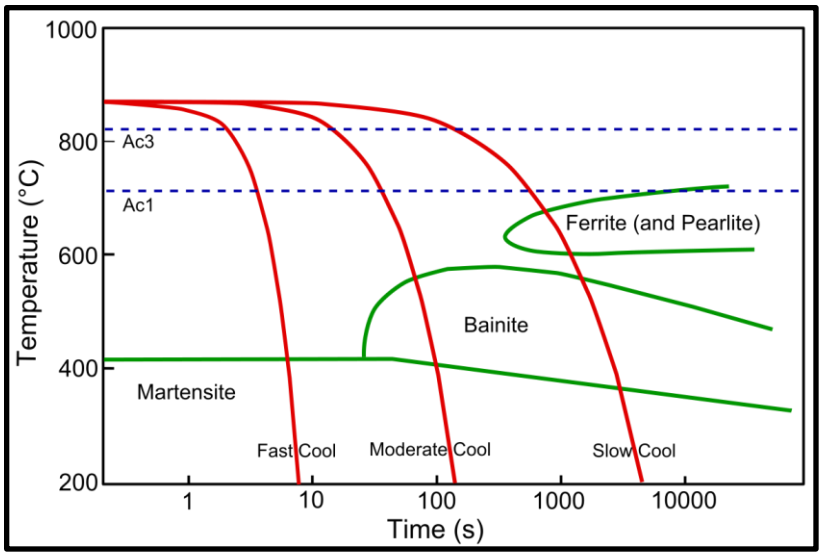

Figure 2. Sample CCT Diagram. (7)

Another such phase diagram is the time-temperature-transformation (TTT) diagram. Also called the isothermal transformation diagram, it instead works by selecting a single target temperature for cooling, and the length of time held at that temperature. Below is an example of a TTT diagram for an unspecified steel:

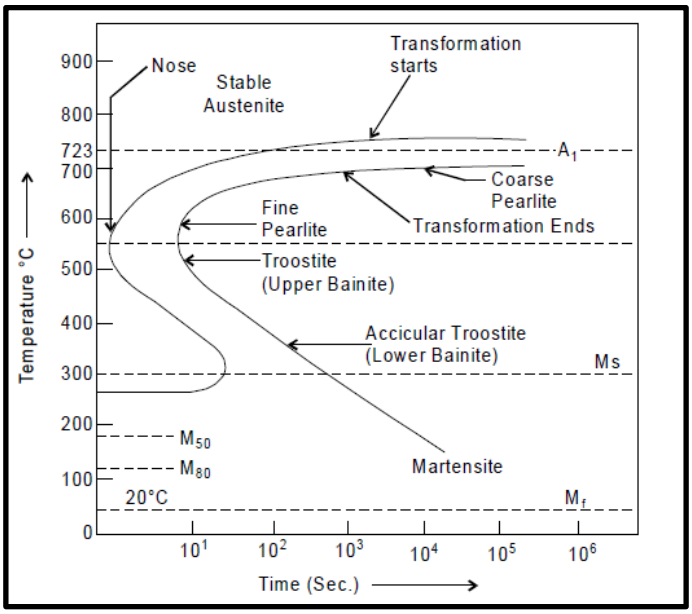

Figure 3. Sample TTT Diagram. (19)

While both of these diagram types serve a purpose, the CCT diagram is more useful. Not only can it be used to determine hardness and microstructure of a piece (which TTT cannot), it is also more user-friendly; for a TTT diagram, the piece needs to be rapidly cooled to a single temperature, and then held there, whereas for a CCT diagram you do not always need to rapidly cool the piece that quickly (11).

2.3 Background of the data analysis tool

As previously mentioned in the introduction, the team used a software model to further analyze all the data. The software was developed based on the Convolutional Neural Network (CNN) model (4). The ideology behind the whole software model is that because the properties of the metal are related to its microstructures, analyzing these microstructures will help the soldier to discover which type of metal it actually is. However, the microstructures are hard to distinguish by the untrained majorities. By using the CNN, it will be able to study the microstructures of the metal sample and then find out its heat treatment parameters along with the mechanical properties (4).

2.4 Temperature Control and Furnace Operation

The ability to accurately control the temperature during any heat treatment process is crucial for obtaining the desired metallurgical properties of an alloyed steel. Under or overshooting the desired temperature may result in either incomplete metallurgic or undesired phase transformations (22). Furnaces must be able to manage various operating conditions and change between them efficiently (6). Operating conditions include loading or removing a workpiece from the furnace and heating or cooling to specific temperatures to manage these changes, each furnace is equipped with a control system which includes a temperature sensor, set-point

programmer, and a controller. The controller uses the temperature sensor to read the temperature inside the furnace, comparing that value to the set point temperature, and makes the necessary adjustments to the heating elements to bring the two values to a matching state.

There are four common types of temperature sensors used in typical applications: thermocouples, thermistors, semiconductor based integrated circuits, and resistance temperature detectors. The thermocouple is the most commonly used type of temperature sensor and operates by using the Seebeck effect, where two different metal wires are joined together and the temperature difference between the two metals causes a voltage difference. This voltage difference between the two metals can be measured and used to calculate the temperature difference (1). Heat treatment ovens use thermocouples as their temperature sensor due to their quick response times and wide temperature ranges. Depending on the type of thermocouple used, temperatures ranging from -270°C to 2000°C can be measured. Table 1 below shows the properties of common thermocouple types.

Table 1. Thermocouple Types and Characteristics Modified from "Tutorial 6500 Temperature Sensor Tutorial" (9)

Code Type	Conductors Alloys (+/-)	Temperature Ranges	Sensitivity (µV/°C)
K	Nickel Chromium / Nickel Aluminum	-180 to 1300°C	41
B	Platinum / Rhodium	0 to 1820°C	10
T	Copper / Constantan	-250 to 400°C	43
J	Iron / Constantan	-180 to 800°C	55
N	Nicrosil / Nisil	-270 to 1300°C	39
R/S	Copper / Copper Nickel Compensating	-50 to 1750°C	10
E	Nickel Chromium / Constantan	-40 to 900°C	68

There are two types of controllers used in typical heat-treating ovens: two-position and modulation control. With two-position control systems, the actual furnace temperature oscillates

in a sinusoidal motion above and below the desired, or set point, temperature, Figure 4 demonstrates this effect.

Figure 4. Two-position temperature control.

During the initial ramping up to the set point temperature, the thermal inertia of the process will cause an overshoot and the controller needs to compensate for this by turning the heating elements off to let the furnace cool. As the furnace cools it then drops below the desired temperature requiring the heating elements to be turned back on. This process repeats in a cyclic nature, moving the temperature above and below the set point by turning on and off the heating elements to generate an average temperature equal to that of the set point temperature. This cycling on and off can cause damage to the furnace and control system if it is done too quickly. Rapid changes to electrical current have the potential to cause power surges which can destroy components if they are not properly rated to manage the needed loads. The way to prevent the damage from constant cycling is to add a hysteresis component to the controller logic. This component limits the frequency at which the furnace is switched on and off by increasing the acceptable accuracy window. For example, rather than having the furnace turn off as soon as the temperature goes above the set point or turn on when the temperature drops below the set point, a window of ±2.5℃ can be added. This means that instead of the furnace switching at exactly 100℃, it will turn off when it reaches 102.5℃ and turn back on when it reaches 97.5℃. Because temperature cannot change instantly it will take longer to reach the trigger points of the window thus slowing the rate at which the switch is flipped. In addition to saving the switch from constantly turning on and off, the hysteresis saves the heating coils from thermal fatigue.

Modulation controllers are used when process temperatures are critical and when substantial amounts of energy are being consumed. They work by more precisely controlling the power that flows into the heating elements, instead of the all-or-nothing like the two position controllers do. Modulation controllers operate using a Proportional, Integral, and Derivative or PID loop. More information about modulating controllers and PID loops can be found in the ASM Handbook, Volume 04B (6). Both controller types have different variations of how they are implemented though generally two-position controls are simpler to operate, easier to maintain, and inexpensive compared to modulation control. However, two-position control is far less efficient in terms of power usage. Modulating controllers adjust the energy input to match any change offsets while maintaining the desired temperature and also have the capability to eliminate the overshoot caused by thermal inertia seen in two-position controllers (14).

An important distinction to make is that the laboratory ovens used during this project and the ones that will be used in the FOBs are controlled with a solid-state relay (SSR) which provide a digital signal output and can be used with a modulating controller, however, they require the use of pulse width modulation which complicates the control logic.

2.4.1 User Interfaces

Heat treatment ovens come with a variety of temperature controllers, all with differing sizes and functionalities. The controllers used throughout this MQP and the ones that will be used on the furnaces in the FOBs are simple on-off or proportional control modules similar to the ones produced by companies like Honeywell and Inkbird. These simple controllers consist of four buttons and two seven-segment display modules similar to the one shown in figure 5.

Figure 5. Temperature control module interface.

In their report on usability of interfaces, Professors Sam Mahemoff and Dawson Johnston define the usability of an interface with six essential properties: task efficiency, reusability, user-computer communication, robustness, flexibility, and comprehensibility. Task efficiency is straightforward in its definition; a user interface should provide an efficient means of accomplishing the user's tasks. Reusability in user interfaces allows users to transfer the prior knowledge from one interface to another while maintaining task efficiency, for example, all furnaces have similar parameters to be programmed—temperature, time, and ramp rate—therefore, entering these values should be consistent when programming any furnace available. User-computer communication relates to the fact that there should be constant back and forth communication whenever changes to the system occur, whether that be errors, confirmations, or other factors instigated by either the human or the computer. Robustness is the program's ability to prevent and recover from errors caused internally or when the user misinforms the program tasks, such as a type. To meet the objective for any collaboration between a user and a computer, the system must be built using aspects from both parties. Flexibility ensures that these aspects are considered. A flexible program should incorporate features that lead to an effective workflow, for example, it may be useful to have the ability to load predetermined temperature profiles into the furnace. This feature will save time rather than having to reprogram the same temperature profile many times. The final property to be taken into consideration is that of comprehensibility.

Interfaces are comprehensible when users of any skill level are able to operate the controls with ease (10).

When we compare the temperature control modules made by Honeywell or Inkbird, like the one in Figure 5, to the interface usability criteria laid out previously, we find that they fall short in many regards. These controller modules are not task efficient and take a considerable amount of time to learn how they operate. They are moderately flexible and allow for up to three saved profiles to be loaded for quick use. Although they are “simple” in design, they are not comprehensible to use as there are many quirks and unique features you need to learn before you can comfortably and efficiently use the furnace, especially for someone with little to no experience. It is assumed that the soldiers who will be operating the ovens in the FOBs will have little to no experience programming them and therefore, it would be beneficial for them to have an improved user interface that encompasses the six essential properties listed prior.

2.5 Axiomatic Design Background

Manufacturing has existed in civilization since before humans kept records. As civilization has grown, so has the complexity of our manufacturing processes. Axiomatic design is a way to lay out customer demands, and customer needs so that the allocation of resources is most effective at producing the consumer's product. A well-known example of axiomatic design is Toyota’s pioneering of lean manufacturing (3)(12)(15). These methods are also applied to anything from software development to the education of engineers in a college course framework (18) (20).

When using axiomatic design, we take the project we are working on and establish the top-level functional requirement (FR0). This is the end product that we want to produce. We then break down the steps to achieve this goal into procedural order creating a list of functional

requirements that all add up to achieve the FR0. After you have mapped out all your FR's for the given manufacturing process, they are converted into Design Parameters or DPs. The DP's and FR's are arranged on a process flow chart DP's being on your X axis and the FR's on your Y axis. The functional requirements are contrasted with the design parameters to establish their relationship to one another. This process is done by looking at a FR and then comparing every DP to it. These interactions show where different production steps interact. It is better to have production steps interact with the rest of the production system after we have performed them otherwise it is a potential for waste.

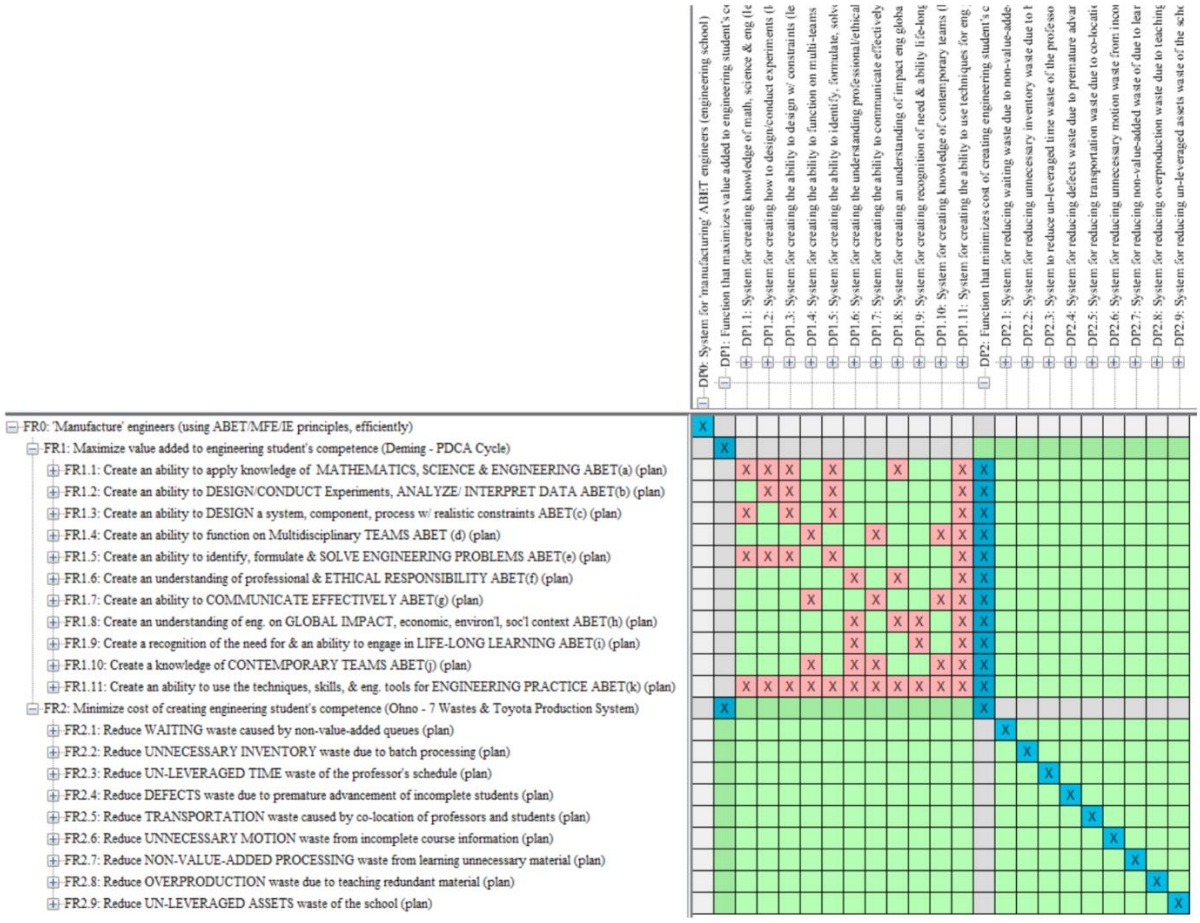

Figure 6. Process Flow chart for producing and engineer (18).

Figure 6 applies axiomatic decomposition for developing an undergraduate engineering degree course framework. We can see that the red X marks are steps that interact the green boxes are steps that do not interact. This example allows us to see the parallels between manufacturing a good and creating a workforce with a systematic framework. We can see we have a top-level requirement and we see the steps that interact. differences between the two exist in how closely related each step is to the others. In the manufacturing line of a good, there is a more sequential order to each step. But when educating an individual, it is not necessarily a linear process. When looking at the simpler DP's and FR's in the figure, it is easier to visualize how the buildup of skills is needed to advance a student through this production system (20).

A similarly obscure application of axiomatic design is in software development. This is not the most intuitive application because you are not creating a physical thing that you can touch and feel but just like the example of teaching an engineer you are still developing a product. Its applications are discussed in '*Decision Making and Software Tools for Product Development Based on Axiomatic Design Theory.'* (18) The authors highlight how mapping the functional requirements and design parameters of a given software can make conceptualizing the desired use of the software easier to achieve while editing.

More specifically when software development teams are, "considering the change of a design, consequences must be identified so a rational economic decision about proceeding with the change can be made" (18). What this author is saying is that axiomatic design helps his team most effectively change their design plans. Thus, saving time and production costs.

3 Experimental Procedure

3.1 Heat Treating and Analyzing Carbon Steels

As stated previously, one of the main objectives for this project was to compare the effects of heat treatment processes on the microstructure and mechanical properties of carbon steel alloys. To achieve this goal, two carbon steels were tested; the first was an 8630-alloy steel and the second a 1018 carbon steel. We chose to cut eighteen samples of the 8630 steel to measure the effects of differing tempering temperatures, and four samples of the 1018 steel to measure the effects of normalizing and annealing the metal. For 8630 steels, the respective number of samples were cut, then hardened and tempered in pairs. For 1018 steel, the team only operated normalizing and annealing. The variation in processes for the two types of steel was done because 8630 steel is meant for harder applications, which is achieved from the quenching and tempering processes, and the 1018 steel is used more commonly in softer applications, which is achieved by normalizing and annealing. Next, all samples were polished, hardness tested, and etched to expose the grain structure. The samples were then placed under a high-power confocal microscope to observe the grain growth and microstructures caused from the heat treatment process. This chapter describes the steps and tools used to go from the raw bar stock to the final results of the project. See below for Figures 7 and 8, flowcharts that concisely display our procedures for both types of steel.

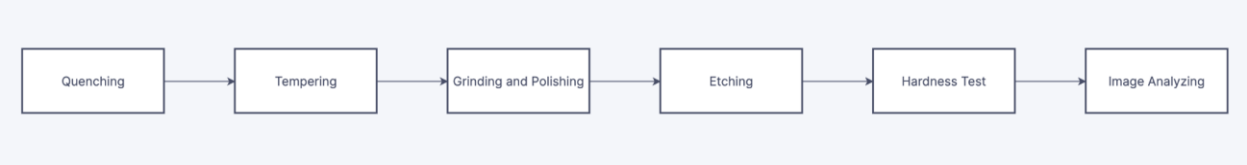

Figure 7. Flow chart of heat treatment on 8630 steel samples.

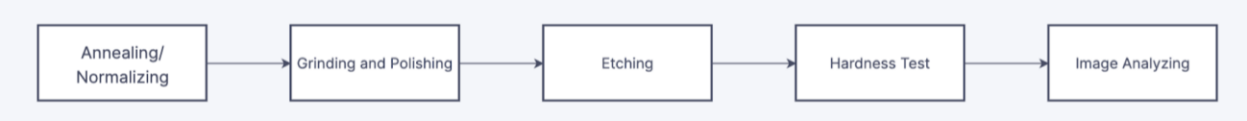

Figure 8. Flow chart of heat treatment on 1018 steel samples.

3.1.1 Sample Preparation

The first step of this project after receiving the steel stock was to check the elemental composition of the material; this was done using a Hitachi PMI-Master portable Optical emission spectroscopy analyzer, which returns a percentage table of each element present. After identifying the composition of the steels, the next step was to cut the stock into small samples approximately half an inch wide. The samples were cut using a EXTEC chop saw with a hard ferrous abrasive disk rated to cut both the 8630 and 1018 steel. Once the samples had been cut, they were ready to be placed in the furnace for heat treating.

Figure 9. Flow chart of heat treatment on 1018 steel samples.

3.1.2 Heat Treating

The initial step of heat-treating the samples was to identify the critical temperature based on the material's composition. The critical temperatures for most alloy steels can be found from places like the American Iron and Steel Institute (AISI), American Society for Metals (ASM), and MatWeb. MatWeb contains one of the largest databases for materials ranging from thermoplastics to superalloys (13). For both 8630-alloy steel and 1018 carbon steel, the critical temperature is 850℃. Once the critical temperature had been identified, the samples were placed in a Baker Model #6 Lab Oven and held at the selected temperature for at least two hours. This allowed for the entire sample to be heated evenly before being removed from the furnace and quenched in water. While quenching the 8630 steel, six samples at a time were brought up to temperature and quenched. Only a maximum of six samples were treated at once to minimize the effect of air-cooling the material while the door is open, which could affect the grain structure.

Figure 10. Baker Furnace with loaded samples.

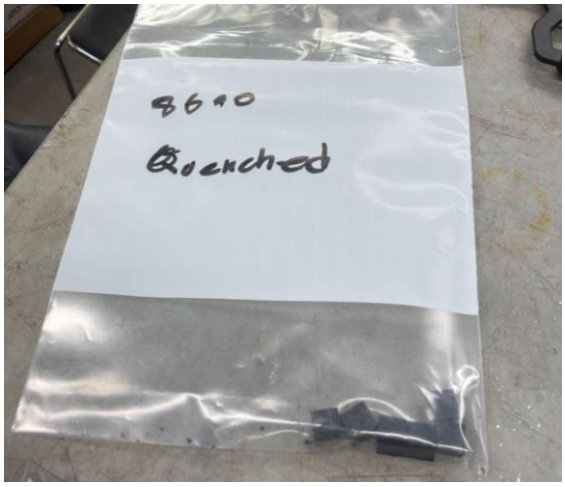

Figure 11. Quenched 8630 samples.

With the quenching complete, the next step of the project was tempering the samples back to reduce their hardness, increase their ductility, and modify other material properties. The samples were tempered, in pairs, starting at 400℃ up to 700℃, in 50℃ increments. Tempering the samples this way gives a wide representation of the common temperatures used to heat treat steel. Figure 12 shows the various temperature profiles used while heat treating the 8630 steel samples. For tempering, the Baker Model #6 Lab Oven was used again, and each pair of samples was held at the desired temperature for one hour. A one hour holding time was chosen as our samples are approximately half an inch thick and tempering parts requires at least two hours of tempering per inch of cross section (2). To keep the tempering cycles consistent, the pairs of samples were placed in the oven at room temperature and brought up to the required temperatures at a rate of 1000℃ per hour.

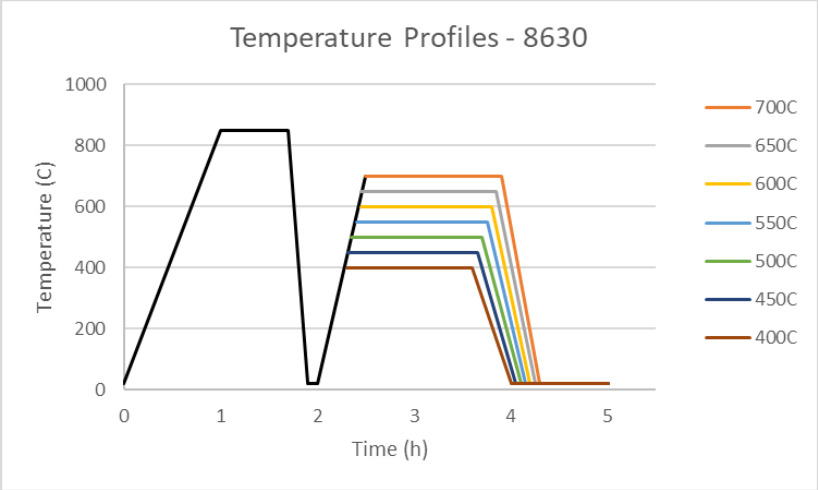

Figure 12. Temperature profiles for 8630 steels.

The 1018 carbon steel samples were heat treated following the temperature profiles shown in Figure 13. All four samples were brought up to the critical temperature of 850℃ and held there for two hours, then two of the samples were pulled out of the oven to normalize in ambient air while the other two samples remained in the furnace and slowly cooled back down to room temperature as the furnace cooled. Once all twenty-two samples had cooled down to room temperature the heat treatment portion of this project was complete, and the next step was to analyze the microstructure and mechanical properties of each sample.

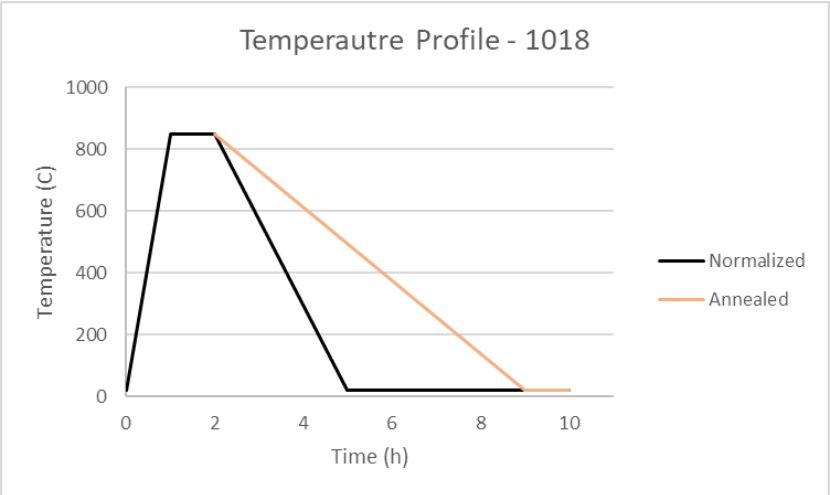

Figure 13. Temperature profiles for 1018 steel.

3.1.3 Grinding and Polishing

The next step was to grind and polish the samples to prepare for hardness testing and image analyzing. First, the samples needed to be mounted in a polyurethane disk using the lab's Buehler SimpliMet™ 4000 Mounting System. The resulting mounted set up consisted of the sample embedded in the disk with one face exposed. This face would then be ground and polished on the Buehler AutoMet 250 automatic polishing machine. The machine's instruction manual described which abrasive cutter and wheel texture to use for each steel alloy, depending on the alloy's hardness and material properties. After the polishing, each sample had a mirror-like surface as shown in Figure 14. Then each sample is ready to be hardness tested, etched, and analyzed under a microscope.

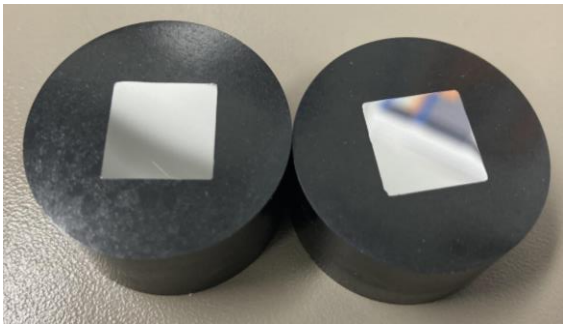

Figure 14. Picture of polished samples.

3.1.4 Etching, Hardness Testing, and Image Analyzing

The final steps of the project were to test the hardness of the samples using the Wilson® VH3300 Vickers Hardness tester, shown in figure 15, followed by etching the surface using a Nital solution, and finally observing the grain size and microstructure using the Sensofar® Metrology Confocal Microscope shown in figure 16. One tempered sample from each pair was hardness tested a total of ten times and had two images taken near each divot to get a clear sign of how

the heat treatment process influenced the microstructure and mechanical properties of the steel alloy.

Figure 15. Wilson® VH3300 hardness tester

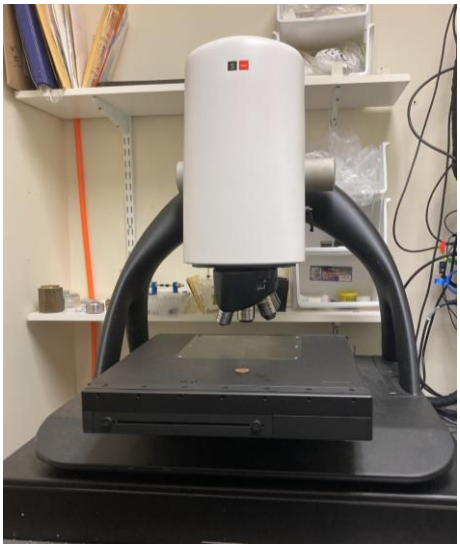

Figure 16. Sensofar® Metrology Confocal Microscope

3.2 Design of a Custom User Interface

During the initial stages of this project, when the team first began heat treating the steel samples, the team members noticed how confusing and inefficient it was to use the temperature controller built into the furnace. After making this observation they proposed the implementation of a custom user interface that would be easier to use and provide more feedback to the furnace operator. Figure 17 shows the block diagram for the proposed control system. This system would allow for the furnace to be controlled with an external CPU such as a laptop or even a phone.

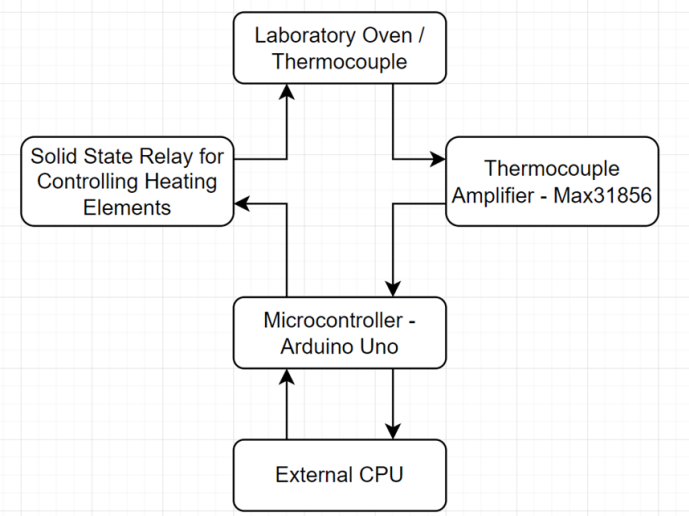

Figure 17. Temperature Control Block Diagram

After the proposed system diagram was reviewed, research was done to implement a new control system into a laboratory oven available for the team to modify. The oven used was a Thermolyne Benchtop Muffle Furnace, Model #F48015 shown in Figure 18 with the accompanying wiring schematic shown in Figure 19.

Figure 18. #F48015 Thermolyne Benchtop Furnace

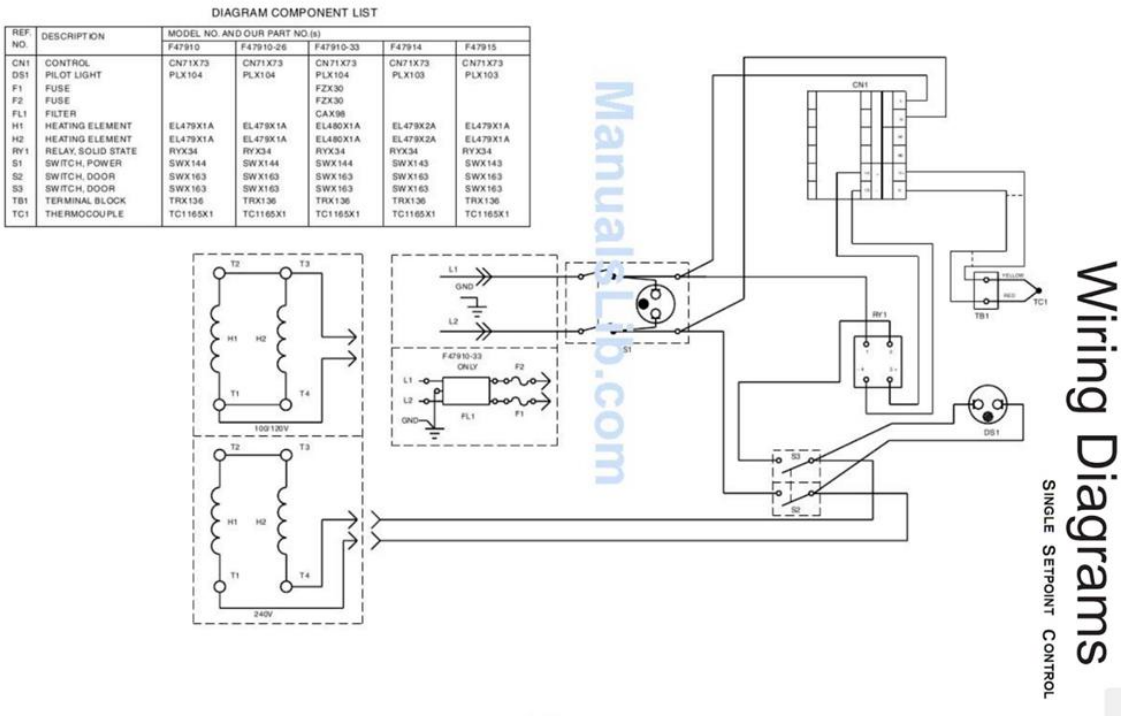

Figure 19. Wiring Diagram for the #F48015 Thermolyne Furnace.

Based on the wiring diagram of Figure 19, it was decided that an improved control system could be implemented by bypassing the CN1: CN71X73 Control Module and inserting a custom circuit in its place. This custom circuit would need to be able to read the thermocouple output, send that data to a microcontroller, and from there both control the solid-state relay RY1 in Figure 19 using a feedback loop as well as send the data to the graphical interface on a laptop for real time analysis. Building this custom circuit required a microcontroller as well as a way to read the thermocouple output, for this, an Arduino Uno, and Adafruit Universal Thermocouple Amplifier model Max31856 were used, respectively. The Max31856 amplifier is used to amplify the voltage difference caused by the Seebeck effect generated by the K-type thermocouple located in the furnace. Figure 20 shows the wiring diagram of the custom circuit and Figure 21 shows the physical circuit fully assembled. The circuit operates by having the Arduino read the temperature inside the furnace through the thermocouple amplifier, comparing this value against the set point

temperature defined by the user through the interface, and either turning on or off the switch in the solid-state relay that gives power to the heating elements of the furnace. If the temperature is below the set point, the heating elements turn on, and if the temperature is above the set point, the heating elements are turned off. This is considered a two-position control system as described in section 2.4. In addition to controlling the heating elements, the Arduino also sends the data to a user interface on a laptop to provide real time analysis for the furnace operator.

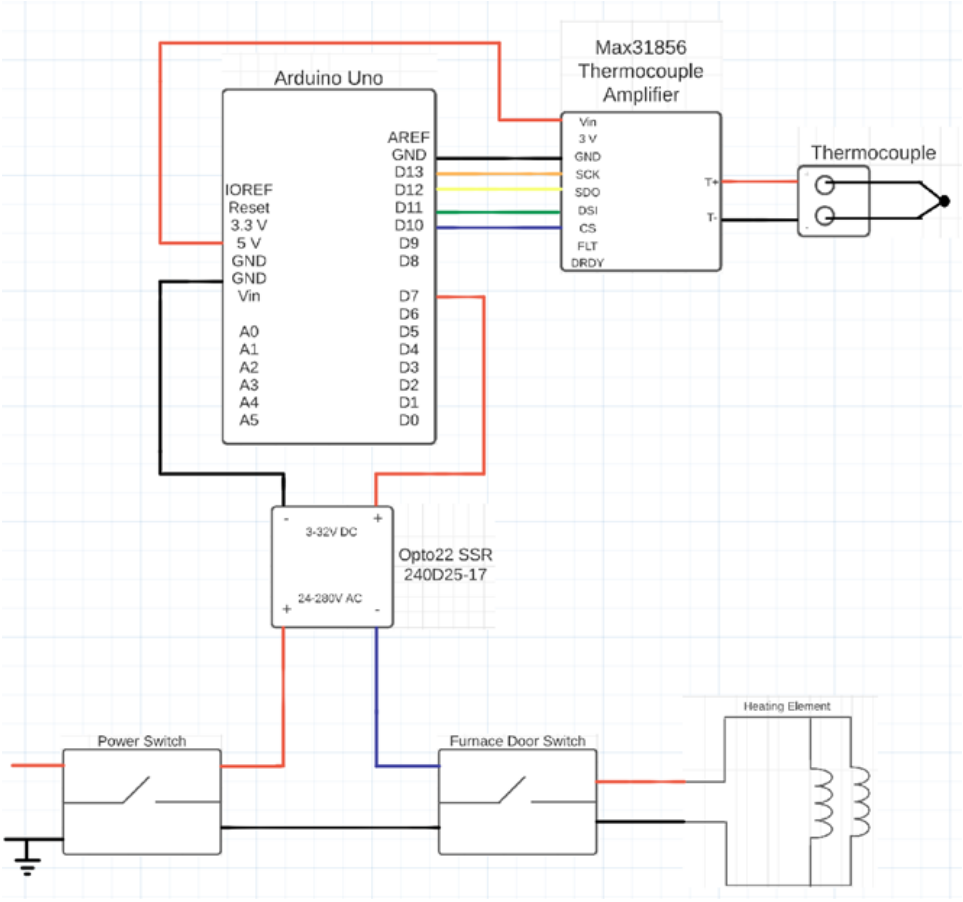

Figure 20. Wiring Diagram for Custom Control Circuit.

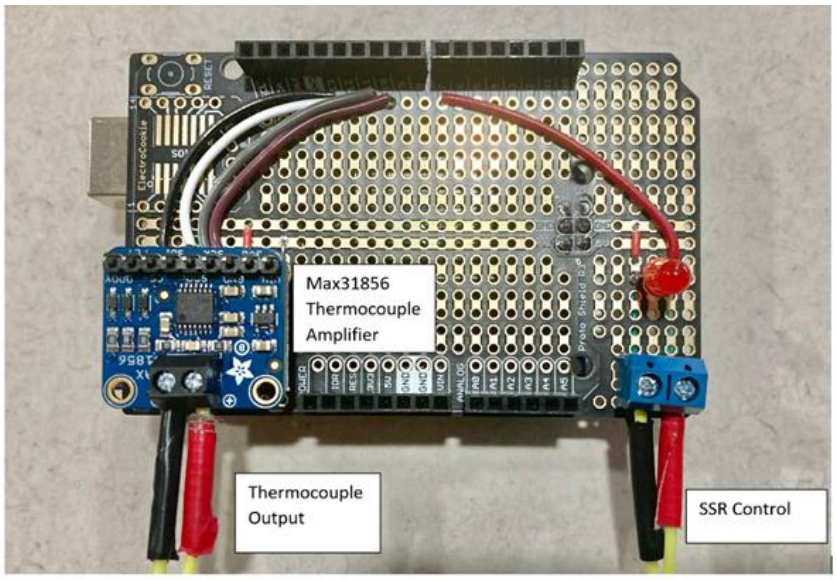

Figure 21. Fully Assembled Custom Control Circuit.

The graphical user interface was designed using the MegunoLink software and accompanying Arduino IDE libraries. MegunoLink is a configurable tool for designing interfaces to control Arduino sketches. The MegunoLink software communicates to the Arduino by sending and receiving serial commands to either update values in the Arduino program or values of the graphical interface. These values can be visualized in many forms including real time plotting, serial monitoring, tabulating, and more (22). Below are functional requirements the team wanted the improved control interface to meet:

- Program up to eight stages of a temperature profile
    - Each stage requiring the set point temperature, holding time, and ramp rate to be defined
- Display, in text format, the actual furnace temperature and the set point temperature
- Calculate and monitor the total program time
- Have the programmed temperature profile overlaid with a real time plot of the furnace temperature

- Make a comprehensible interface incorporating the usability elements defined by Mahemoff and Johnston as described in section 2.4.1

Meeting these requirements would provide a usable interface for both students and soldiers alike allowing for quick, efficient, and precise control of a furnace while also enabling the possibility of future data analysis and improvements to the heat-treating process.

3.3 Process Flow

During the axiomatic design breakdown, we found our top-level functional requirements. We then created a process flow chart to break down our FR0. We then took our FR chart and used it to develop a design parameter (DP) chart. These two charts intersect to produce a process flow chart.

3.3.1 Heat Treating steel

DP0	System for providing test samples with known data points.
DP1	System for preparing the test sample.
DP1.1	System for cutting the test sample.
DP1.2	System for heat treating the test sample
DP1.3	System for polishing the test sample
DP2	System for harness testing the test sample
DP2.1	System for eching the test sample
DP3	System for imaging the test sample

Figure 22. Designs Parameters for heat treatment.

FR0	provide test samples with known data points.
FR1	Prepare the test sample.
FR1.1	Cut the test sample.
FR1.2	Heat treat the test sample
FR1.3	Polish the test sample
FR2	Harness test the test sample
FR2.1	Etch the test sample
FR3	Image the test sample

Figure 23. Functional Requirements for heat treatment.

	DP0	DP1	DP1.1	DP1.2	DP1.3	DP2	DP2.1	DP3
FR0	X			X		X		X
FR1	X	X						
FR1.1	X	X	X					
FR1.2	X	X	X	X				
FR1.3	X	X	X		X			
FR2	X	X	X	X	X	X		
FR2.1	X	X	X	X	X		X	
FR3	X	X	X	X	X		X	X

Figure 24. Process flow chart for heat treatment.

The three figures listed above are an axiomatic design decomposition and a process flow analysis that helped us order the steps of this project to better. We can see that the majority of the interactions are underneath the diagonal line created by the FR's and DP's of the same number interacting with each other. This is good because it shows that our project was completed

in a manner to reduce as much waist as possible.

3.3.2 Furnace Control System

DP0	System to improve user interface with visually demonstration of: programmed temp, actual temp, time elapsed, time to go.
DP1	System to research temperature control systems .
DP1.1	System to buy a voltage amplifier to read furnace data.
DP2	System to research hardware to communicate between thermocouple and software.
DP2.1	System to order components for custom control circuit.
DP2.2	System to assemble hardware.
DP3	System to research software to read and plot data.
DP3.1	System to buy software to read and plot plot data.
DP3.2	System to program software to read and plot plot data.

Figure 25. Design Parameters for furnace control system.

FR0	Improve user interface with visually demonstration of: programmed temp, actual temp, time elapsed, time to go.
FR1	Research temperature control systems .
FR1.1	Buy a voltage amplifier to read furnace data.
FR2	Research hardware to communicate between thermocouple and software.
FR2.1	Order components for custom control circuit.
FR2.2	Assembled hardware.
FR3	Research software to read and plot data.
FR3.1	Buy software to read and plot plot data.
FR3.2	Program software to read and plot plot data.

Figure 26. Functional requirements for furnace control system.

	DP0	DP1	DP1.1	DP2	DP2.1	DP2.2	DP3	DP3.1	DP3.2
FR0	X		X			X			X
FR1	X	X							
FR1.1	X	X	X						
FR2	X	X	X	X					
FR2.1	X			X	X				
FR2.2	X			X	X	X			
FR3	X	X					X		
FR3.1	X						X	X	
FR3.2	X		X			X	X	X	X

Figure 27. Process flow chart for furnace control system.

In these past three figures we mapped and then plotted the steps to creating the furnace control system. We can see the interactions plotted and we can see how the graph supports the procedural order that we used. The graph supports this process because as stated earlier having the interactions below the line allows steps to build properly on one another.

4 Results

4.1 Hardness results for 8630 and 1018 steel

The following table is the hardness tests results on our 8630 heat treated samples:

Table 2. 8630 steel hardness test results.

Tempering Temperature	**Hardness Test Results [HV]**
As Purchased (Neither quenched nor tempered)	173
Control (Quenched but not tempered)	511
400℃	388
450℃	327
500℃	305
550℃	290
600℃	259
650℃	230
700℃	201

These results clearly show that the quenched control is the hardest, with the tempered samples coming after, progressively getting softer from 400℃ to 700℃, and finally arriving at the softest sample; the as purchased control. This is consistent with the heat treatment processes discussed in Section 2.2; the faster a steel sample is cooled from its high furnace temperature, the harder the steel sample will be, making the quenched sample the hardest. Additionally, it

makes sense that a lower tempering temperature results in a harder sample than a higher tempering temperature, because at lower temperatures the sample is given less energy to change its microstructure, and therefore more closely resembles the quenched steel.

Below is the data from the hardness tests conducted on our annealed and normalized samples:

Table 3. Annealed and normalized 1018 steel hardness test results.

Process followed	Hardness Test Results [HV]
Annealing	123
Normalizing	141

These results are consistent with what we know about annealing and normalizing (see Section 2.2). Annealing will produce the softest possible steel for a given composition, with normalizing also producing soft steel, but to a slightly lesser extent. Another finding from this data is the role the steel's chemical composition plays in hardness. The 8630 stock was purchased as cast steel from the manufacturer, where molten metal is poured into a mold, and left to cool. So, the "as purchased" 8630 steel is normalized. After both undergoing normalization, the 1018 steel is a full thirty hardness values softer than the 8630 steel, showing the importance of identifying the composition of the steel one wishes to treat.

4.2 CNN Model

The captured images were then uploaded to the CNN model being developed by Yutao Wang. The model analyzed the images based on the microstructure of the image and inputted values for chemical composition and heat treatment cycle and gave its prediction on what the hardness of the sample would be. Out of 1260 available data points, 252 were randomly selected

to test the accuracy of the model. Comparing the model-predicted hardness value to the actual hardness value, the model was 95.37955% accurate, the full results can be found in Appendix A.

4.3 Using the Improved Interface

Implementing the design requirements for the user interface described in section 3.2 resulted in the user interface panel that is shown in figure 28. There is a drop-down box for selecting between one and eight stages to be run. Each stage consists of a ramping time to get the furnace up to temperature plus the holding time where the furnace and sample sit at the set point temperature. Once the number of stages has been selected, entry boxes appear to set each of the stage's parameters: the set point temperature in degrees Celsius, the holding time in minutes for how long the sample should be held at the set point temperature, and finally the ramp rate in degrees Celsius per hour for how long it will take to ramp up or down to the next stage's temperature. Once all the stage parameters are set the operator can click the Set Stages button to display the temperature profile in the plot on the bottom half of the interface and read the Total Program Time to check that the parameters, they entered are correct. If they are incorrect, the user can either change the parameters individually or press the Clear Stages button to reset all parameters back to a zero value. After confirming the temperature profile and stage parameters are set to the desired configuration, the furnace operator can then press the green Start button in the top middle panel which will begin the heat treatment process. During this process, the Arduino reads the actual furnace temperature every second and runs the logic to either turn the heating elements on or off. In addition, the Arduino will also update the user interface by displaying the Programmed Temperature, the Furnace Temperature, the Time Elapsed from when the program started, and the Time Remaining until the program finishes. Along with displaying the furnace temperature in a textual format, the custom interface has a real-time data plot which displays the

programmed temperature profile with a black line and the real furnace temperature with a red line on top of each other. This allows the furnace operator to check whether or not the furnace has maintained the proper temperature through the heat treatment process.

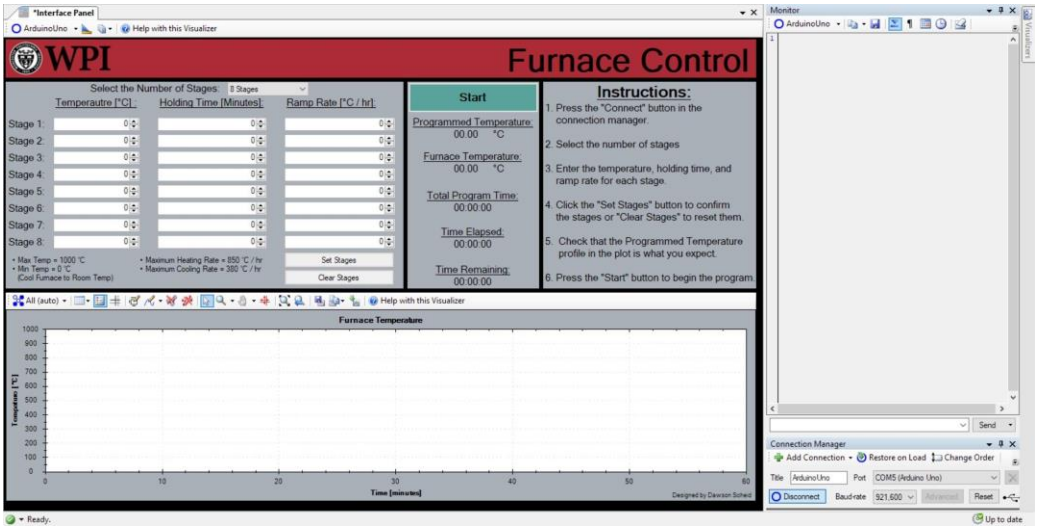

Figure 28. Custom graphical user interface for furnace control.

After the program completes the Arduino will turn off the switch giving power to the heating elements and the furnace will cool down at its natural cooling rate. Figures 29 and 30 show two different temperature profiles being run with the interface. The first program ran was a four-stage profile starting at ~100°C ramping up to 150°C, holding for 5 minutes, then increasing to 200°C for 10 minutes, then 400°C for 5 minutes, and finally ramping down to 100°C for another 10 minutes. From the temperature plot generated as the program ran it can be observed that around the 15-minute mark there was a sudden increase in temperature from what was programmed. This was due to the furnace door being secured shut as there is naturally a large gap in the insulation through which heat can escape. Another feature of note, by looking at the temperature plot, is that the programmed ramp down rate was higher than that of the natural cooldown rate of the furnace and thus the sample being heat treated did not have the expected properties. The second program run was a five-stage profile starting at room temperature, increasing to 100°C

for 10 minutes, then to 200°C for 15 minutes, then 300°C for 10 minutes, then 500°C for 15 minutes, and finally 600°C for 5 minutes. There was no cooldown stage for this profile signifying that the sample was removed from the furnace to normalize.

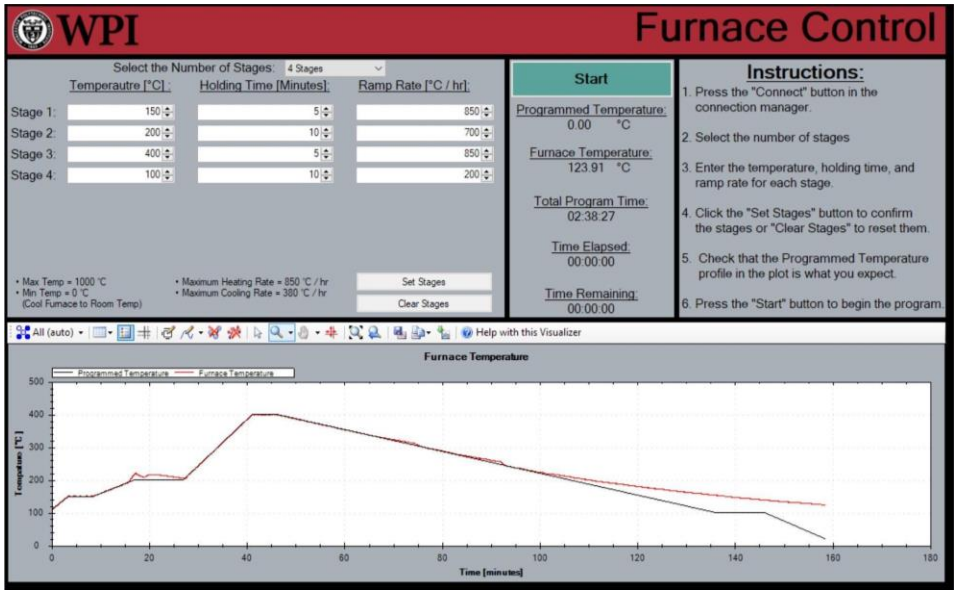

Figure 29. Four stage temperature profile run with the custom interface.

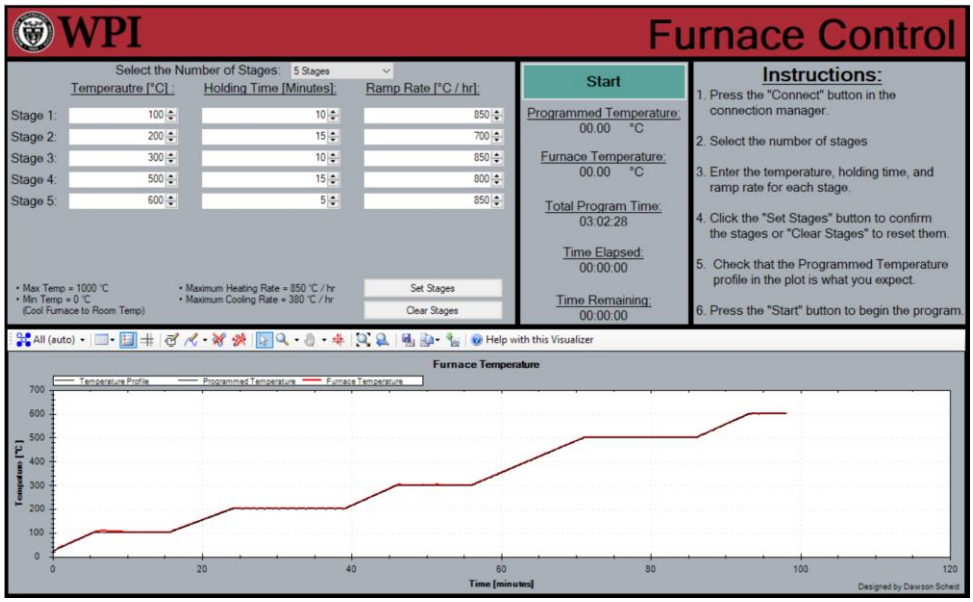

Figure 30. Five stage temperature profile run with the custom interface.

After the second program had been run, both the programmed and furnace temperature data were downloaded and compared to evaluate the accuracy of the control system. Averaging the difference between what the temperature should have been to what it actually was resulted in a -0.05°C offset over the entire course of the heating cycle. This value is well within the ±2.5°C for temperatures under 675°C to maintain an accurate and uniform temperature distribution within the furnace and sample being heated (5).

Apart from having improved accuracy over the original control interface, the new interface incorporates the six essential properties defined by Mahemoff and Johnston in section 2.4.1. The interface has improved task efficiency as it requires less time to program the desired heat treatment profile, it is reusable as it incorporates the familiar configurable parameters such as the set point temperature, holding time, and ramp rate. Next, the user interface has improved user-computer communication with real time plotting and error messages that tell the furnace operator exactly what is happening at a given time. The design is robust and will limit the user inputs as to not damage itself or any of the equipment used while the program is running. Finally, it is flexible and comprehensive in regards that future additions can be easily implemented and operators of any level, from novice to experienced, can use the interface without the need of training.

4.4 Economic Analysis

When looking at the economic effect that our project had on the greater DOD initiative, we can identify multiple different costs. Our first step was to understand what percentage our project affected the greater DOD initiative. To do this we looked at what WPI added which was a computer algorithm to predict grain structure and hardness from known heat treatment data. This provided an understanding of the material properties without expensive testing equipment.

Our project provided half the test data that was needed to properly calibrate WPI's

algorithm. The initial investment into the project consisted of metal to perform tests of and the cost of using the machines to run these tests equated to a total expense of six hundred dollars.

Table 4. Initial investment to the project.

Machine time	$500
Test material	$100
Total	$600

To present a value-added chart we first subtracted our six-hundred-dollars to find our initial starting point. Then we estimated how much we predict this will save the DOD as a percentage of what they invested to fund this research. For this, we are going to say that our MQP team contributed 0.01% to solving WPI's section of the project. We know that WPI was given 1.15 million to complete its research. that leaves us with an evaluation of $11,500 annually for having our team conduct this research.

Figure 31. Value added over 1 year.

Displayed above we can see a visual representation of the value that our project adds from the initial point of investment. The visual representation that this provides is extremely powerful as presenting graphs instead of a variety of charts with numbers provides clarity as to when our project will be seeing significant growth. This allows our project to be shared more broadly as it is simple to understand what these numbers mean. It is obvious that this project has a low initial cost and a significant amount of lead time but once we reach implementation our project rapidly becomes profitable. It also displays when our project is predicted to become profitable and at what rate it will be profitable.

These analytics are all very theoretical as we are taking the value of our project based on the DoD investment, we are also approximating the amount of involvement that our project has with the DoD. This creates a certain level of ambiguity that cannot be avoided. The important thing that this evaluation shows is that our project is continuously providing value. We also see that even though our numbers are rough estimates we still have significant margins and if we were valued at a significantly less percent of involvement our project would still be profitable. The one major drawback for the work we did is the time of implementation. During our project we did not take into consideration that the DoD granted the funds and then had to wait for however long our project took to implement. This is a significant opportunity cost that could be avoided by collaborating directly with a corporation instead of an institution of higher learning. However, if we were employees the cost to implement would have been in the thousands not the hundreds. This demonstrates another extremely effective part of the value-added chart. We can demonstrate how increasing the speed at which our project gets completed would decrease the time to implementation, but it would increase the startup cost significantly.

Table 5. Potential cost of contracting.

Machine time	$500
Test material	$100
Engineer working cost	$5000
Total	$600

As an example, we are going to say that an engineer must spend half a month collecting data samples for our algorithm. We approximate this is going to cost approximately five thousand dollars in engineering working time. This increases our initial cost to five thousand six hundred but decreases our wait time by six months resulting in a very similar value-added chart just without that six-month lead period.

Figure 32. Potential value added of contracting.

This hypothetical situation demonstrates the importance of creating value added charts. It outlines how our ability to demonstrate how the effects of what we do will greatly impact your ability to share our work with other MQP teams. We can also see that rapid prototyping allows us to view a problem from many angles. We could look at examples with an added variable of the DOD spending two million dollars a month until our product is produced. Does it make sense to wait for a college team to solve this problem instead of paying for a contractor? It might be more expensive upfront but helps the DOD solve the problem sooner so they can reduce spending. Or we could see how this project would affect a civilian contractor who might not have the same ability to wait as long or thinks they can make more money off it so paying a significantly higher upfront cost increases revenue in that 1-year window.

In conclusion, value add tools are an extremely versatile and powerful tool that helped our team demonstrate the profitability of our project. With an analysis of our value-added chart, we strongly support the execution of this project.

5 SUMMARY AND CONCLUSIONS

5.1 Summary

According to the final results, the team found out that 8630 has its highest hardness results while it was just quenched as 201HV and quenched at 700℃. The range of the hardness results is in between 511 HV to 201HV, with 173 HV as the hardness of just bought 8630 material. These findings are expected because this has been supported by the background research that after the metals have been heat-treated, they will become softer. Here is a temperature profile of 8630 after tempering conducted by all the results that was collected from the imaging section.

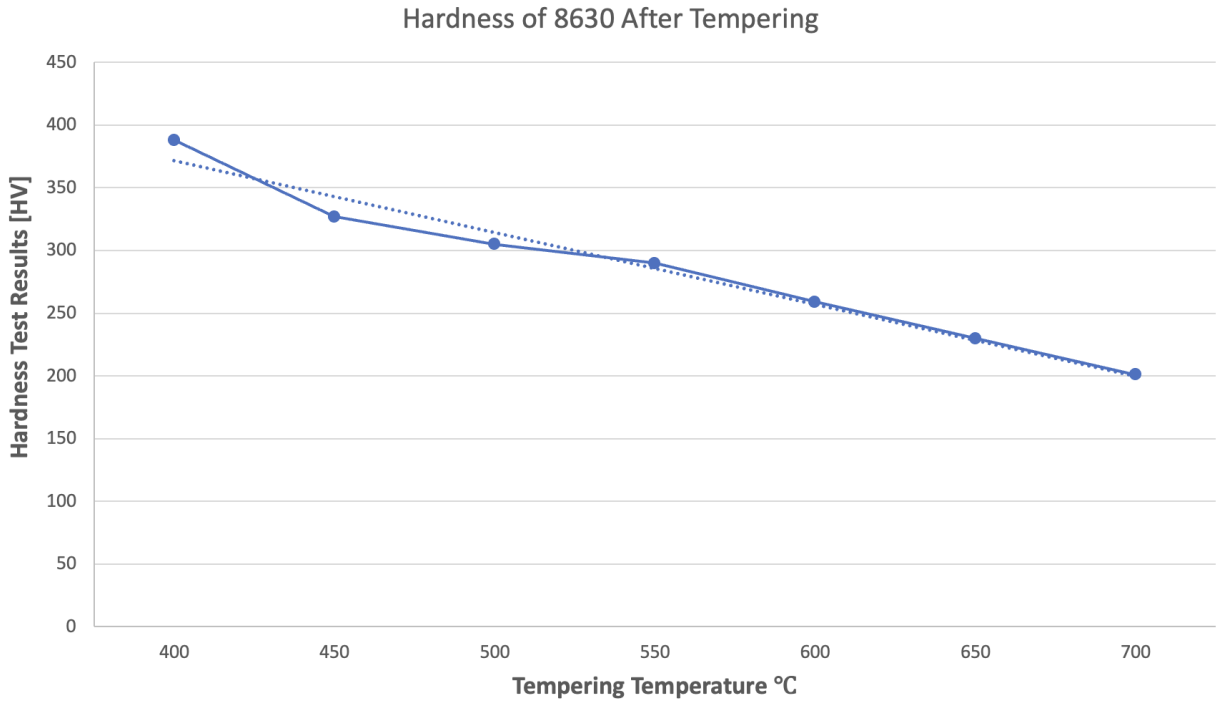

Figure 33. Temperature profile of 8630 after tempering with linear regression line.

Adding on to the previous finding, during the quenched process, the higher the temperature it has been quenched, the lower the hardness results the materials will get. The grain size for 8630 after quenched is normally smaller visually under confocal. The smaller the grain

size of steel, the harder that steel is, and that is reflected in these samples' microstructures. To highlight this finding, below are two images captured using the lab microscope, one of the 400℃ tempered sample, and the other of the 700℃ tempered sample. One can clearly see the difference in grain size between the 400℃ and 700℃ samples.

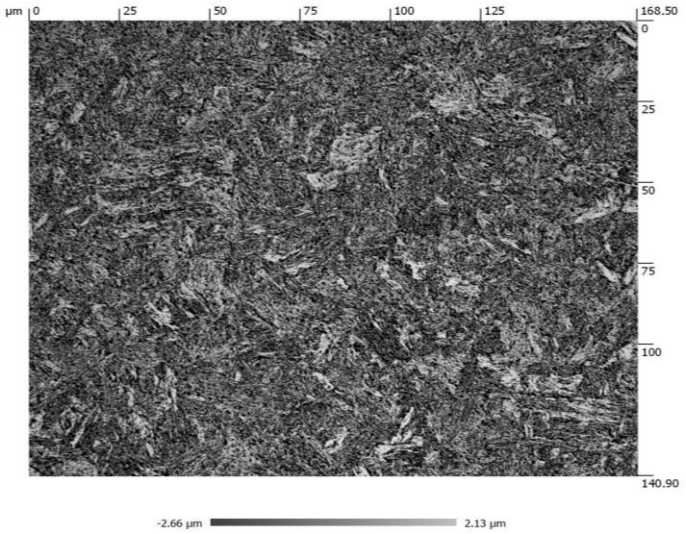

Figure 34. 400℃ tempered 8630 steel.

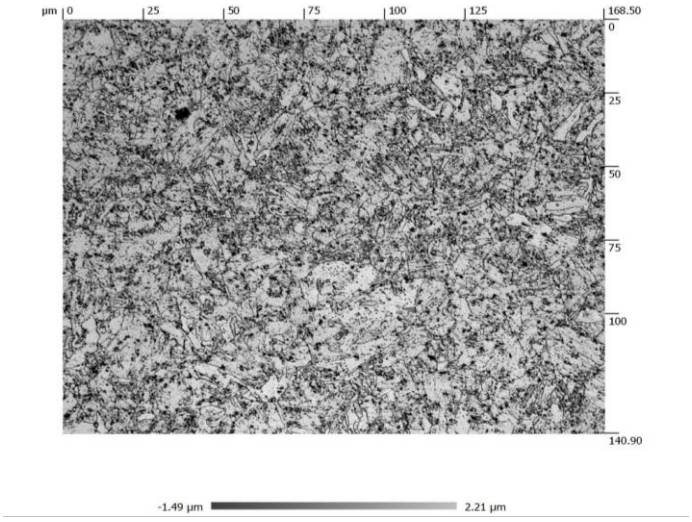

Figure 35. 700℃ tempered 8630 steel.

The grain size for the images of 8630 that was normalized and annealed are dissimilar. The grain sizes for 8630 are normally smaller compared to annealed 1018 steel. Based on the results, the annealed sample is considerably softer than the normalized sample. These differences between the two processes are also shown in our captured images, seen below. The earlier correlation between larger grain size and softer steel applies here, with the larger grained annealed sample being softer than the smaller grained normalized sample.

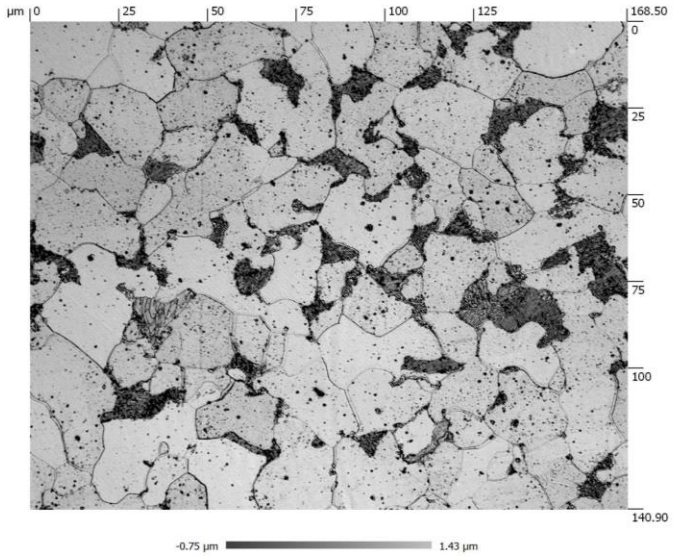

Figure 36. Annealed 1018 steel image.

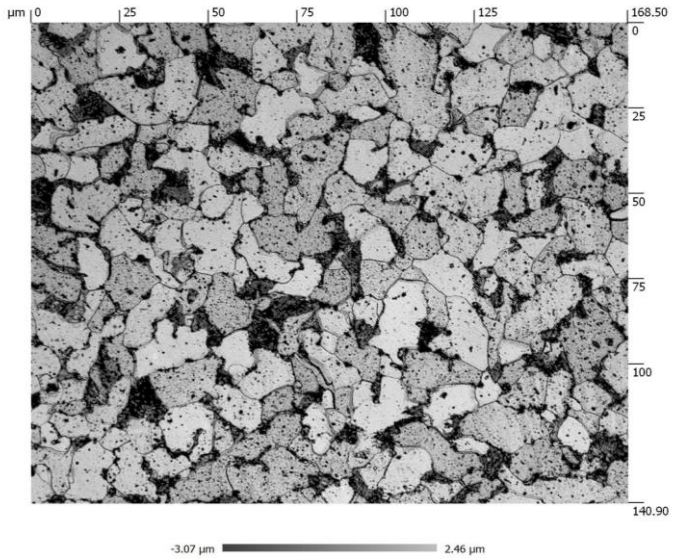

Figure 37. Normalized 1018 steel image.

5.2 Conclusion and Future Recommendations

We accomplished the three main objectives we set for ourselves with this project. Firstly, providing data samples for the CNN model. While there were many roadblocks we ran into along the way, we were able to accomplish our objective of selecting multiple varieties of commonly used steels (8630 and 1018), heat treating them in controlled environments, testing their properties, imaging them, and running that data through the CNN model. These efforts produced 160 unique images, and one hundred hardness data points, improving the accuracy and utility of the CNN model. The second objective was a success as well. Not only was the new furnace control interface more accurate than the original, and also effectively incorporated the six essential interface properties of task efficiency, reusability, user-computer communication, robustness, flexibility, and comprehensibility, resulting in an all-around improved interface. The third objective was also completed, with an axiomatic design decomposition executed in the form of a process flow and economic analysis.

While we were successful in achieving our objectives, the work does not end there; these objectives are iterative and ongoing. While we added a significant body of data to the CNN model, more can and must be added. We improved the user interface, but it can be improved further. And any changes to those objectives will prompt a change in their respective process flows and economic analyses. For those who pick up these objectives where we leave them off, we recommend the following:

1. Continue to gather heat treatment data for new steel alloys commonly used by the DoD and get more data points on alloys that have already been tested, to increase the CNN model's accuracy.
2. Conduct nontypical heat treatments on alloys, instead of just typical ones (ex. Put 8630 steel through annealing). Even though 8630 steel is best used for applications requiring high hardness, a situation on a FOB may call for a softer

part, and the only steel in copious quantities on base is 8630. This will increase the usefulness of the model, allowing soldiers to be more resourceful.

3. Ensure full access (or as close to full access as possible) to facilities necessary for conducting your heat treatment operations. Our team was limited to only times when our TA could let us into the furnace room, polishing room, etc. This hindered our ability to plan and conduct lab work, as we were limited to our TA's availability from the beginning.
4. Make the user-interface wireless, so one does not need to hook up their laptop to the furnace in order to run the program, and instead can control the furnace from elsewhere with their phone, laptop, or other wireless device.
5. To improve the custom control system to be even more accurate, a PID control loop can be added which can completely eliminate the overshoot caused by thermal inertia.
6. When designing a custom user interface, use a software program that is well documented and easy to debug and get help on. During this project we started with using Visual Studio but ran into many issues with no valid solutions even after researching.

REFERENCES

1. Awati, R. (2021, October 18). What is the Seebeck effect? SearchNetworking. Retrieved April 4, 2022, from https://www.techtarget.com/searchnetworking/definition/Seebeck-effect
2. Bryson, B. (1997). Heat treatment, selection, and application of Tool Steels. Hanser Gardner Publications.
3. Becker, Ronald M. "Lean manufacturing and the Toyota production system." *Encyclopedia of world biography* (1998).
4. Deep learning tools to predict mechanical properties of steel alloys, Presented by Yutao Wang, DoD Steel Summit, November 2021.
5. Dossett, J. L., & Boyer, H. E. (2006). Practical heat treating. Asm International.
6. Dossett, Jon L. Totten, George E.. (2014). ASM Handbook, Volume 04B - Steel Heat Treating Technologies. ASM International. Retrieved from https://app.knovel.com/hotlink/toc/id:kpASMHVBS2/asm-handbook-volume-4b/asm-handbook-volume-4b
7. File:CCT curve steel.svg. Wikimedia Commons. (n.d.). Retrieved April 11, 2022, from https://commons.wikimedia.org/wiki/File:CCT_curve_steel.svg
8. Final Revised serdp technical section. (n.d.). Retrieved October 11, 2022, from https://drive.google.com/drive/folders/1dfN2bPlORKBHjKJGkvnmk0VGZBdxQ9zg
9. Gums, J. (2018, January 26). Types of temperature sensors. Digi. Retrieved April 4, 2022, from https://www.digikey.com/en/blog/types-of-temperature-sensors#:~:text=There%20are%20four%20types%20of,based%20integrated%20circuits%20(IC)

10. Lin, W. (2021, April 15). Megunolink Pro: The Swiss Army Knife for Arduino. MegunoLink. Retrieved April 4, 2022, from https://www.megunolink.com/
11. MacKenzie, D. S. (2018, November 29). D. Scott Mackenzie, Ph.d., FASM. Gear Solutions Magazine Your Resource to the Gear Industry. Retrieved April 4, 2022, from https://gearsolutions.com/departments/hot-seat/continuous-cooling-transformation-diagrams/
12. Ohno, Taiichi. Toyota Production System Beyond Large-Scale Production.Boca Raton, FL: CRC Press, 1988.
13. Online materials information resource. MatWeb.(n.d.). Retrieved April 11, 2022, from http://www.matweb.com/index.aspx
14. Principles for a usability-oriented pattern language. IEEE Xplore. (n.d.). Retrieved April 4, 2022, from https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=732206&tag=1
15. Shirouzu, Norihiko. “How Toyota Thrives When the Chips Are Down.” Reuters. Thomson Reuters, March 9, 2021. https://www.reuters.com/article/us-japan-fukushima-anniversary-toyota-in/how-toyota-thrives-when-the-chips-are-down-idUSKBN2B1005.
16. Temperature sensor tutorial - maxim. DigiKey. (n.d.). Retrieved April 4, 2022, from https://www.digikey.com/en/pdf/m/maxim/temperature-sensor-tutorial
17. Totten, G. E. (Ed.). (2019). Steel heat treatment: Equipment and process design. CRC Press.
18. Towner, Jr., Walter T. 2013. The Design of Engineering Education As a Manufacturing System. : Worcester Polytechnic Institute.
19. TTT Curve. MECHTECH GURU. (1970, January 1). Retrieved April 11, 2022, from https://www.mechtechguru.com/2020/08/ttt-curve.html

20. V. Harutunian, M. Nordlund, D. Tate, et al. Decision Making and Software Tools for Product Development Based on Axiomatic Design Theory CIRP Annals - Manufacturing Technology, 45 (1) (1996)
21. What is the holloman-jaffe parameter? TWI. (n.d.). Retrieved April 4, 2022, from https://www.twi-global.com/technical-knowledge/faqs/faq-what-is-the-holloman-jaffe-parameter
22. Why control heat treatment temperature? Industrial Metallurgists.(2021, December 13). Retrieved April 4, 2022, from https://www.imetllc.com/why-control-heat-treatment-temperature/

APPENDICES

APPENDIX A: Model Output Raw Data

This is the raw data dump from the model run described in the Results section. It shows what the actual hardness was, what the model predicted the hardness would be, how accurate that prediction was in percentage, and the error, or difference between the actual and predicted value.

Hardness (HV)	Predicted Hardness (HV)	Accuracy (%)	Error
378	337.4424473	89.27049	-40.5576
378	367.8706312	97.32027	-10.1294
378	343.1127142	90.77056	-34.8873
389	343.4574181	88.2924	-45.5426
389	330.8285696	85.0459	-58.1714
390	336.5091877	86.28441	-53.4908
390	377.4991227	96.79465	-12.5009
390	331.4503354	84.98727	-58.5497
380	330.3526549	86.93491	-49.6473
380	337.9197757	88.92626	-42.0802

380	380.9896273	99.73957	0.989627
380	348.8850552	91.81186	-31.1149
380	338.6094951	89.10776	-41.3905
380	344.5907766	90.68178	-35.4092
380	361.0230612	95.00607	-18.9769
389	359.8225638	92.49937	-29.1774
389	368.0039027	94.60255	-20.9961
389	330.8970743	85.06352	-58.1029
385	357.3887663	92.82825	-27.6112
385	353.504198	91.81927	-31.4958
385	319.5879056	83.00985	-65.4121
397	381.8605881	96.18655	-15.1394
397	382.4584887	96.33715	-14.5415
397	343.9799131	86.64481	-53.0201
397	380.1574594	95.75755	-16.8425
397	354.8349477	89.37908	-42.1651
397	379.4819729	95.5874	-17.518
397	393.0948306	99.01633	-3.90517

394	384.9378122	97.69995	-9.06219
394	353.4960033	89.7198	-40.504
394	353.4502856	89.70819	-40.5497
394	372.5514737	94.55621	-21.4485
394	366.4001644	92.99497	-27.5998
394	336.8042922	85.48332	-57.1957
394	336.3569151	85.36978	-57.6431
394	362.776628	92.07529	-31.2234
378	381.6970778	99.02194	3.697078
378	354.9390346	93.89922	-23.061
378	351.279293	92.93103	-26.7207
378	341.7043785	90.39798	-36.2956
378	381.5865694	99.05117	3.586569
361	336.3572745	93.17376	-24.6427
361	312.7872034	86.64465	-48.2128
361	324.9858789	90.02379	-36.0141
335	327.7025129	97.82165	-7.29749
335	325.8919906	97.28119	-9.10801

335	331.7649921	99.03433	-3.23501
327	325.5169999	99.54648	-1.483
327	322.60833	98.65698	-4.39167
327	325.1448366	99.43267	-1.85516
327	335.2396824	97.48022	8.239682
327	315.5831486	96.50861	-11.4169
322	321.1957891	99.75025	-0.80421
322	336.2909143	95.56183	14.29091
322	316.5841701	98.31807	-5.41583
322	313.8582152	97.4715	-8.14178
322	322.3039885	99.90559	0.303988
322	307.3217078	95.44152	-14.6783
330	336.1018256	98.15096	6.101826
330	336.8047235	97.93796	6.804723
330	324.9775884	98.47806	-5.02241
318	325.1471129	97.75248	7.147113
318	317.1131586	99.72112	-0.88684
323	317.9234141	98.4283	-5.07659

320	328.4673741	97.35395	8.467374
320	328.365276	97.38585	8.365276
320	350.6076403	90.43511	30.60764
320	311.5316675	97.35365	-8.46833
320	321.66787	99.47879	1.66787
310	323.7864238	95.55277	13.78642
310	327.6926649	94.29269	17.69266
310	309.906508	99.96984	-0.09349
310	311.8034339	99.41825	1.803434
310	302.6851813	97.64038	-7.31482
320	299.8368453	93.69901	-20.1632
320	317.5621171	99.23816	-2.43788
320	279.4390879	87.32471	-40.5609
320	287.9035687	89.96987	-32.0964
310	336.8902166	91.32574	26.89022
310	298.0082085	96.13168	-11.9918
310	293.2053024	94.58236	-16.7947
310	283.2932196	91.38491	-26.7068

309	302.9296321	98.03548	-6.07037
309	301.8447109	97.68437	-7.15529
309	322.6687478	95.57646	13.66875
309	307.7787652	99.60478	-1.22123
309	291.6445143	94.38334	-17.3555
309	323.98445	95.15066	14.98445
311	307.6881684	98.9351	-3.31183
311	312.0061924	99.67647	1.006192
311	298.807406	96.07955	-12.1926
311	303.915067	97.72189	-7.08493
299	306.5174067	97.48582	7.517407
299	309.1013203	96.62163	10.10132
299	304.4703647	98.17045	5.470365
299	313.0033083	95.31662	14.00331
299	316.9482106	93.99725	17.94821
303	338.2301914	88.37287	35.23019
303	329.4624772	91.26651	26.46248
303	306.5647058	98.82353	3.564706

303	321.1082713	94.02367	18.10827
303	309.5092695	97.85173	6.509269
303	322.4546317	93.57933	19.45463
299	356.3316758	80.82553	57.33168
299	305.2661837	97.90429	6.266184
299	299.7371913	99.75345	0.737191
299	328.5226762	90.1262	29.52268
299	311.4811336	95.82571	12.48113
302	298.9900133	99.00332	-3.00999
302	293.1291421	97.06263	-8.87086
302	299.5795274	99.19852	-2.42047
298	308.7100361	96.40603	10.71004
298	284.7830769	95.56479	-13.2169
298	297.9367806	99.97879	-0.06322
283	275.6760022	97.41201	-7.324
283	286.8613761	98.63556	3.861376
283	267.1171208	94.38768	-15.8829
293	281.118609	95.94492	-11.8814

293	286.2388915	97.69245	-6.76111
293	278.9907284	95.21868	-14.0093
293	278.4868511	95.04671	-14.5131
293	277.9796492	94.8736	-15.0204
297	271.6098629	91.45113	-25.3901
297	271.6061669	91.44989	-25.3938
290	282.0406848	97.25541	-7.95932
290	292.3751831	99.18097	2.375183
290	299.4380134	96.74551	9.438013
290	275.3072893	94.93355	-14.6927
290	280.3641888	96.67731	-9.63581
290	279.1464753	96.25741	-10.8535
290	280.4910686	96.72106	-9.50893
289	272.5490229	94.30762	-16.451
289	273.5448988	94.65221	-15.4551
289	272.191404	94.18388	-16.8086
289	299.5893515	96.33586	10.58935
287	264.6530724	92.21361	-22.3469

287	281.8770486	98.215	-5.12295
287	277.6300991	96.73523	-9.3699
287	279.2692398	97.30636	-7.73076
287	273.5031407	95.29726	-13.4969
294	274.864345	93.49127	-19.1357
294	284.1675889	96.65564	-9.83241
294	294.9207531	99.68682	0.920753
294	280.6660084	95.46463	-13.334
294	271.3406184	92.29273	-22.6594
294	271.3317887	92.28972	-22.6682
285	297.0195749	95.78261	12.01957
285	275.7569726	96.75683	-9.24303
285	270.0610494	94.75826	-14.939
255	253.4361897	99.38674	-1.56381
255	247.672259	97.12638	-7.32774
255	252.619303	99.06639	-2.3807
255	253.3942339	99.37029	-1.60577
261	255.9663587	98.0714	-5.03364

261	268.1136256	97.27447	7.113626
261	258.6116186	99.08491	-2.38838
264	270.2211992	97.64349	6.221199
264	258.4700626	97.90533	-5.52994
264	259.6347921	98.34651	-4.36521
264	259.0047478	98.10786	-4.99525
264	256.0768671	96.99881	-7.92313
264	282.237489	93.09186	18.23749
264	262.1354231	99.29372	-1.86458
262	240.8891203	91.94241	-21.1109
262	258.2414441	98.56544	-3.75856
262	256.8277231	98.02585	-5.17228
262	242.5371207	92.57142	-19.4629
262	263.9833605	99.24299	1.983361
258	268.9733008	95.74678	10.9733
258	261.922523	98.47964	3.922523
258	254.4339406	98.61781	-3.56606
256	265.184463	96.41232	9.184463

256	256.9751675	99.61908	0.975168
256	245.5108449	95.90267	-10.4892
256	266.364258	95.95146	10.36426
256	263.474739	97.08018	7.474739
256	269.7172321	94.64171	13.71723
258	242.7453333	94.08734	-15.2547
258	252.5550933	97.88957	-5.44491
258	250.2121921	96.98147	-7.78781
260	263.1313169	98.79565	3.131317
258	247.2523713	95.83425	-10.7476
258	243.3018142	94.30303	-14.6982
227	230.3693101	98.51572	3.36931
227	228.2750858	99.43829	1.275086
227	230.0727409	98.64637	3.072741
236	234.5677833	99.39313	-1.43222
236	232.9667347	98.71472	-3.03327
233	211.4752324	90.7619	-21.5248
233	226.1231303	97.04855	-6.87687

233	214.5155415	92.06676	-18.4845
232	230.3055107	99.26962	-1.69449
232	234.2354733	99.03643	2.235473
229	230.4540454	99.36505	1.454045
229	210.3606085	91.86053	-18.6394
229	220.9663686	96.49186	-8.03363
229	232.6103857	98.42341	3.610386
229	210.3606085	91.86053	-18.6394
230	232.0829038	99.09439	2.082904
230	211.8476653	92.10768	-18.1523
230	233.1477247	98.63142	3.147725
230	229.2253967	99.66322	-0.7746
230	214.2164355	93.13758	-15.7836
230	226.8840919	98.64526	-3.11591
230	234.7007403	97.9562	4.70074
230	235.2354076	97.72374	5.235408
230	220.4141453	95.83224	-9.58585
230	224.4255905	97.57634	-5.57441

233	238.0586593	97.8289	5.058659
233	234.3714404	99.4114	1.37144
233	229.9774148	98.70275	-3.02259
233	230.3297713	98.85398	-2.67023
233	231.4309321	99.32658	-1.56907
233	224.4500577	96.3305	-8.54994
233	229.6365387	98.55645	-3.36346
233	210.3606085	90.28352	-22.6394
221	210.3606085	95.1858	-10.6394
221	216.1325976	97.79756	-4.8674
196	220.0264553	87.7416	24.02646
196	221.1585858	87.16399	25.15859
204	211.3746828	96.38496	7.374683
204	207.8414526	98.11694	3.841453
204	209.6052058	97.25235	5.605206
204	207.2741683	98.39502	3.274168
200	258.8869852	70.55651	58.88699
200	210.9104521	94.54477	10.91045

200	219.3365607	90.33172	19.33656
200	207.7847248	96.10764	7.784725
200	211.8170999	94.09145	11.8171
200	209.5852553	95.20737	9.585255
200	210.3289709	94.83551	10.32897
198	207.387624	95.25878	9.387624
198	209.2860145	94.29999	11.28601
202	205.5723155	98.23153	3.572316
200	218.4281592	90.78592	18.42816
200	202.4522461	98.77388	2.452246
200	204.437744	97.78113	4.437744
200	209.3658108	95.31709	9.365811
200	212.756856	93.62157	12.75686
201	202.6980716	99.15519	1.698072
201	201.2231301	99.88899	0.22313
201	210.4496298	95.29869	9.44963
201	207.8225443	96.6057	6.822544
201	210.7311694	95.15862	9.731169

201	205.4210373	97.80048	4.421037
201	207.2174405	96.90675	6.217441
206	213.4183642	96.39885	7.418364
206	195.8573695	95.07639	-10.1426
204	211.1333619	96.50325	7.133362
204	214.6021592	94.80286	10.60216
200	215.0343854	92.48281	15.03439
200	208.440692	95.77965	8.440692
		AVG: 95.37954766	

APPENDIX B: Arduino Code for Furnace Interface

```
//Impact of Heat Treating Carbon Steels MQP: Furnace Control Code

//MegunoLink and Related Libraries
#include <MegunoLink.h>
#include <CommandHandler.h>
#include <TCPCommandHandler.h>
#include <ArduinoTimer.h>
#include <CircularBuffer.h>
#include <EEPROMStore.h>
#include <Filter.h>

//Adafruit MAX31856 Thermocouple Amplifier Library
#include <Adafruit_MAX31856.h>

// Use software SPI: CS, DI, DO, CLK
Adafruit_MAX31856 maxthermo = Adafruit_MAX31856(13, 12, 11, 10);   //Setting SPI pins


//================================================================//
//Parameters

const int SSRPin = 7;    //Arduino pin connected to the solid state relay in the furnace
long CountedSeconds = 0;  //Continuously increasing variable as each second passes
int RunFlag = 0;      //Flag to begin running the furnace or not
float InitialTemp = 21.5;  //Room temperature
float DesiredTemp = 0.0;  //Instantiating desired temperature variable
float TargetTemp = 0.0;   //Instantiating target temperature variable
float MaxRampDownRate = 75; //  °C / hr  maximum cooldown rate of the furnace, measured
int NumStages;       //How many stages will be run
char PhaseMode = 0; // 0 = Ramp Up, 1 = Hold  //Which mode is the furnace in? Ramping or
holding?
char CurrentPhase = 0;
char CurrentStage = 0;
float CurrentStageStartTemp = 0.0;

int TotalHours = 0;
int TotalMinutes = 0;
int TotalSeconds = 0;


float TimeDelta;
float TempDelta;
float TempIncrement;

float StageTemp [8];
float StageHoldTime [8];
```

```
float StageRampRate [8];
float StageRampTime [8];
float AccumulatedTime [17];


CommandHandler<15> SerialCmds;
InterfacePanel ControlPanel;
XYPlot TempPlot;
ArduinoTimer SecTimer;

void EndProgram();

//================================================================//
//SETUP

void setup() {
 //Open and Begin Serial Communication
 Serial.begin(921600);
 while (!Serial) delay(10);


 //Thermocouple Setup
 Serial.println("MAX31856 thermocouple test");

 maxthermo.begin();

 maxthermo.setThermocoupleType(MAX31856_TCTYPE_K);

 Serial.print("Thermocouple type: ");
 switch (maxthermo.getThermocoupleType() ) {
  case MAX31856_TCTYPE_B: Serial.println("B Type"); break;
  case MAX31856_TCTYPE_E: Serial.println("E Type"); break;
  case MAX31856_TCTYPE_J: Serial.println("J Type"); break;
  case MAX31856_TCTYPE_K: Serial.println("K Type"); break;
  case MAX31856_TCTYPE_N: Serial.println("N Type"); break;
  case MAX31856_TCTYPE_R: Serial.println("R Type"); break;
  case MAX31856_TCTYPE_S: Serial.println("S Type"); break;
  case MAX31856_TCTYPE_T: Serial.println("T Type"); break;
  case MAX31856_VMODE_G8: Serial.println("Voltage x8 Gain mode"); break;
  case MAX31856_VMODE_G32: Serial.println("Voltage x8 Gain mode"); break;
  default: Serial.println("Unknown"); break;
 }

 //Set Initial Values to 0


 //Serial Commands from MegunoLink
 SerialCmds.AddCommand(F("RunFurnace"), RunFurnace);       //Begin logic to turn on heating
elements and run the furnace
```

```
 SerialCmds.AddCommand(F("StopFurnace"), StopFurnace);        //Stop the furnace and turn off the heating elements

   //Set the properties of each stage: Temperature, Holding time, Ramp Rate,
 SerialCmds.AddCommand(F("SetStage1"), SetStage1);
 SerialCmds.AddCommand(F("SetStage2"), SetStage2);
 SerialCmds.AddCommand(F("SetStage3"), SetStage3);
 SerialCmds.AddCommand(F("SetStage4"), SetStage4);
 SerialCmds.AddCommand(F("SetStage5"), SetStage5);
 SerialCmds.AddCommand(F("SetStage6"), SetStage6);
 SerialCmds.AddCommand(F("SetStage7"), SetStage7);
 SerialCmds.AddCommand(F("SetStage8"), SetStage8);

 SerialCmds.AddCommand(F("SetStages"), SetStages);           //Update the plot to show the temperature profile
 SerialCmds.AddCommand(F("ClearStages"), ClearStages);        //Clear the values set in the stage entry boxes and clear the plot

 SerialCmds.AddCommand(F("SelectStages"), SelectStages);

 pinMode(SSRPin, OUTPUT);
}

//================================================================//
//MAIN LOOP

void loop() {
 SerialCmds.Process();

 float Temp = maxthermo.readThermocoupleTemperature();

 if(RunFlag == 1){
  if(SecTimer.EllapsedSeconds() > CountedSeconds){

  float PlotMinutes = CountedSeconds/60.0;

  int RunningSeconds = CountedSeconds;
  int RunningMinutes = CountedSeconds / 60;
  int RunningHours = RunningMinutes / 60;
  RunningSeconds %= 60;
  RunningMinutes %= 60;
  RunningHours %= 24;

  String RunningTime;

  if(RunningHours < 10){
   RunningTime += "0";
  }
  RunningTime += String(RunningHours) + ":";
```

```
  if(RunningMinutes < 10){
   RunningTime += "0";
  }
  RunningTime += String(RunningMinutes) + ":";
  if(RunningSeconds < 10){
   RunningTime += "0";
  }
  RunningTime += String(RunningSeconds);

  //Display Elapsed Time
  ControlPanel.SetText(F("TimeElapsed"), RunningTime);

  float RunningDecimalMinutes = CountedSeconds / 60.0;
  float RemainingDecimalMinutes = AccumulatedTime[16] - RunningDecimalMinutes;

//  TotalHours = (AccumulatedTime[16] / 60);
//  TotalMinutes = (AccumulatedTime[16] - (TotalHours * 60));
//  TotalSeconds = (AccumulatedTime[16] - (TotalHours * 60) - TotalMinutes)*60;


  int RemainingHours = (RemainingDecimalMinutes / 60);
  int RemainingMinutes = (RemainingDecimalMinutes - (RemainingHours * 60));
  int RemainingSeconds = (RemainingDecimalMinutes - (RemainingHours * 60) -
RemainingMinutes)*60;



  String RemainingTime;

  if(RemainingHours < 10){
   RemainingTime += "0";
  }
  RemainingTime += String(RemainingHours) + ":";
  if(RemainingMinutes < 10){
   RemainingTime += "0";
  }
  RemainingTime += String(RemainingMinutes) + ":";
  if(RemainingSeconds < 10){
   RemainingTime += "0";
  }
  RemainingTime += String(RemainingSeconds);



  //Display Remaining Time
  ControlPanel.SetText(F("RemainingTime"), RemainingTime);
```

```
//  Serial.print("Cold Junction Temp: ");
//  Serial.println(maxthermo.readCJTemperature());
//
//  Serial.print("Thermocouple Temp: ");
//  Serial.println(maxthermo.readThermocoupleTemperature());
  // Check and print any faults
  uint8_t fault = maxthermo.readFault();
  if (fault) {
   if (fault & MAX31856_FAULT_CJRANGE) Serial.println("Cold Junction Range Fault");
   if (fault & MAX31856_FAULT_TCRANGE) Serial.println("Thermocouple Range Fault");
   if (fault & MAX31856_FAULT_CJHIGH) Serial.println("Cold Junction High Fault");
   if (fault & MAX31856_FAULT_CJLOW)  Serial.println("Cold Junction Low Fault");
   if (fault & MAX31856_FAULT_TCHIGH) Serial.println("Thermocouple High Fault");
   if (fault & MAX31856_FAULT_TCLOW)  Serial.println("Thermocouple Low Fault");
   if (fault & MAX31856_FAULT_OVUV)  Serial.println("Over/Under Voltage Fault");
   if (fault & MAX31856_FAULT_OPEN)  Serial.println("Thermocouple Open Fault");
  }

  TempPlot.SendData(F("Programmed Temperature"), PlotMinutes, DesiredTemp,
XYPlot::Black, XYPlot::Solid, 1 ,XYPlot::NoMarker);
  //Plot Furnace Temperature
  TempPlot.SendData(F("Furnace Temperature"), PlotMinutes, Temp, XYPlot::Red,
XYPlot::Solid, 1 ,XYPlot::NoMarker);

  //Display Furnace Temperature
  ControlPanel.SetText(F("FurnaceTemp"), Temp);

  if(CountedSeconds == 0){
    TimeDelta = (StageRampTime[CurrentStage] * 60.0);
    TempDelta = (StageTemp[CurrentStage] - InitialTemp);
    TempIncrement = (TempDelta / TimeDelta);
    TargetTemp = StageTemp[CurrentStage];
    DesiredTemp = InitialTemp;
  }

  //Controlling the Heating Elements
  if(PhaseMode == 0){
   DesiredTemp += TempIncrement;
//   Serial.print("Desired Temp: ");
//   Serial.println(DesiredTemp);
  }


  //Change PhaseMode Depented on Stage
  if((CountedSeconds / 60.0) >= AccumulatedTime[CurrentPhase]){
   CurrentPhase++;
//   Serial.println("PhaseSelection Test");
   //Starting a new stage, Ramping
   if((CurrentPhase % 2) == 0){
```

```
    CurrentStage++;
    CurrentStageStartTemp = StageTemp[CurrentStage - 1];
    PhaseMode = 0;
    TimeDelta = (StageRampTime[CurrentStage] * 60.0);
    TempDelta = (StageTemp[CurrentStage] - CurrentStageStartTemp);
    TempIncrement = (TempDelta / TimeDelta);
    TargetTemp = StageTemp[CurrentStage];
    DesiredTemp = CurrentStageStartTemp;
//    Serial.println("Flip");
   }
   //Done Ramping, now Holding
   else{
    PhaseMode = 1;
    TargetTemp = StageTemp[CurrentStage];
    DesiredTemp = TargetTemp;
//    Serial.println("Flop");
   }
  }

  ControlPanel.SetText(F("ProgTemp"), DesiredTemp);

  if(Temp <= DesiredTemp - 0.5){
   digitalWrite(SSRPin, HIGH);
  }
  else if(Temp >= DesiredTemp + 0.5){
   digitalWrite(SSRPin, LOW);
  }
  else{
   digitalWrite(SSRPin, LOW);
  }

  CountedSeconds++;
  }
  if(CountedSeconds >= AccumulatedTime[16]*60){
  EndProgram();
  }
 }

}


//=================================================================//
//Serial Command Functions

//Run the furnace when the Start button is pressed
void RunFurnace(CommandParameter &p){
 SecTimer.Reset();
 ControlPanel.ShowControl(F("BtnStop"), true);
 ControlPanel.ShowControl(F("BtnRun"), false);
```

```
 ControlPanel.DisableControl(F("StageSelection"));
 TempPlot.Clear("Furnace Temperature");
 RunFlag = 1;
}

//Stop the furnace and temperature monitoring when the program reaches the end
void EndProgram(){
 Serial.println("END PROGRAM");
 SecTimer.Reset();
 ControlPanel.ShowControl(F("BtnStop"), false);
 ControlPanel.ShowControl(F("BtnRun"), true);
 ControlPanel.EnableControl(F("StageSelection"));
 ControlPanel.SetText(F("TimeElapsed"), "00:00:00");
 digitalWrite(SSRPin, LOW);
 RunFlag = 0;
 CountedSeconds = 0;
}

//Stop the furnace when the Stop button is pressed
void StopFurnace(CommandParameter &p){
 SecTimer.Reset();
 ControlPanel.ShowControl(F("BtnStop"), false);
 ControlPanel.ShowControl(F("BtnRun"), true);
 ControlPanel.EnableControl(F("StageSelection"));
 ControlPanel.SetText(F("TimeElapsed"), "00:00:00");
 digitalWrite(SSRPin, LOW);
 RunFlag = 0;
 CountedSeconds = 0;
}


void SetStage1(CommandParameter &p){
 StageTemp[0] = p.NextParameterAsInteger();
 StageHoldTime[0] = p.NextParameterAsInteger();
 StageRampRate[0] = p.NextParameterAsInteger();
 StageRampTime[0] = (abs((StageTemp[0]-InitialTemp)) / StageRampRate[0])*60;
}

void SetStage2(CommandParameter &p){
 StageTemp[1] = p.NextParameterAsInteger();
 StageHoldTime[1] = p.NextParameterAsInteger();
 StageRampRate[1] = p.NextParameterAsInteger();
 if(StageTemp[0] > StageTemp[1]){
  if(StageRampRate[1] > MaxRampDownRate){
   StageRampRate[1] = MaxRampDownRate;
  }
 }
 StageRampTime[1] = (abs((StageTemp[1] - StageTemp[0])) / StageRampRate[1])*60;
}
```

```
void SetStage3(CommandParameter &p){
 StageTemp[2] = p.NextParameterAsInteger();
 StageHoldTime[2] = p.NextParameterAsInteger();
 StageRampRate[2] = p.NextParameterAsInteger();
 if(StageTemp[1] > StageTemp[2]){
  if(StageRampRate[2] > MaxRampDownRate){
   StageRampRate[2] = MaxRampDownRate;
  }
 }
 StageRampTime[2] = (abs((StageTemp[2] - StageTemp[1])) / StageRampRate[2])*60;
}

void SetStage4(CommandParameter &p){
 StageTemp[3] = p.NextParameterAsInteger();
 StageHoldTime[3] = p.NextParameterAsInteger();
 StageRampRate[3] = p.NextParameterAsInteger();
 if(StageTemp[2] > StageTemp[3]){
  if(StageRampRate[3] > MaxRampDownRate){
   StageRampRate[3] = MaxRampDownRate;
  }
 }
 StageRampTime[3] = (abs((StageTemp[3] - StageTemp[2])) / StageRampRate[3])*60;
}

void SetStage5(CommandParameter &p){
 StageTemp[4] = p.NextParameterAsInteger();
 StageHoldTime[4] = p.NextParameterAsInteger();
 StageRampRate[4] = p.NextParameterAsInteger();
 if(StageTemp[3] > StageTemp[4]){
  if(StageRampRate[4] > MaxRampDownRate){
   StageRampRate[4] = MaxRampDownRate;
  }
 }
 StageRampTime[4] = (abs((StageTemp[4] - StageTemp[3])) / StageRampRate[4])*60;
}

void SetStage6(CommandParameter &p){
 StageTemp[5] = p.NextParameterAsInteger();
 StageHoldTime[5] = p.NextParameterAsInteger();
 StageRampRate[5] = p.NextParameterAsInteger();
 if(StageTemp[4] > StageTemp[5]){
  if(StageRampRate[5] > MaxRampDownRate){
   StageRampRate[5] = MaxRampDownRate;
  }
 }
 StageRampTime[5] = (abs((StageTemp[5] - StageTemp[4])) / StageRampRate[5])*60;
}
```

```
void SetStage7(CommandParameter &p){
 StageTemp[6] = p.NextParameterAsInteger();
 StageHoldTime[6] = p.NextParameterAsInteger();
 StageRampRate[6] = p.NextParameterAsInteger();
 if(StageTemp[5] > StageTemp[6]){
  if(StageRampRate[6] > MaxRampDownRate){
   StageRampRate[6] = MaxRampDownRate;
  }
 }
 StageRampTime[6] = (abs((StageTemp[6] - StageTemp[5])) / StageRampRate[6])*60;
}

void SetStage8(CommandParameter &p){
 StageTemp[7] = p.NextParameterAsInteger();
 StageHoldTime[7] = p.NextParameterAsInteger();
 StageRampRate[7] = p.NextParameterAsInteger();
 if(StageTemp[6] > StageTemp[7]){
  if(StageRampRate[7] > MaxRampDownRate){
   StageRampRate[7] = MaxRampDownRate;
  }
 }
 StageRampTime[7] = (abs((StageTemp[7] - StageTemp[6])) / StageRampRate[7])*60;
}


//Set the stages based on the values entered in the stage parameter boxes and update the plot
to show the temperature profile
void SetStages(CommandParameter &p){
 TempPlot.Clear("Temperature Profile");
 TempPlot.Clear("Programmed Temperature");
 TempPlot.Clear("Furnace Temperature");

 InitialTemp = maxthermo.readThermocoupleTemperature();

 AccumulatedTime[0] = StageRampTime[0];
 AccumulatedTime[1] = StageHoldTime[0]+AccumulatedTime[0];

 int i;
 for(i=1; i<NumStages; i++){
  AccumulatedTime[2*i] = StageRampTime[i]+AccumulatedTime[(2*i)-1];
  AccumulatedTime[(2*i)+1] = StageHoldTime[i]+AccumulatedTime[2*i];
 }

 AccumulatedTime[16] = AccumulatedTime[(2*(i-1))+1];   // + (((StageTemp[(i-1)] - 21.5)) /
(MaxRampDownRate / 60.0));

 TotalHours = (AccumulatedTime[16] / 60);
 TotalMinutes = (AccumulatedTime[16] - (TotalHours * 60));
 TotalSeconds = (AccumulatedTime[16] - (TotalHours * 60) - TotalMinutes)*60;
```

```
String TotalTime;
if(TotalHours < 10){
 TotalTime += "0";
}
TotalTime += String(TotalHours) + ":";
if(TotalMinutes < 10){
 TotalTime += "0";
}
TotalTime += String(TotalMinutes) + ":";
if(TotalSeconds < 10){
 TotalTime += "0";
}
TotalTime += String(TotalSeconds);



ControlPanel.SetText(F("TotalTime"), TotalTime);

//TempPlot.SendData("Programmed Temperature", AccumulatedTime[16], 21.5,
XYPlot::Black, XYPlot::Solid, 1 ,XYPlot::NoMarker);

switch (NumStages){
 case 8:
  TempPlot.SendData("Temperature Profile", AccumulatedTime[15], StageTemp[7],
XYPlot::Black, XYPlot::Solid, 1 ,XYPlot::NoMarker);
  TempPlot.SendData("Temperature Profile", AccumulatedTime[14], StageTemp[7],
XYPlot::Black, XYPlot::Solid, 1 ,XYPlot::NoMarker);
 case 7:
  TempPlot.SendData("Temperature Profile", AccumulatedTime[13], StageTemp[6],
XYPlot::Black, XYPlot::Solid, 1 ,XYPlot::NoMarker);
  TempPlot.SendData("Temperature Profile", AccumulatedTime[12], StageTemp[6],
XYPlot::Black, XYPlot::Solid, 1 ,XYPlot::NoMarker);
 case 6:
  TempPlot.SendData("Temperature Profile", AccumulatedTime[11], StageTemp[5],
XYPlot::Black, XYPlot::Solid, 1 ,XYPlot::NoMarker);
  TempPlot.SendData("Temperature Profile", AccumulatedTime[10], StageTemp[5],
XYPlot::Black, XYPlot::Solid, 1 ,XYPlot::NoMarker);
 case 5:
  TempPlot.SendData("Temperature Profile", AccumulatedTime[9], StageTemp[4],
XYPlot::Black, XYPlot::Solid, 1 ,XYPlot::NoMarker);
  TempPlot.SendData("Temperature Profile", AccumulatedTime[8], StageTemp[4],
XYPlot::Black, XYPlot::Solid, 1 ,XYPlot::NoMarker);
 case 4:
  TempPlot.SendData("Temperature Profile", AccumulatedTime[7], StageTemp[3],
XYPlot::Black, XYPlot::Solid, 1 ,XYPlot::NoMarker);
  TempPlot.SendData("Temperature Profile", AccumulatedTime[6], StageTemp[3],
XYPlot::Black, XYPlot::Solid, 1 ,XYPlot::NoMarker);
 case 3:
```

```
   TempPlot.SendData("Temperature Profile", AccumulatedTime[5], StageTemp[2],
XYPlot::Black, XYPlot::Solid, 1 ,XYPlot::NoMarker);
   TempPlot.SendData("Temperature Profile", AccumulatedTime[4], StageTemp[2],
XYPlot::Black, XYPlot::Solid, 1 ,XYPlot::NoMarker);
  case 2:
   TempPlot.SendData("Temperature Profile", AccumulatedTime[3], StageTemp[1],
XYPlot::Black, XYPlot::Solid, 1 ,XYPlot::NoMarker);
   TempPlot.SendData("Temperature Profile", AccumulatedTime[2], StageTemp[1],
XYPlot::Black, XYPlot::Solid, 1 ,XYPlot::NoMarker);
  case 1:
   TempPlot.SendData("Temperature Profile", AccumulatedTime[1], StageTemp[0],
XYPlot::Black, XYPlot::Solid, 1 ,XYPlot::NoMarker);
   TempPlot.SendData("Temperature Profile", AccumulatedTime[0], StageTemp[0],
XYPlot::Black, XYPlot::Solid, 1 ,XYPlot::NoMarker);
 }

 TempPlot.SendData("Temperature Profile", 0, InitialTemp, XYPlot::Black, XYPlot::Solid,
1 ,XYPlot::NoMarker);
}

//Clear all values entered in the stage parameters
void ClearStages(CommandParameter &p){
 TempPlot.Clear("Temperature Profile");
 ControlPanel.SetNumber(F("Stage1Temp"), 0); ControlPanel.SetNumber(F("Stage1Time"),
0); ControlPanel.SetNumber(F("Stage1Ramp"), 0);
 ControlPanel.SetNumber(F("Stage2Temp"), 0); ControlPanel.SetNumber(F("Stage2Time"),
0); ControlPanel.SetNumber(F("Stage2Ramp"), 0);
 ControlPanel.SetNumber(F("Stage3Temp"), 0); ControlPanel.SetNumber(F("Stage3Time"),
0); ControlPanel.SetNumber(F("Stage3Ramp"), 0);
 ControlPanel.SetNumber(F("Stage4Temp"), 0); ControlPanel.SetNumber(F("Stage4Time"),
0); ControlPanel.SetNumber(F("Stage4Ramp"), 0);
 ControlPanel.SetNumber(F("Stage5Temp"), 0); ControlPanel.SetNumber(F("Stage5Time"),
0); ControlPanel.SetNumber(F("Stage5Ramp"), 0);
 ControlPanel.SetNumber(F("Stage6Temp"), 0); ControlPanel.SetNumber(F("Stage6Time"),
0); ControlPanel.SetNumber(F("Stage6Ramp"), 0);
 ControlPanel.SetNumber(F("Stage7Temp"), 0); ControlPanel.SetNumber(F("Stage7Time"),
0); ControlPanel.SetNumber(F("Stage7Ramp"), 0);
 ControlPanel.SetNumber(F("Stage8Temp"), 0); ControlPanel.SetNumber(F("Stage8Time"),
0); ControlPanel.SetNumber(F("Stage8Ramp"), 0);
}

//Update the control panel to the selected number of stages
void SelectStages(CommandParameter &p){
 NumStages = p.NextParameterAsInteger();
 switch (NumStages){
  case 1:
   ControlPanel.ShowControl(F("Stage1"), true);
   ControlPanel.ShowControl(F("Stage2"), false);
   ControlPanel.ShowControl(F("Stage3"), false);
```

```
 ControlPanel.ShowControl(F("Stage4"), false);
 delay(10);
 ControlPanel.ShowControl(F("Stage5"), false);
 ControlPanel.ShowControl(F("Stage6"), false);
 ControlPanel.ShowControl(F("Stage7"), false);
 ControlPanel.ShowControl(F("Stage8"), false);
 break;
case 2:
 ControlPanel.ShowControl(F("Stage1"), true);
 ControlPanel.ShowControl(F("Stage2"), true);
 ControlPanel.ShowControl(F("Stage3"), false);
 delay(10);
 ControlPanel.ShowControl(F("Stage4"), false);
 ControlPanel.ShowControl(F("Stage5"), false);
 ControlPanel.ShowControl(F("Stage6"), false);
 ControlPanel.ShowControl(F("Stage7"), false);
 ControlPanel.ShowControl(F("Stage8"), false);
 break;
case 3:
 ControlPanel.ShowControl(F("Stage1"), true);
 ControlPanel.ShowControl(F("Stage2"), true);
 ControlPanel.ShowControl(F("Stage3"), true);
 delay(10);
 ControlPanel.ShowControl(F("Stage4"), false);
 ControlPanel.ShowControl(F("Stage5"), false);
 ControlPanel.ShowControl(F("Stage6"), false);
 ControlPanel.ShowControl(F("Stage7"), false);
 ControlPanel.ShowControl(F("Stage8"), false);
 break;
case 4:
 ControlPanel.ShowControl(F("Stage1"), true);
 ControlPanel.ShowControl(F("Stage2"), true);
 ControlPanel.ShowControl(F("Stage3"), true);
 delay(10);
 ControlPanel.ShowControl(F("Stage4"), true);
 ControlPanel.ShowControl(F("Stage5"), false);
 ControlPanel.ShowControl(F("Stage6"), false);
 ControlPanel.ShowControl(F("Stage7"), false);
 ControlPanel.ShowControl(F("Stage8"), false);
 break;
case 5:
 ControlPanel.ShowControl(F("Stage1"), true);
 ControlPanel.ShowControl(F("Stage2"), true);
 ControlPanel.ShowControl(F("Stage3"), true);
 delay(10);
 ControlPanel.ShowControl(F("Stage4"), true);
 ControlPanel.ShowControl(F("Stage5"), true);
 ControlPanel.ShowControl(F("Stage6"), false);
 ControlPanel.ShowControl(F("Stage7"), false);
```

```
   ControlPanel.ShowControl(F("Stage8"), false);
   break;
  case 6:
   ControlPanel.ShowControl(F("Stage1"), true);
   ControlPanel.ShowControl(F("Stage2"), true);
   ControlPanel.ShowControl(F("Stage3"), true);
   delay(10);
   ControlPanel.ShowControl(F("Stage4"), true);
   ControlPanel.ShowControl(F("Stage5"), true);
   ControlPanel.ShowControl(F("Stage6"), true);
   ControlPanel.ShowControl(F("Stage7"), false);
   ControlPanel.ShowControl(F("Stage8"), false);
   break;
  case 7:
   ControlPanel.ShowControl(F("Stage1"), true);
   ControlPanel.ShowControl(F("Stage2"), true);
   ControlPanel.ShowControl(F("Stage3"), true);
   delay(10);
   ControlPanel.ShowControl(F("Stage4"), true);
   ControlPanel.ShowControl(F("Stage5"), true);
   ControlPanel.ShowControl(F("Stage6"), true);
   ControlPanel.ShowControl(F("Stage7"), true);
   ControlPanel.ShowControl(F("Stage8"), false);
   break;
  case 8:
   ControlPanel.ShowControl(F("Stage1"), true);
   ControlPanel.ShowControl(F("Stage2"), true);
   ControlPanel.ShowControl(F("Stage3"), true);
   delay(10);
   ControlPanel.ShowControl(F("Stage4"), true);
   ControlPanel.ShowControl(F("Stage5"), true);
   ControlPanel.ShowControl(F("Stage6"), true);
   ControlPanel.ShowControl(F("Stage7"), true);
   ControlPanel.ShowControl(F("Stage8"), true);
   break;
 }
}
```