2022

Saturn Engineering Solutions

Critical Design Review

GAS TURBINE ENGINE UPGRADES

April 11, 2022

CLIENT: Dr. Loren Sumner

MANAGER: Dr. Wade Shaw

Mary Lichtenwalner

Barrett McDonald

Ryan McMillan

Table of Contents

List of Figures

List of Tables

Executive Summary

Our Client, Dr. Sumner, has tasked our company, Saturn Engineering Solutions, with revitalizing his gas turbine engine. His major requests include improving the maneuverability of the transport cart, instrumenting the cart at various locations within the system, and creating a mechanism that will allow two people to start the engine. Improving the maneuverability of the cart allows students to spend more time running experiments and less time transporting the cart to a safe location for tests. Instrumenting the system with electronic sensors for temperatures and pressures allows students to diagnose why the engine is not starting. Students can also use collected data to determine the efficiency of the engine, which allows future projects to improve efficiency by observing the collected data. Finally, creating a system that requires only two people to start the engine allows fewer students to be present for safe operation of the engine, meaning more student groups have access to the engine.

The existing cart was cleaned, painted, and rebuilt with engine components closer to the center of the cart. This entailed removing old angle iron supports as well as adding new supports. All engine components were then cleaned and added to the cart. To instrument the cart in a manner that defines each state in the system, pressure and temperature must be recorded in the atmosphere and after the compressor section. Temperature must also be measured after the combustion chamber and at the exhaust. To take measurements, SES utilized a digital system in which an Arduino Uno controls an array of sensors. This design allows for the recording of measurements automatically. Ambient pressure and compressed air pressure are measured using pressure transducers. Ambient temperature and ambient humidity are measured by a temperature/humidity sensor. The temperature of compressed air, temperature after the combustion chamber, and temperature of the exhaust are measured using thermocouples. Finally, the compressor shaft rotational speed is measured using an infrared sensor module. The user can connect their laptop to the Arduino Uno and view live measurements on a windows application. All these readings can be easily viewed during engine operation and then exported to an excel file for further calculations/observations.

SES was able to improve the maneuverability of the engine, instrument the engine components with accurate sensors, and develop a system that allows for two operators to easily run the gas turbine engine. All of these were requests given to SES by Dr. Sumner. Saturn Engineering Solutions has completed the requests made by their client and will allow students in the future to better develop this gas turbine engine at Mercer University.

Glossary

Arduino – A single-board microcontroller development board with a multitude of programming libraries and accessories

Bearing – A machine element that guides, supports, and reduces friction between fixed and moving parts.

Cart – The physical structure that supports the gas turbine engine

Combustion Chamber – Also known as a combustor, in a jet engine this chamber provides a continuous flame by mixing compressed air with fuel. The combustion chamber lies between the compressor and turbine.

Compressor – Turbocharger inlet component used to add energy and pressure to working fluid.

Engine Control Tool – Computer program that records data from the sensors, displays the data in real time, and outputs the data in an Excel spreadsheet

Gas Turbine Engine – The compressor, journal bearing, pipes, combustion chamber, and turbine

GUI – Graphical user interface, a window that allows the user to control a computer program

Supporting Components – Oil tank, oil pump, oil filter, heat exchanger, spark plug electronics, and propane tank

Turbine – Turbocharger exit component that uses kinetic energy of the exhaust gas to directly rotate the centralized shaft connecting to the compressor

Turbocharger – A device normally within an internal combustion that converts exhaust kinetic energy into usable shaft work that powers the compressor

1. Introduction

The Mercer University gas turbine engine, belonging to the School of Engineering, has gone unused for several years. Dr. Loren Sumner, our client, and Associate Professor of Mechanical Engineering at Mercer has requested that our team, Saturn Engineering Solutions (SES), design and install upgrades to the gas turbine engine.

In the future, Dr. Sumner would like to use the gas turbine engine for class demonstrations, and he would like it to be available for future senior design projects. To meet these requirements, SES has rebuilt the cart that holds the engine and has installed an electronic system to monitor the engine as it is running. The engine in its initial state is shown in Figure 1 below.

Figure 1. Initial state of gas turbine engine.

2. Summary of Preliminary Design Review

The following sections provide a summary of the design phase of this project. Out of three potential design solutions, Saturn Engineering Solutions elected to refurbish the existing cart structure and provide a control system that records and displays sensor data using an Arduino Uno.

2.1 Project Description

There are two main objectives for this project. A new cart will be designed and built for the gas turbine engine, and the engine will be instrumented with an array of sensors. Sensors will be installed to record four temperature measurements, two pressure measurements, one humidity measurement, and a measurement of the rotational speed of the turbine. Having these measurements will allow the operator to determine if the engine is running efficiently and safely.

There were certain requirements requested by the client for both the cart and electronic system. These feasibility criteria are shown in Table 1.

Table 1. Feasibility Criteria.

Cart Design
Space for a second turbine in the future
Removable fuel tank when engine is running
New wheels for cart
Electronic System
Has a gas emergency shutoff
Spark plug has its own switch
Measures pressure of compressed air, temperature after combustion chamber, and shaft speed

To assist with comparing feasible designs, three merit criteria were considered for the cart, and six merit criteria were considered for the electronic system. These are listed in Table 2.

Table 2. Merit criteria.

Cart Design
Aesthetics
Safety
Cost
Electronic System
Takes additional measurements (10 %)
Can record measurements (30 %)
Operation simplicity (10 %)
Maintenance accessibility (10 %)
Safety (20 %)
Cost (20 %)

2.2 Roles and Responsibilities of Team Members

Saturn Engineering Solutions consists of three team members. Mary Lichtenwalner, an Electrical Engineer, will be primarily responsible for the design and implementation of the electronic system. Barrett McDonald, a Mechanical Engineer, will lead the cart fabrication. Ryan McMillan, a Mechanical Engineer, will be responsible for the solid modeling required for the cart. Resumes for the SES team may be viewed in Appendix I: Resumes.

2.3 Visual Rendering of PDR Design

The following image, Figure 2, represents the initial cart design and GUI design proposed by SES. This cart design moved the engine mounting location farther forward on the cart and repaired old supporting structures. The GUI was created to display the data from the sensors.

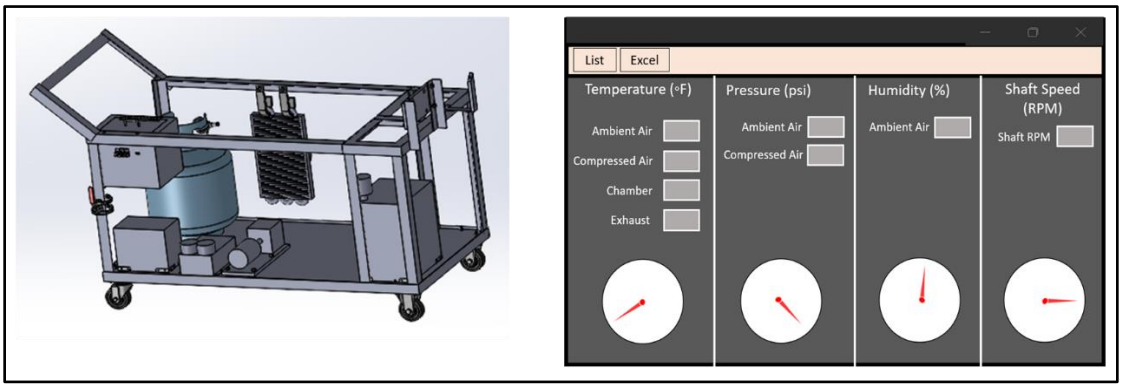


Figure 2. Original cart design and proposed GUI.

3. Work Accomplished

The following section outlines the work SES completed to take the given components and rework them to provide an excellent product for our client.

3.1 Design Revisions

Throughout the fabrication of the cart and electronic system for the gas turbine engine, SES had to make multiple design alterations. All design revisions were done to improve the system design, lower the overall budget, and meet the client's criteria more effectively.

3.1.1 Cart Design Revisions

During the construction of the steel cart, a few design aspects were altered from the proposed design. The cart design alterations impacted the electrical box, oil tank mounting, exhaust temperature gauge bracket, and overall vertical supports. The location of the electrical box was moved from the top corner of the cart to the lower corner where it is mounted on the bottom base board. This change was made for stability and aesthetic purposes. This alteration can be seen below in Figure 3.

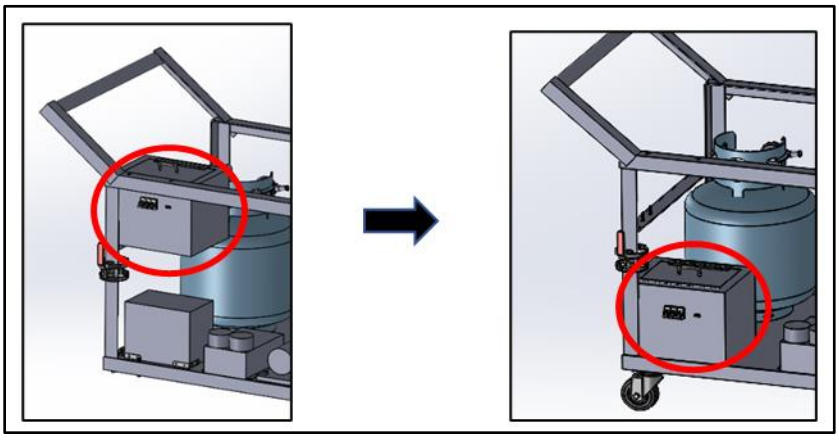


Figure 3. Electrical box location revision.

The oil tank was originally designed to be mounted on the bottom base board of the cart, but this added unnecessary stresses on the oil lines. To reduce these stresses, an additional set of mounting brackets was implemented on the front of the cart. This new mounting system holds the oil tank securely to the cart by utilizing a set screw to securely fasten the oil tank in place. Mounting the oil tank along the front of the cart also increases the serviceability of the system and allows the oil tank to easily be drained and refilled. This change is illustrated by Figure 4.

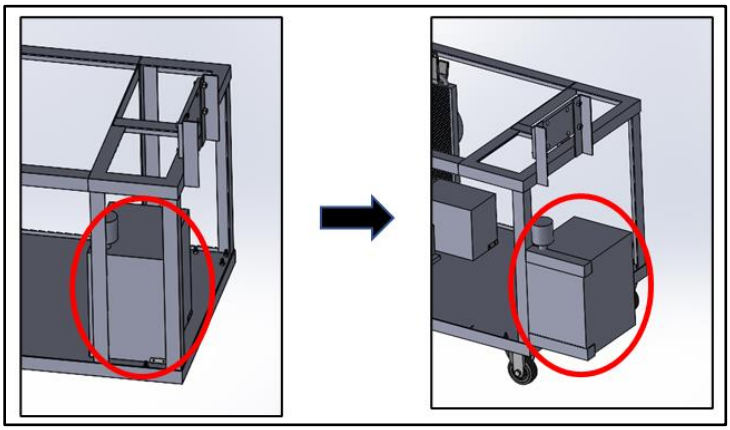


Figure 4. Oil tank location revision.

The bracket designed to hold the exhaust thermocouple in place was also altered. This bracket was altered to be easily bolted on/off instead of welded in place. This was requested by our client so that a group could easily implement other components down flow of the exhaust by removing the temperature sensor bracket.

The final alteration to the cart design was implementing four rather than six vertical supports connecting the upper and lower frames of the cart. This promoted a more open design for the cart, making the inside of the cart more easily accessible. This alteration is depicted in Figure 5.

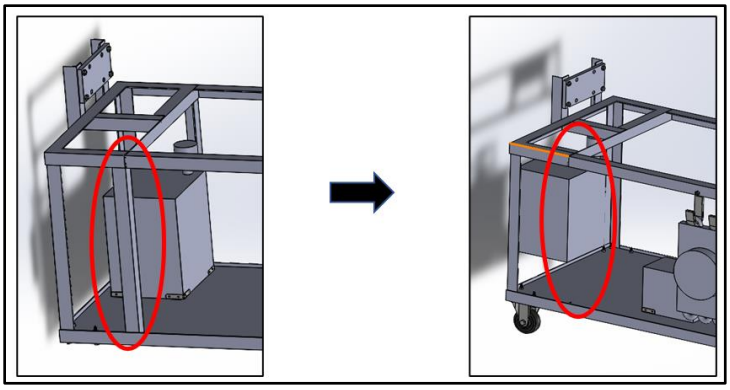


Figure 5. Removed vertical support.

The structural analysis was performed a second time with only four vertical supports. The results of this analysis are shown in Appendix B: Revised Supporting Calculations, and the results clearly demonstrate that the cart is structurally sound with only four supports. The design calls for the turbocharger assembly, tubing, and combustion chamber to be supported by one plate on

the cart. Because of this, a large shear force and series of moments act on this plate. The shear force is equal to 136.2 pounds force, and the net moment acting on the plate is equal to –13.5-foot pounds along the x-axis and –2.125-foot pounds along the y-axis.

The original cart design, shown in Figure 2, may be compared below with the altered cart design, drawn in Figure 6.

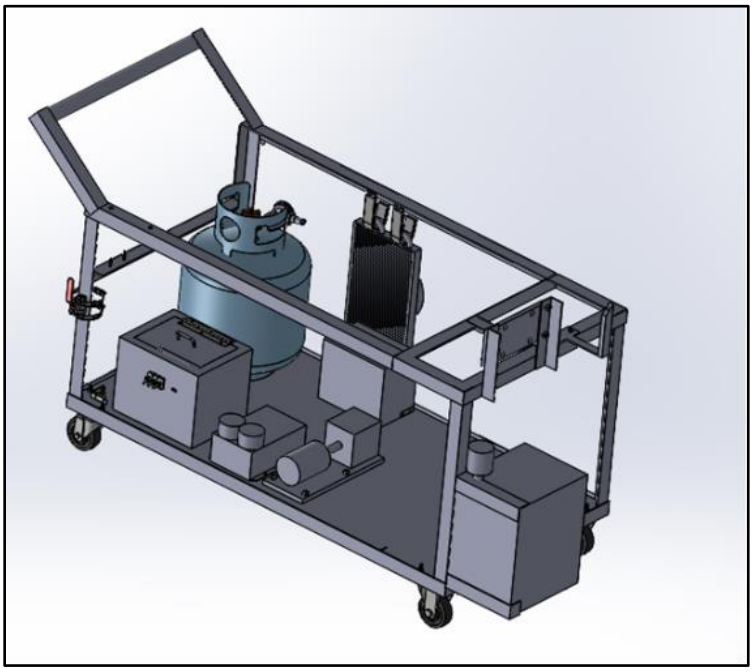


Figure 6. Final cart design.

3.1.2 Electronic System Design Revisions

The electronic system design also underwent some changes during implementation. Design changes were applied to both the physical components installed on the cart and to Engine Control Tool, the computer program responsible for recording the measured data.

The biggest revision realized for the sensor array was the removal of the Arduino controlled pitot tube sensor. The pitot tube sensor that was chosen during the preliminary design phase was out of stock, and SES was not able to procure a replacement that would be compatible with the Arduino. Because of this, it was determined that a handheld anemometer would be acceptable to determine flow rate of the exhaust. Therefore, the flow rate is not automatically recorded by Engine Control Tool. An input textbox was added to the Engine Control Tool GUI where the user inputs this flow rate value by hand. This value is then used in the Excel spreadsheet output. Figure 7 below illustrates the flow rate input text box in Engine Control Tool on the left and the handheld anemometer on the right.

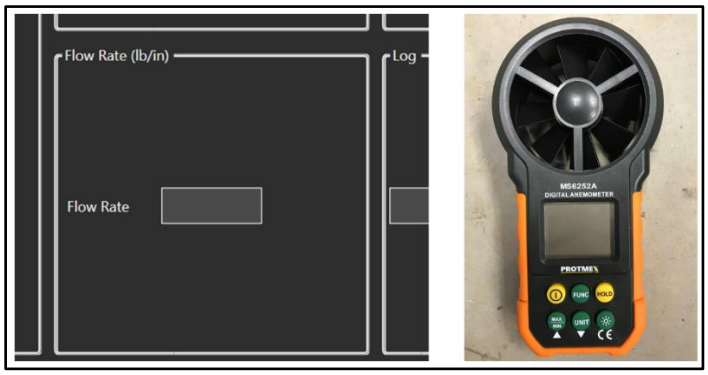


Figure 7. Flow rate input textbox and handheld anemometer.

There were multiple revisions made in the design of Engine Control Tool. It was originally proposed that Engine Control Tool display data in two views, a gauge view and a list view. However, SES determined that this was unnecessary to effectively display the measured values, so Engine Control Tool was built with one main view, the gauge view. This streamlines the program and makes it easy for the user to see the current sensor values while also watching the engine and controlling the propane flow.

Second, the final version of Engine Control Tool outputs two spreadsheets rather than the one spreadsheet initially proposed. The first spreadsheet outputs the values in imperial units, such as Fahrenheit and psi, while the second spreadsheet outputs values in metric values, that is, Celsius and pascals. Figure 8 shows an example of the Excel output. Note that two spreadsheets are created, one titled “Measured Values” and one titled “Metric.”

	A	B	C	D	E	F	G	H	I	J	K
1	Time Stamps		Temperature (ºF)				Pressure (psi)		Humidity (%)	Shaft Speed (RPM)	Flow Rate (idk)
2	Time	Log	Ambient	Compressed	Chamber	Exhaust	Ambient	Compressed	Ambient	Turbine	Exhaust
3	04/07/22 16:12:22		74.444	86.45	142.7	143.15	12.81	13.44	25.8	0	0
4	04/07/22 16:12:23		74.39	86	142.7	143.15	12.81	13.44	25.92	0	
5	04/07/22 16:12:24		74.426	86.45	143.15	143.6	12.81	13.23	26.27	0	
6	04/07/22 16:12:25		74.408	86.45	143.15	143.6	12.81	13.44	26.34	0	
7	04/07/22 16:12:26		74.372	86	143.15	143.6	12.81	13.44	26.34	0	
8	04/07/22 16:12:27		74.336	86.45	143.15	143.15	13.02	13.44	26.33	0	
9	04/07/22 16:12:28		74.354	86.45	143.6	143.15	12.81	13.44	26.17	0	
10	04/07/22 16:12:29		74.354	86.45	143.6	142.7	13.02	13.23	26.03	0	
11	04/07/22 16:12:30		74.372	86.45	144.05	142.25	13.02	13.44	25.89	0	
12	04/07/22 16:12:31		74.372	86.45	144.05	141.8	13.02	13.23	25.89	0	
13	04/07/22 16:12:32		74.372	86.9	144.5	141.35	13.02	13.44	26.21	0	
14	04/07/22 16:12:33		74.264	86.9	145.85	140.9	13.02	13.23	26.13	0	
15	04/07/22 16:12:35		74.192	86.9	147.2	140.45	12.81	13.44	25.8	0	
16	04/07/22 16:12:36		74.138	86.9	149	140	12.81	13.44	25.48	0	
17	04/07/22 16:12:37		74.156	86.9	150.35	139.1	13.02	13.23	25.33	0	
18	04/07/22 16:12:38		74.174	87.35	150.35	138.65	12.81	13.44	25.17	0	
19	04/07/22 16:12:39		74.192	87.35	150.8	138.2	12.81	13.23	25.24	0	
20	04/07/22 16:12:40		74.12	87.8	150.35	137.75	12.81	13.44	25.21	0	

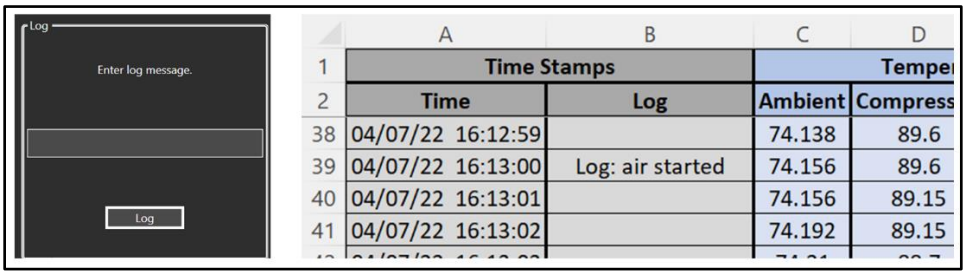
Measured Values | Metric

Figure 8. Dual Spreadsheet Excel Output.

This was done so that a user with familiarity in imperial values might get a better sense of the levels of recorded values, but calculations may still be easily performed with the recorded values in metric units.

Third, a log feature was added to Engine Control Tool. This feature allows the user to write a short message and log it at a specific time in the Excel spreadsheet. For example, the user could mention at some specific time the propane flow was decreased, or the user could mark when the

air flow from the blower was decreased. These notes would show up in the output Excel spreadsheet. SES determined that this feature would be helpful while testing the engine. The log input text box in Engine Control Tool and an example log Excel output is displayed below in Figure 9.



	A	B	C	D
1	Time Stamps			Tempe
2	Time	Log	Ambient	Compress
38	04/07/22 16:12:59		74.138	89.6
39	04/07/22 16:13:00	Log: air started	74.156	89.6
40	04/07/22 16:13:01		74.156	89.15
41	04/07/22 16:13:02		74.192	89.15

Figure 9. Log Message Input Feature and Example Excel Output.

3.2 Final Cart Design and Construction

The final cart construction began with disassembling the entire gas turbine engine. Once each component was removed, each was thoroughly cleaned to remove debris and other sediments contained within the system. This was done by forcing air through all the components individually followed by cleaning with water. Degreaser was then applied to each component, and each component was scrubbed. The components were then allowed to air dry while the cart was being fabricated.

To fabricate the cart, many of the old angle iron supports had to be removed or cleaned. The removal of unnecessary angle irons was completed by using a jigsaw with a metal cutting blade. Many angle irons needed to be removed due to rust accumulation or oversized welds from previous senior design projects. Once all the undesired angle iron supports had been removed, the remaining cart components were cleaned with an angle grinder before new angle iron supports were welded on.

Before the new angle irons were attached to the cart, each piece needed to be cut to the correct dimensions and prepared for welding. This task was performed with a vertical and horizontal band saw to cut the pieces to the desired dimensions. Once each piece had been cut to dimension, an angle grinder was used to cut a chamfer into the part for the weld puddle to fall into. A MIG welder was used for all welds on the cart structure. All welds were either a butt joint or a lap joint, depending on the geometry of the parts being joined together.

After all welds were completed, the cart was spray painted with gray and orange paint. The paint will help to prevent oxidation in the future. The paint also improves the cart aesthetic appeal. Once the painting was completed, the main engine assembly was installed, followed by the supporting components (i.e., oil system and spark plug system). The final cart is shown in Figure 10.

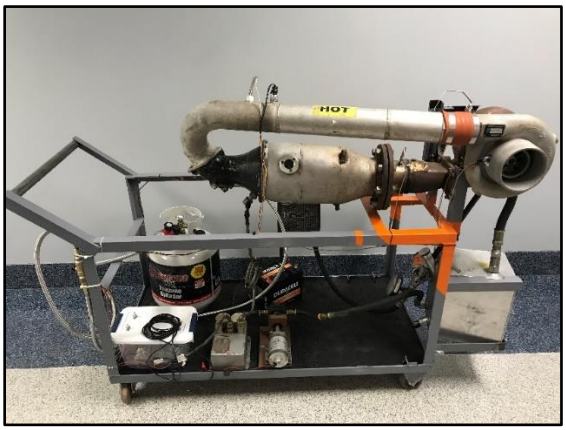


Figure 10. Final Gas Turbine Engine System.

3.3 Electronic System Final Design and Construction

The electronic system was built in three sections. First, the physical components installed on the cart will be examined, that is, the oil pump, fan, glow plug, and sensors. Next, the code controlling the Arduino will be discussed, and finally, the program Engine Control Tool will be reviewed.

3.3.1 Physical Components

The oil pump, fan, and glow plug are powered by a car battery. In the preliminary design review, it was proposed that these three components be connected to two switches with one switch controlling the oil pump and fan, and the second controlling the glow plug. SES implemented this design, and the two switches are attached to the top left of the electrical box as shown in Figure 11.

Figure 11. Switches on Electrical Box.

One difficulty that arose during this installation involved the glow plug. Because the glow plug is an older component, there was deterioration in its wiring, causing electrical arcing outside of the combustion chamber. To solve this problem, SES soldered a new wire to the glow plug, and used electrical tape to seal the joint between the glow plug and the combustion chamber. This fix is illustrated by Figure 12.

Figure 12. Glow Plug Fix.

There were five sensors installed on the engine, and two sensors installed on the electrical box. First, the infrared shaft/speed sensor was installed in such a location where it can read the RPMs of the turbine shaft. This sensor was secured in place with Flex Seal Epoxy. This sensor is shown below in Figure 13, from both the exterior and interior of the compressor.

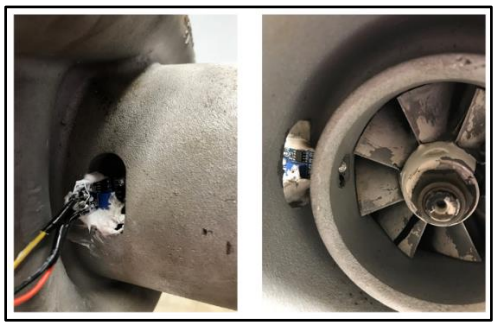


Figure 13. Infrared Shaft Speed Sensor.

The compressed air pressure transducer and compressed air thermocouple were installed by welding a nut to the engine so that the sensors could be secured in place. The combustion chamber thermocouple was installed in the same manner as the compressed air thermocouple. Finally, a custom bracket was built to hold the exhaust thermocouple in the correct location. These sensors are all displayed in Figure 14.

Figure 14. Exhaust Thermocouple, Chamber Thermocouple, Compressed Air Sensors.

The wires connecting these five sensors to the Arduino were routed along the edges of the cart and held in place with zip ties. Each thermocouple connects to a MAX31855 thermocouple amplifier breakout board set in the breadboard in the electrical box. The wiring and MAX31855 amplifiers are illustrated below in Figure 15.

Figure 15. Zip Ties Routing Wires and MAX31855 Amplifiers.

Two sensors were installed on the electrical box to obtain the necessary ambient air measurements. The AHT20 temperature/humidity sensor is installed on the side of the electrical box, and a pressure transducer is installed on the top of the electrical box. These are shown below in Figure 16.

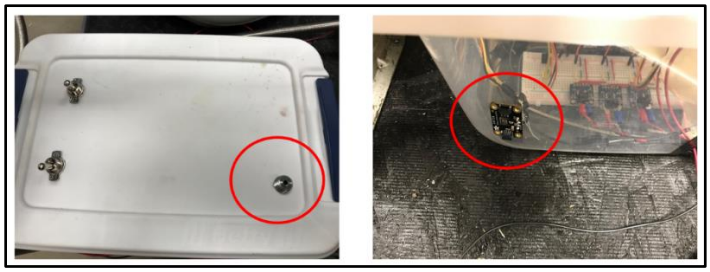


Figure 16. Ambient Pressure Transducer and AHT20 Temp/Humidity Sensor.

The Arduino connections for each sensor are drawn in Appendix D: Arduino Circuit Diagram. The connected system in the electrical box is shown in Figure 17.

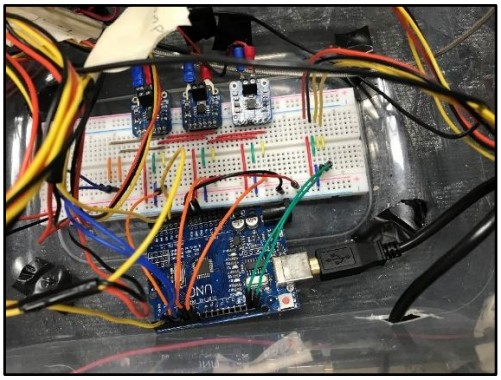


Figure 17. Connected Arduino Circuit with Sensors.

3.3.2 Arduino Code

There were two programming components to the electronic system. The first to be examined is a program uploaded to the Arduino Uno. This program is responsible for reading measurements from the sensors.

The Arduino program was written using standard Arduino code; this is a version of the programming language C++. The Arduino program has three distinct sections: the initialization of the sensors, program setup, and a loop section.

First, each sensor requires a different initialization process. Each thermocouple is defined using an installed Arduino library written for use with the Adafruit MAX31855 thermocouple amplifier breakout board. The pressure sensors are assigned analog input pins to be used. In this case, the pins A1 and A2 were chosen as the input pins to read the pressure transducers. The Adafruit AHT20 temperature/humidity sensor is defined using another Adafruit Arduino library. Finally, the variables and interrupt used for the infrared shaft speed sensor are defined.

The second section of the Arduino program is responsible for starting up the program. This is where each sensor is activated, and the baud rate is set for the serial communication that occurs between the Arduino and Engine Control Tool.

The final section of the Arduino program is the section that runs on a loop, taking one measurement from the sensors every second. This section takes a reading from each sensor, combines these values into one string with a space between each measurement, and sends this string to the serial port. This is the string that Engine Control Tool receives. A snippet of the Arduino source code is included below in Figure 18. The entirety of the Arduino source code may be viewed in Appendix F. Arduino Program Source Code.

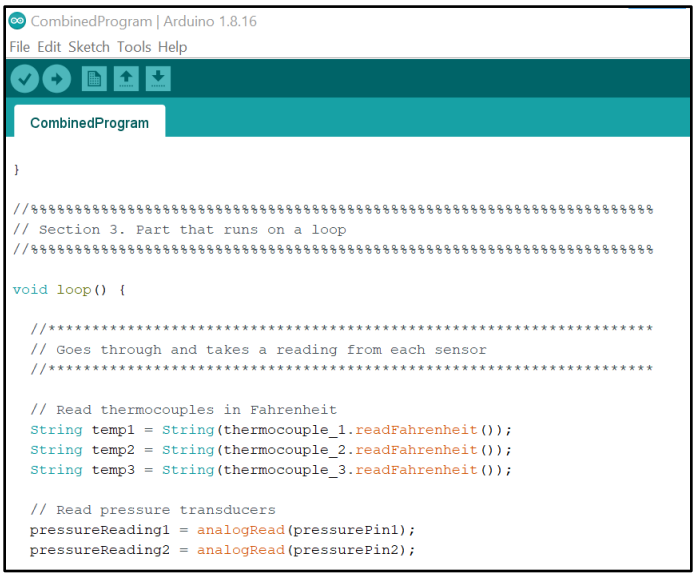


Figure 18. Arduino Source Code Snippet.

3.3.3 Engine Control Tool

The second programming component for the electronic system is a program named Engine Control Tool. Engine Control Tool is a .NET Framework Windows Presentation Foundation (WPF) application, built using Microsoft Visual Studio. WPF is a system that allows for the creation of a user interface for desktop client applications (Lee, 2022). A WPF application is written in two programming languages: Extensible Application Markup Language (XAML) and C#. The XAML language is used to create the physical components of the GUI. The C# code controls the functionality of the application.

Engine Control Tool is organized into four main files. The first, *MainWindow.xaml* defines the visual elements of the GUI. Then, there are three C# classes each written in a separate file. These classes are *MainWindow*, *CreateExcel*, and *ViewModel*. These four sections will be discussed in detail below.

The file *MainWindow.xaml* is written in XAML, and it defines the visual elements of Engine Control Tool. The GUI is a single window with two main sections. There is a ribbon at the top that contains three buttons. The first button allows the user to convert the recorded data to an Excel spreadsheet. The next two buttons simply give the user options to change between dark and light mode and the option to pick an accent color. The second section is the main body of the page. This area is divided into six sections. The sections are as follows: temperature gauges, pressure gauges, a humidity gauge, a shaft speed gauge, a flow rate input section, and the log section. Each gauge has a corresponding label and real time value displayed.

The class *MainWindow.xaml.cs* contains the C# code controlling the overall functioning of Engine Control Tool. When the program is run, this file creates instances of the other two classes, *ViewModel.cs* and *CreateExcel.cs*. It then contains methods that are called whenever the user clicks a button on the GUI. These include the color selection buttons, the Excel generation button, and the log button. This class also opens the correct serial port and begins reading data from the Arduino as soon as the program is started. Finally, it starts a separate thread that continuously monitors the temperature and pressure measurements. If either of these reaches a dangerous level, it flashes a warning label to the user and instructs them to terminate the flow of propane.

The *CreateExcel.cs* class is responsible for generating both Excel spreadsheets, one in imperial units and one in metric units. This class contains three methods besides the constructor. All three of these methods are called from *MainWindow.xaml.cs* when the user clicks on the Excel button. The first method starts Excel and sets up the workbook and two worksheets. The second method populates the first spreadsheet with the recorded data. The third method performs the same function as the second, but the values are converted to metric units.

The final class, *ViewModel.cs*, is a model class. Its only function is to store data so that data may be passed between the XAML view and other classes. This is essential, because the only thread that can interact with the GUI elements is the main thread. However, the serial data being read from the Arduino is being received in a separate thread. This thread cannot access the GUI elements, so it instead passes its received data to the view model. The GUI elements are bound to

the view model, so they can see data stored in the view model. This is how the gauges and labels in the GUI are updated to match the current sensor readings. The view model also contains lists that are constantly receiving new data from the Arduino These lists can be accessed by the *CreateExcel.cs* class to be used to populate the Excel spreadsheets.

A snippet of the Engine Control Tool source code is displayed below in Figure 19. This image is taken from the *CreateExcel* class. The source code for Engine Control Tool may be seen in Appendix G: Engine Control Tool Source Code.

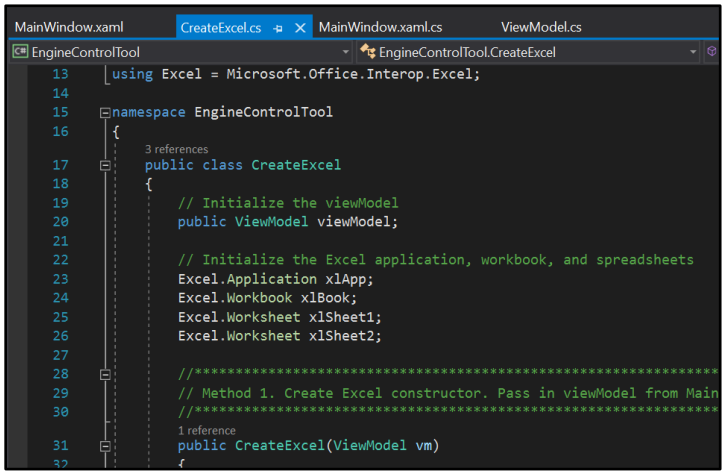


Figure 19. Engine Control Tool Source Code Snippet.

3.4 Visual Rendering of Final Design

The final design consists of the finished cart, Engine Control Tool, and the Excel output data. These are shown in Figure 20.

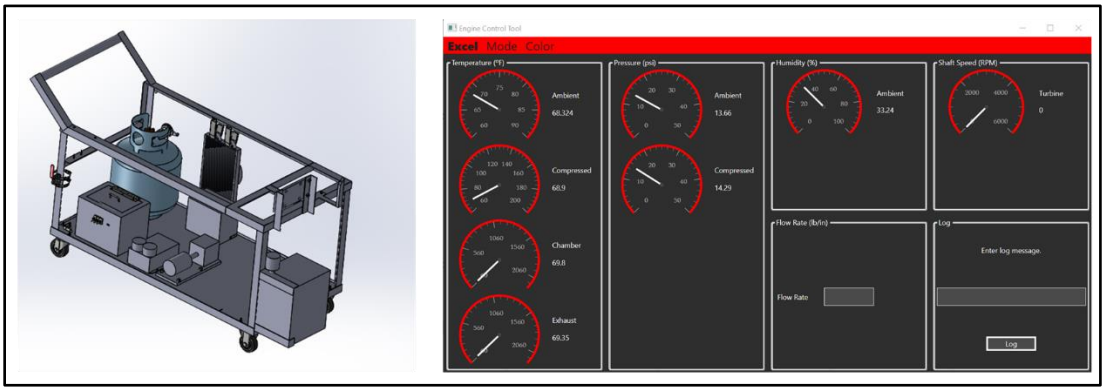


Figure 20. Final Cart Design and GUI Design.

4. Results and Discussion

The following sections cover the results of tests performed on the gas turbine engine as well as the changes in the budget over the course of this project.

4.1 Test Results

There were a variety of tests performed on the engine to ensure that it was operating properly. Five tests were performed on the mechanical system, or the physical components of the cart and engine. Five tests were completed for the electronic system. Two tests were performed on the entirety of the gas turbine engine system. The results from these twelve tests are discussed below.

4.1.1 Mechanical System Test Results

A total of five mechanical tests were performed to confirm that SES delivers a safe and high-quality product. These tests include the following: propane shutoff valve performance, cart mobility, pressure leaks, propane mass flow rate, and weld quality.

4.1.1.1 Propane Shutoff Valve Performance

The propane shutoff valve plays a critical role in overall user safety. If the Engine Control Tool prompts the user to shut off the fuel or if the user identifies a different safety concern, the shutoff valve must be easily accessible and easily actuated to quickly render the cart safe. A 0.75-inch diameter brass ball valve was selected for its large handle and quick actuation. A ninety degree turn of the valve handle completely closes the valve. To ensure that the valve is accessible, it is mounted on the vertical leg of the cart nearest the electrical box. The instruction manual provided by SES suggests that a user is stationed near the electrical box to monitor live sensor readings and operate the shutoff valve. The installed propane shutoff valve may be viewed in Figure 21.

Figure 21. Propane Shutoff Valve.

This test calls for the operators to turn on the oil pump and heat exchanger, spool up the turbocharger assembly using a leaf blower, turn on the spark generator, open the shutoff valve, allow the engine to run for thirty seconds, and then close the valve. The operators should observe the flame through the window in the combustion chamber and ensure that the flame is out immediately following closing the valve. The propane shutoff valve performance test was deemed a success following two iterations of the test regimen.

4.1.1.2 Cart Mobility

The cart mobility test is to ensure that the users can move the engine from a storage area to a testing area. Previously, the casters on the cart were dry-rotted, and moving the cart through doorframes was difficult. Now, new casters are in place, and the overall weight of the cart components is lower than the previous design.

This test calls for five independent users to push the cart through the door frame in Science and Engineering Building room 128. These users rate the cart's performance via a Google form. The collected data is shown below in Figure 22.

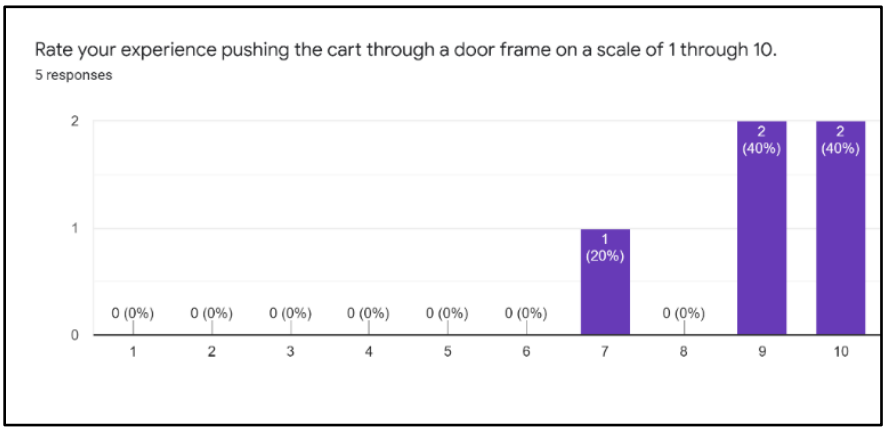


Figure 22. Mobility User Test Results.

The results from this test were averaged together for a final performance value of 9 out of 10. A final value of 9 is required for the test to be successful; hence, this test was a success.

4.1.1.3 Pressure Leaks

The pressure leak test confirms that there are no major sources of pressure losses in the system. Dish soap was mixed with water and sprayed onto the connection points between the piping, combustion chamber, and turbocharger assembly, along with the sensor interfaces with the engine components. A leaf blower is used to pressurize the system. During the pressurization, all interface locations were observed for the presence of bubbles. Bubbles indicate a loss in pressure. After the test, no bubbles were observed; therefore, this test was a success.

4.1.1.4 Propane Mass Flow Rate

The propane mass flow rate test defines the mass flow rate variable that will be used in the future to calculate theoretical work output values and efficiency. A large digital scale is used to measure the mass of the propane tank after one minute of the engine running with the regulator set to 2 psi. This is the minimum amount of fuel intake possible for the propane tank which is why it is used for this experiment. Propane has a mass flow rate of 0.002 Kilograms per second when operating at 2 psi.

4.1.1.5 Weld Quality

The weld quality test determined the proper settings that yield the best welds. Various voltages and wire feed rates were tested using a MIG welding machine. One eighth inch steel angle iron was used to ensure that the same settings could be used on components for the cart. Two welds were used to determine quality: a vertical butt weld and a horizontal butt weld. After each weld, the work piece was checked for proper penetration and porosity. The final settings for major voltage were 18-22V and fine voltage setting of 4. The wire feed rate was 150 inches per minute.

4.1.2 Electronic System Test Results

There were five tests performed on components of the electronic system: maximum pressure/temperature safety feature, switches, Arduino software recording measurements, sensor tests, and GUI acceptance.

4.1.2.1 Maximum Pressure/Temperature Safety Feature

To confirm that the maximum pressure/temperature safety feature was functional, the pressure and temperature limits were set to levels lower than atmospheric values. The Engine Control Tool was then connected to the Arduino, and it was confirmed that the warning label flashes on the Engine Control Tool screen. The warning specifies whether a dangerous temperature or pressure value was detected. Each of these warnings is shown in Figure 23.

Figure 23. Engine Control Tool warning labels.

4.1.2.2 Switches

The switches test was a simple performance check to verify that the switches control the oil pump, fan, and glow plug as intended. Each switch was first installed on the electrical box. Then, each switch was operated independently, and SES confirmed that the correct components were powered by each switch.

4.1.2.3 Arduino Software Recording Measurements

To confirm that Engine Control Tool was recording measurements correctly, the program was run in debugging mode. SES then stepped through the source code, that is, ran the program one line of code at a time. By doing this, SES confirmed that the incoming measurements were stored correctly in the designated list variables, and the Excel spreadsheet was being populated with the correct data. Figure 24 shows the process of stepping through the Engine Control Tool source code.

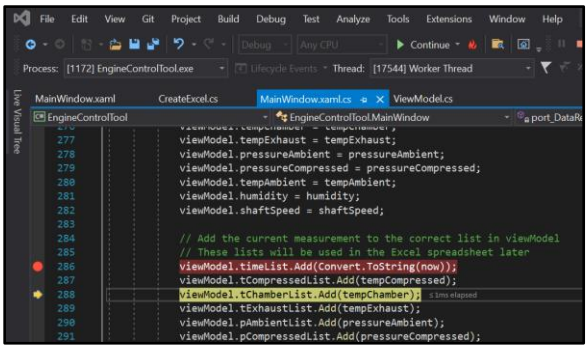


Figure 24. Stepping through Engine Control Tool source code.

4.1.2.4 Sensors

Each sensor was tested and calibrated to see if it was not reading correctly. For the pressure transducers, an ambient reading was taken, and compared to known ambient air pressure data provided by The Weather Channel (*Macon, GA weather forecast and conditions - the weather channel* 2022). The same method was used to test the humidity sensor. The measurements from the thermocouples were compared to an analog thermometer, and the infrared shaft speed sensor was assessed with a handheld tachometer. The results from these tests are shown in Table 3 below. The percentage errors from these measurements were deemed acceptable for the sensors.

Table 3. Sensor test results.

Sensor	Measured Value	True Value	Percent Error
Compressed air thermocouple	68.9°F	69 °F	0.145 %
Combustion chamber thermocouple	69.8°F	69°F	1.159 %
Exhaust thermocouple	69.064°F	69°F	0.093 %
Ambient air pressure transducer	13.6982 psi	14.75 psi	7.131 %
Compressed air pressure transducer	14.29 psi	14.75 psi	3.119 %
Ambient temperature AHT20	68.3175°F	69°F	0.989 %
Ambient humidity AHT20	25.86389 %	30 %	13.787 %
Infrared shaft speed sensor	487.5 RPM	470 RPM	3.723 %

4.1.2.5 GUI Acceptance

For the GUI acceptance test, the Engine Control Tool was examined by members of SES and our client, Dr. Sumner. Changes to Engine Control Tool were proposed and implemented, and this procedure was repeated until all parties deemed the user interface acceptable. It was during this process that the log feature was added to Engine Control Tool. The final Engine Control Tool GUI is illustrated below in Figure 25.

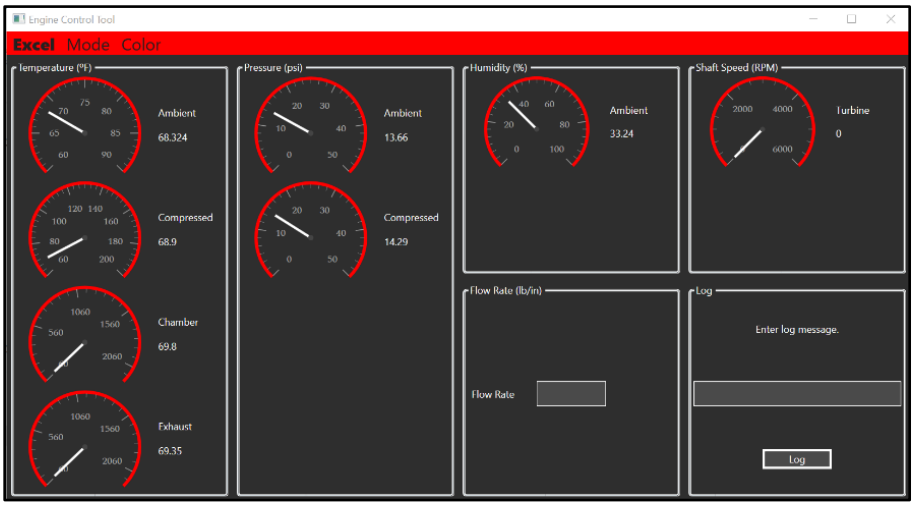


Figure 25. Engine Control Tool GUI.

4.1.3 System Performance Results

After the cart, engine, and electronic system were all fully constructed, a series of tests were performed to check the overall performance of the system. First, the cart was tested using a single leaf blower to spool up the turbocharger. The results from this test are shown in Figure 26, Figure 27, and Table 4.

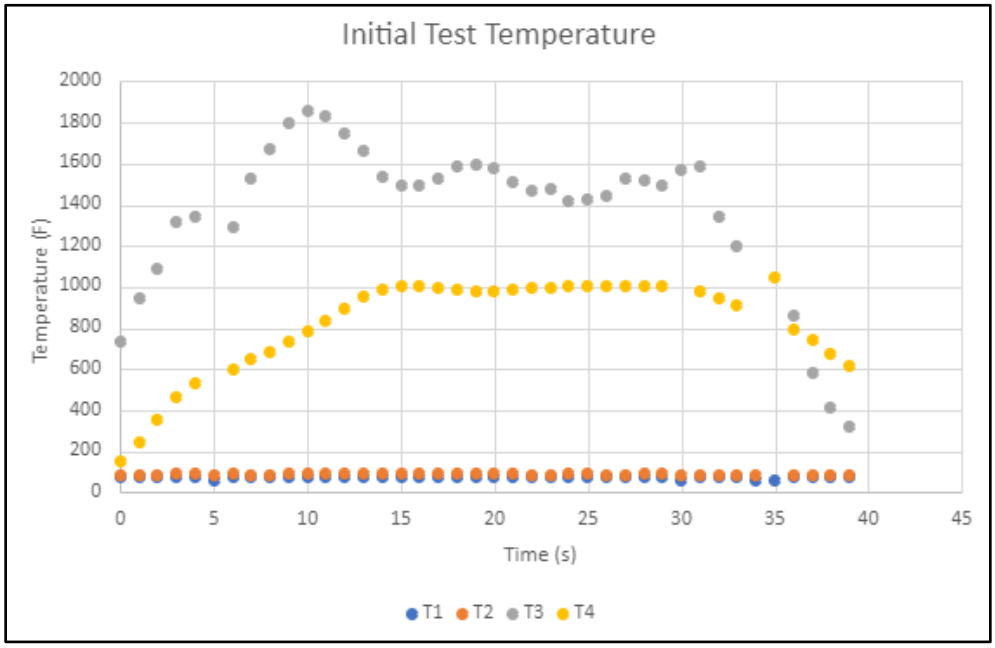


Figure 26. Temperature values during initial test.

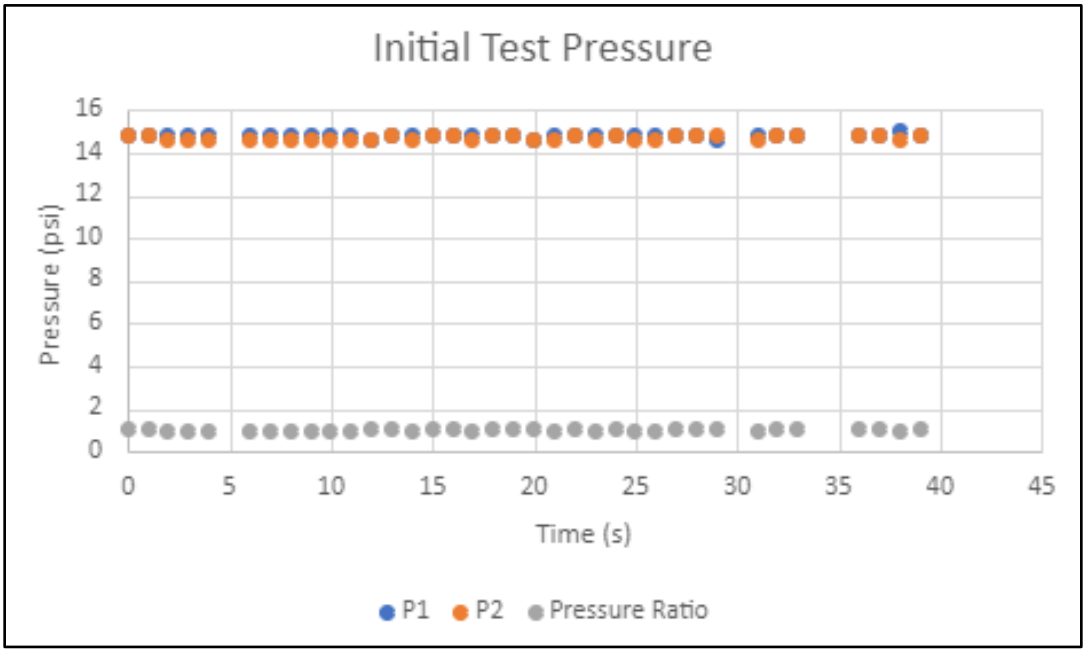


Figure 27. Pressure values during initial test.

Table 4. Results from initial test.

Values	Initial Test						
	T1	T2	T3	T4	P1	P2	Pressure Ratio
Average	73.84	87.10	1371.43	824.08	14.75	14.64	0.99
Max	75.74	87.8	1857.65	1042.25	14.99	14.76	1.01
Min	58.568	86.45	317.3	153.5	14.56	14.55	0.97

This test provided SES with valuable insight as to the performance of the engine. The maximum temperature of 1857.65°F is approximately 150 degrees lower than our expected maximum operating temperature of 2000°F. The most concerning data coming from this test is the value of the pressure ratio between the ambient air and the compressed air. The average pressure ratio of 0.99 suggests that there is no work being done on the air by the compressor. This could be one of the driving factors behind the unstable combustion currently present during the engine's operation.

To increase the pressure ratio, SES introduced a second leaf blower into the start-up sequence. The first leaf blower was an electric axial fan, which is known to underperform in the presence of back pressure. Back pressure is acting on this fan from the compressor. The second leaf blower is a gas-powered radial fan, which should handle back pressure much better than the axial leaf blower. To increase the pressure ratio, the radial fan was blown into the intake of the axial fan, which was blown into the intake of the compressor. With this new startup procedure in place, two additional tests were conducted: one axial leaf blower and both the axial leaf blower along with the radial leaf blower. Graphs of temperature and pressure for the test with one

blower are shown in Figure 28. The corresponding graphs of the test using two leaf blowers are displayed in Figure 29. Important values from these two tests are found in Table 5, Table 6, and Table 7.

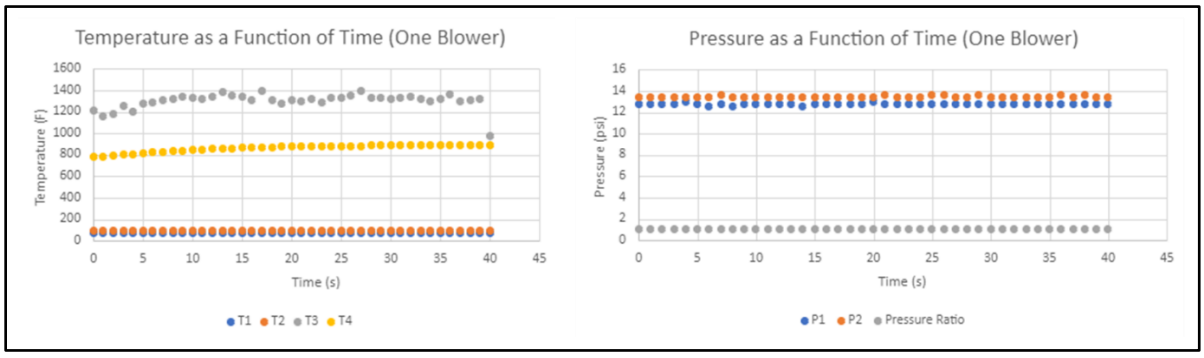


Figure 28. Temperature and pressure during axial blower test.

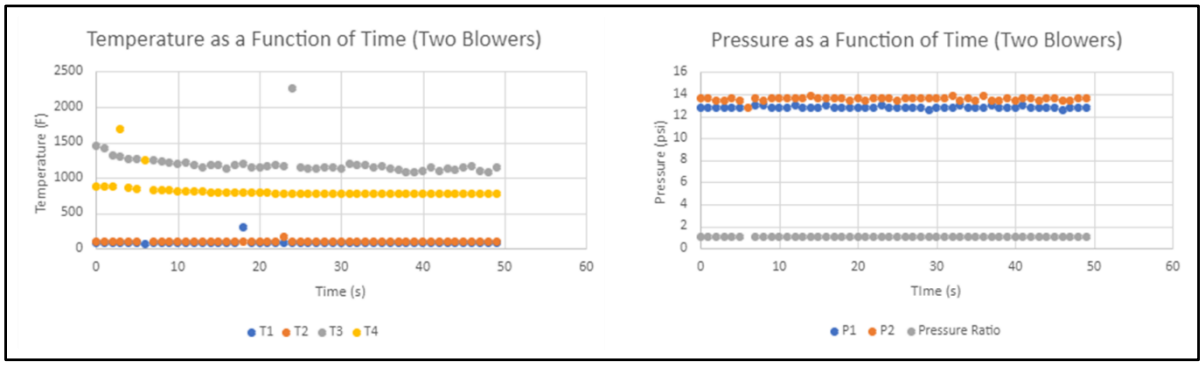


Figure 29. Temperature and pressure during dual blower test.

Table 5. Average values from secondary testing.

Number of Blowers	Average Values						
	Temperature (Fahrenheit)				Pressure (psi)		
	T1	T2	T3	T4	P1	P2	Pressure Ratio
1	76.48	94.85	1301.98	860.80	12.80	13.47	1.05
2	80.18	96.59	1196.78	819.10	12.83	13.58	1.06

Table 6. Maximum values from secondary testing.

Number of Blowers	Maximum Values						
	Temperature (Fahrenheit)				Pressure (psi)		
	T1	T2	T3	T4	P1	P2	Pressure Ratio
1	76.64	95.9	1395.95	891.05	13.02	13.66	1.06
2	302	158.9	2256.8	1695.2	13.02	13.87	1.08

Table 7. Minimum values from secondary testing.

Number of Blowers	Minimum Values						
	Temperature (Fahrenheit)				Pressure (psi)		
	T1	T2	T3	T4	P1	P2	Pressure Ratio
1	76.37	94.1	976.1	777.2	12.59	13.44	1.032258065
2	56.192	94.55	1072.4	767.3	12.59	12.81	1.032258065

As seen in the results above, the average T3 and T4 values for the single leaf blower were significantly higher than the values from the dual leaf blower test. However, the average pressure ratio from the dual leaf blower was only slightly higher than the average pressure ratio from the single leaf blower test. Further investigation will need to be done to determine what is causing this lower pressure ratio. Additional factors that could be affecting the pressure ratio include viscous friction on the turbocharger shaft from the oil in the journal bearing and an oversized combustion chamber. The friction from the oil could be restricting the shaft from spinning at the speed required for stable combustion. Furthermore, turbulence caused by an oversized combustion chamber could have adverse effects on fuel mixing and could even cause flame blowout. Both issues with the combustion chamber could be restricting the amount of energy transferred to the working fluid, which reduces the amount of energy the turbine can take from the fluid.

4.2 Other Factors

The following section outlines the other factors that Saturn Engineering Solution considered during the design and construction phase of this project.

4.2.1 Health, Safety, and Welfare

By nature, a gas turbine engine is a dangerous machine during operation. There were two main points in which SES addressed safety concerns. The first area of caution comes during startup. If an excess amount of propane builds up in the combustion chamber without the presence of a spark or air flow, an explosion can occur when the next spark is generated from the glow plug. This could cause debris to fly out of the engine, which could injure the operators. The second concern comes during normal operation. If stress cracks form on the blades of either the compressor or turbine, the blades can break off the rotor. These blades will explode radially, which can injure the operators.

To mitigate these risks, operators should not stand directly in front of or behind the compressor intake and turbine exhaust. Furthermore, operators should not stand in the radial direction of the turbocharger.

Other safety concerns include hot temperatures and shaft spinning at high revolutions per minute. With internal temperatures reaching 2000 degrees Fahrenheit, the exterior surfaces of the engine will also reach hot temperatures. Operators should not touch the main engine components after running the engine. Furthermore, the oil system should be left on for 5 minutes after running the

engine to cool the journal bearing and turbocharger shaft. Additionally, operators should be cautious of the compressor intake as clothing could be sucked into the intake.

4.2.2 Global, Cultural, and Social

The gas turbine engine in this configuration could be used to generate electrical energy. This can be done using an additional turbine stage, which can then be attached to a generator or alternator. Developing countries could use this technology as is, or in a scaled-up form factor, to generate electricity for small villages. As an example, many African countries use cell phones to both communicate and manage their finances using crypto currency. If a village does not currently have electricity and therefore cell phones, implementing a gas turbine engine for electrical power generation could provide the people with the opportunity to access crypto currencies. The use of crypto currency could increase the villagers' financial freedom (L. Sumner, personal communication, April 6, 2022).

4.2.3 Environmental

The combustion of propane yields both water and carbon dioxide. Carbon dioxide is a greenhouse gas that has negative effects on the Earth's atmosphere. However, this turbine engine could be run on many different fuels, including biodiesel. While biodiesel does not necessarily burn clean, the fuel itself can be produced in a more sustainable manner.

4.3 Costs

The overall cost of this project was less than proposed during the design phase. SES was able to acquire some materials from Mercer University to reduce costs. Angle irons, bolts, screws, and washers were all able to be obtained at no cost from Mercer University.

The original budget was $585.17 for the construction and instrumentation of the gas turbine engine. This original budget is detailed in Appendix E: Original Budget. SES was able to complete this project for $449.71, which is less than the expected value. The final budget is shown in Table 8 below. Some items that were donated by Mercer University include angle irons, screws, and washers. Some items were not purchased due to other reasons. For example, the pitot tube sensor was out of stock, so it was removed from the final budget. The heavy-duty hooks were not purchased as they were deemed unnecessary.

Table 8. Final design budget.

Part	Quantity	Price Per Part	Cost
Swivel Casters	2	$37.01	$74.02
Fixed Casters	2	$20.62	$41.24
Propane Valve	1	$14.01	$14.01
Pressure Transducer	2	$15.99	$31.98
Temperature & Humidity Sensor	1	$4.50	$4.50
K-Type Thermocouple	3	$6.09	$18.27
Short K-Type Thermocouple	1	$16.18	$16.18
Toggle Switch	2	$9.99	$19.98
Thermocouple Amplifier Breakout Board	3	$14.95	$44.85
IR Infrared Sensor Module	1	$9.58	$9.58
15-Feet USB Cable	1	$2.45	$2.45
Wood 1/2' thick	1	$50.00	$50.00
Brass On/Off Valve with Lever Handle	1	$9.01	$9.01
Propane Hose Connection	4	$10.00	$40.00
Propane Hose	1	$30.22	$30.22
Toggle Switch	2	$5.31	$10.62
USB Port	1	$1.52	$1.52
Low Temperature Sealant	1	$16.29	$16.29
22 Gauge Wire	1	$14.94	$14.94
Total			**$449.66**

5. Summary and Conclusions

The following sections provide a summary of the entirety of the Gas Turbine Engine Upgrades project as well as recommendations for projects to be performed by senior design teams in the future.

5.1 Project Summary

Saturn Engineering Solutions was asked to address three problems for our client: refurbish the cart structure, instrument the system, and provide a method to read sensor data in real time. To improve the usability of the cart, various supporting structures were removed, casters were replaced, and mounting locations were repositioned on the cart. Because the engine had been sitting in disrepair for several years, the cart had begun to corrode and develop surface rust.

To refurbish the cart, first, the entire engine and supporting components were removed, leaving the bare cart. The surface rust was removed with an angle grinder. The access supports on the cart were cut off using a jigsaw. The two vertical posts used to attach the engine to the cart were cut off, and new supports were fabricated. These new supports were welded to the front of the cart, effectively moving the entire engine towards the front. A cross bar along the rear of the cart was removed to allow for additional clearance for future senior design projects. New casters

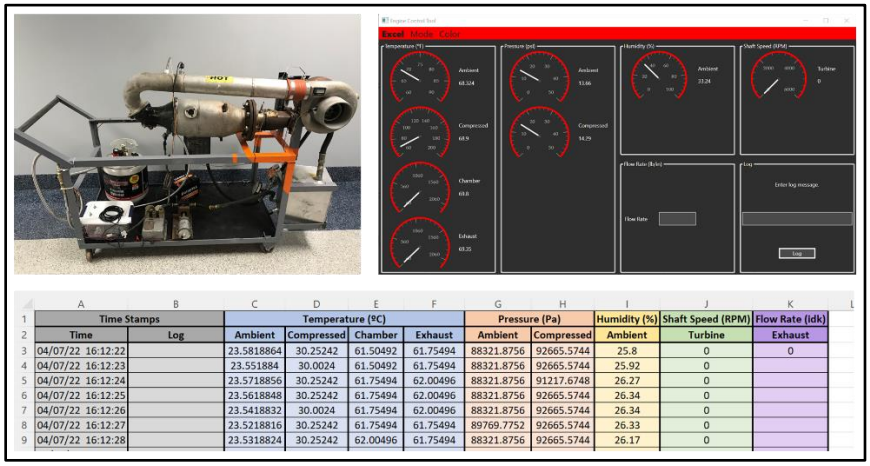
were added to improve the mobility of the cart. After welding new supports to the cart, a coat of paint was applied to the cart to improve visual appearance and resistance to corrosion.

A series of sensors were mounted to the engine to record different values throughout the system. Temperature is recorded for the following states: ambient air, compressed air, combustion chamber, and exhaust. Pressure was recorded for ambient air and compressed air. The pressure is assumed to be constant through both the combustion chamber and the turbine. Additionally, ambient humidity and turbine shaft speed are recorded.

The data from these sensors are collected using an Arduino Uno, and then displayed to a GUI, known as Engine Control Tool. This user interface displays all the sensor readings using gauges. This user interface also accepts a reading for air flow rate using a hand-held anemometer. Furthermore, the user has an option to add time stamps to the data to assist in differentiating between various tests being conducted on the engine.

Finally, the data collected during the test is exported to an Excel file. This Excel contains two spreadsheets. The first presents the data in imperial units and the second in metric units. Operators may use this data to perform a wide variety of calculations including net work and efficiency.

The final cart, Engine Control Tool, and Excel output are shown in Figure 30 below.



Time Stamps		Temperature (ºC)				Pressure (Pa)		Humidity (%)	Shaft Speed (RPM)	Flow Rate (idk)
Time	Log	Ambient	Compressed	Chamber	Exhaust	Ambient	Compressed	Ambient	Turbine	Exhaust
04/07/22 16:12:22		23.5818864	30.25242	61.50492	61.75494	88321.8756	92665.5744	25.8	0	0
04/07/22 16:12:23		23.551884	30.0024	61.50492	61.75494	88321.8756	92665.5744	25.92	0	
04/07/22 16:12:24		23.5718856	30.25242	61.75494	62.00496	88321.8756	91217.6748	26.27	0	
04/07/22 16:12:25		23.5618848	30.25242	61.75494	62.00496	88321.8756	92665.5744	26.34	0	
04/07/22 16:12:26		23.5418832	30.0024	61.75494	62.00496	88321.8756	92665.5744	26.34	0	
04/07/22 16:12:27		23.5218816	30.25242	61.75494	61.75494	89769.7752	92665.5744	26.33	0	
04/07/22 16:12:28		23.5318824	30.25242	62.00496	61.75494	88321.8756	92665.5744	26.17	0	

Figure 30. Final deliverables, cart, Engine Control Tool, Excel output.

5.2 Recommendations for Future Work

Additional turbines can be added to the system for assorted reasons, including harvesting both mechanical and electrical energy. A turbine could be added in two locations: directly following the combustion chamber, or after the turbine used to drive the compressor. Adding a turbine after the combustion chamber places the drive turbine in a scenario that more closely resembles the situation the original designers had in mind. Turbochargers normally compress air before entering an internal combustion engine, and the exhaust gasses flow into the turbine section,

which drives the compressor. Taking energy out of the working fluid before it enters the drive turbine places the drive turbine in a state that might match the original design point.

These turbines could be connected to an alternator to harvest electrical energy or to harvest mechanical energy. The electric energy generated from the engine could be used in a wide range of applications. One application could be to power phone chargers for third world countries. Furthermore, mechanical energy could be used to drive an axel for a go-kart. Many examples of gas turbine engines powering go-karts can be found on the internet. One such example is shown below in Figure 31.

Figure 31. Gas turbine engine powered go-kart (Gorgan, 2020).

More ideas for future projects involving the Mercer University gas turbine engine are outlined in Appendix H: Ideas for Future Projects.

5.3 Acknowledgements

Saturn Engineering Solutions would not have been able to complete this project without the support of our Technical Advisors and Project Manager. Specifically, Mr. John Mullis supported this project through advising on the fabrication phase as well as TIG welding various sensor interfaces. Additionally, SES would like to thank our client, Dr. Loren Sumner, for trusting our team to deliver a quality product that met his design criteria. Finally, SES would like to thank the previous senior design teams that made the gas turbine engine possible.

References

Gorgan, E. (2020, September 14). *Engineer is building jet-powered go kart in his back yard, hopes for 140 mph*. autoevolution. Retrieved April 10, 2022, from https://www.autoevolution.com/news/engineer-is-building-jet-powered-go-kart-in-his-back-yard-hopes-for-140-mph-148682.html#

Lee, T. G. (2022, January 28). *What is WPF? - visual studio (windows)*. What is WPF? - Visual Studio (Windows) | Microsoft Docs. Retrieved April 10, 2022, from https://docs.microsoft.com/en-us/visualstudio/designers/getting-started-with-wpf?view=vs-2022

Macon, GA weather forecast and conditions - the weather channel. The Weather Channel. (2022, April 10). Retrieved April 10, 2022, from https://weather.com/weather/today/l/267edf19d818ea2409743a70a191cb89a15a2b97edab07daac656ebca0d30c08

Appendix A: Test Plans

This section details the testing plans proposed during the design phase of the project.

1. Propane Shutoff

Type: Performance check
Test Objective: Ensure that the propane valve stops the flow of propane through the system.
Equipment Needed:

- Fully assembled cart
- Full propane tank
- Leaf blower

Location: South Side of Science and Engineering Building
Date/Total Time: March 7, 2022, 30 minutes
Personnel: Ryan McMillan, Barrett McDonald, and Mary Lichtenwalner
Criteria for success: User can quickly shut off the flow of propane, using the shut-off valve, during engine operation.
Procedure:

1. Spool up the compressor and turbine using the leaf blower.
2. Turn on the spark generator.
3. Ensure that the shut-off valve is open.
4. Open the propane valve on the tank.
5. Allow engine to run for approximately 30 seconds.
6. Quickly close the shut-off valve.
7. Ensure that the flame is no longer present in the combustion chamber.

2. Maximum Pressure/Temperature Safety Feature

Type: Safety test
Test Objective: To ensure that the software notifies the user when an unsafe temperature or pressure is recorded by Arduino.
Equipment Needed:

- Sensor array with Arduino
- Laptop
- Recording software

Location: Senior design lab – SEB (Science and Engineering Building) Room 128
Date/Total Time: March 12, 2022, 30 minutes
Personnel: Barrett McDonald, Ryan McMillan, and Mary Lichtenwalner
Criteria for success: The software notifies the user when a value higher than the specified maximum temperature/pressure levels is recorded.
Procedure:

1. Open the recording software in Visual Studio and connect sensor array to laptop.
2. In the code for the recording software, change the safe maximum levels to a value below atmospheric temperature and pressure.
3. Run software.

4. Confirm that the recording software notifies the user when recorded values are higher than the input maximum levels from step 2.

3. Switches

Type: Performance check
Test Objective: To ensure that all switches correctly turn the electrical components on and off.
Equipment Needed:

- Battery
- Glow plug
- Oil cooler
- Oil pump
- Oil tank
- Electrical wiring

Location: Senior design lab – SEB (Science and Engineering Building) Room 128
Date/Total Time: March 12, 2022, 5 minutes
Personnel: Barrett McDonald, Ryan McMillan, or Mary Lichtenwalner
Criteria for success: Switches successfully close circuits, allowing electrical components to operate.
Procedure:

1. Turn on the glow plug switch.
2. Ensure that the glow plug is operating.
3. Turn glow plug switch off.
4. Turn on the oil pump and oil cooler switch.
5. Ensure that the oil pump and oil cooler are operating.
6. Turn off the oil cooler and the oil pump switch.

4. Arduino Software Recording Measurements

Type: Performance test
Test Objective: To confirm that the recording software is recording measurements at a specified time interval and outputs these measurements into an Excel file after running the engine.
Equipment Needed:

- Sensor array with Arduino
- Laptop
- Recording software

Location: Senior design lab – SEB (Science and Engineering Building) Room 128
Date/Total Time: March 12, 2022, 2 hours
Personnel: Mary Lichtenwalner
Criteria for success: Recording software measures each value at the correct time and stores the recorded value in the correct variable. Correct variables are exported to Excel.
Procedure:

1. Connect sensor array to laptop and open recording software in Visual Studio.
2. Step through the code and confirm that each measurement is being recorded to the correct variable.

3. Run the program and measure the time between measurements. Confirm that this matches the time delay set in the code.
4. Let the recording software run for 5 minutes, then click the button on the GUI to convert the data into an Excel spreadsheet.
5. Confirm that the spreadsheet is generated correctly.
6. Repeat this process, modifying the code until the criteria for success is met.

5. Mobility

Type: Human acceptance check
Test Objective: Ensure that the new casters allow the cart to easily travel over obstacles (I.e., door jams and bricks).
Equipment Needed:

- Fully assembled cart

Location: Engineering Building and Science and Engineering Building
Date/Total Time: March 7, 2021, 30 minutes
Personnel: Ryan McMillan, Barrett McDonald, and Mary Lichtenwalner
Criteria for success: Cart achieves an average score of 9 or above from operator.
Procedure:

1. Operator pushes cart through door frame in Science and Engineering Building Room 128.
2. Operator rates the cart's ability to easily pass through the door frame on a scale of 1-10.
3. Operator pushes cart through the door frame on the first-floor doors on the eastern side of the engineering buildings.
4. The operator repeats step 2 for this door frame.
5. Operator pushes cart around the bricks between the Science and Engineering Building and the Engineering Building.
6. Operator repeats step 2.
7. Operator pushes the cart onto the grass on the south side of the Science and Engineering Building.
8. From the grass, the operator pushes the cart onto the sidewalk on the south side of the engineering building.
9. Operator repeats step 2.
10. Finally, the operator averages all the scores from the previously mentioned tests.

6. Pitot Tube

Type: Performance test
Test Objective: To ensure the pitot tube is measuring accurate values for static and dynamic pressure.
Equipment Needed:

- Leaf blower
- Anemometer
- Pitot Tube

- Arduino
- Computer

Location: Senior design lab – SEB (Science and Engineering Building) Room 128
Date/Total Time: January 15, 2022, 30 min
Personnel: Ryan McMillan, Barrett McDonald, and Mary Lichtenwalner
Criteria for success: The pitot tube is measuring accurate data within 5 % compared to the anemometer data.
Procedure:

1. Turn on the computer and Arduino system.
2. Turn on the leaf blower and set to a constant flow rate.
3. Place the Anemometer directly in the center of the airflow exiting the leaf blower.
4. Record flow rate reading from an anemometer.
5. Place the pitot tube directly in the center of the airflow exiting the leaf blower.
6. Record the dynamic and static pressures delivered to the computer from the Arduino.
7. Use the dynamic and static pressure values delivered by the pitot tube to calculate the mass flow rate of the air exiting the leaf blower.
8. Compare the results from the anemometer and the pitot tube to ensure accurate data and calculations.

7. Pressure/Temperature Sensors

Type: Performance test
Test Objective: To ensure that the thermocouples and pressure transducers are functioning correctly.
Equipment Needed:

- Thermometer
- Barometer
- Pressure transducers
- Thermocouples
- Arduino
- Recording software

Location: Senior design lab – SEB (Science and Engineering Building) Room 128
Date/Total Time: January 15, 2022, 1 hour
Personnel: Mary Lichtenwalner, Barrett McDonald, and Ryan McMillan
Criteria for success: The values recorded by the pressure transducers and thermocouples match that of an analog barometer and thermometer by within 2 %.
Procedure:

1. Connect the sensor array with the Arduino, pressure transducers, and thermocouples to the laptop.
2. Open the recording software.
3. Use an analog thermometer and barometer to measure the ambient pressure and temperature of the testing location.
4. Confirm that the pressure transducers and thermocouples are recording the same values as the analog devices within 2 %.

5. If the pressure transducers and thermocouples are not recording the correct values, calibrate them and repeat this test until the criteria for success is met.

8. Pressure Leaks

Type: Performance test
Test Objective: Ensure that there are no pressure leaks at all sensor locations and interfaces between components of the engine (i.e., between combustion chamber and the turbine)
Equipment Needed:

- Assembled jet engine
- Dish Soap
- Water
- Sponge
- Bucket
- Leaf Blower

Location: Senior design lab – SEB (Science and Engineering Building) Room 128
Date/Total Time: March 7, 2022, 30 minutes
Personnel: Ryan McMillan, Barrett McDonald, and Mary Lichtenwalner
Criteria for success: No noticeable bubbles form at all component interfaces
Procedure:

1. Fill a 5-gallon bucket with water until the bucket is half full.
2. Put dish soap into the water bucket.
3. Agitate the solution so that there are a few soap bubbles at the surface.
4. Using the sponge, apply the solution to the component interfaces on the engine (including the sensor interfaces).
5. One operator uses the leaf blower to spool up the compressor and turbine.
6. While the leaf blower is in use, the other two operators will observe the interfaces for the formation of bubbles in the applied solution.

9. Propane Mass Flow Rate

Type: Performance test
Test Objective: Determine the mass flow rate for different pressure variation on the opening valve.
Equipment Needed:

- Scale
- Propane tank
- Gas lines
- Gas turbine engine
- Stopwatch

Location: Outside south side of Science and Engineering building at Mercer University, Macon Ga
Date/Total Time: March 7, 2022, 30 minutes
Personnel: Ryan McMillan, Barrett McDonald, and Mary Lichtenwalner
Criteria for success: Successfully obtaining mass flow rate values for varying valve pressures of the propane tank

Procedure:

1. Measure the initial mass of the propane tank by placing it on the scale and recording the mass.
2. Initiate the gas turbine engine.
3. Open the propane valve to 5 psi and start the stopwatch.
4. Run the gas turbine engine at constant propane valve pressure for 5 minutes.
5. Close the propane valve to 0 psi and stop the stopwatch.
6. Turn off the gas turbine engine.
7. Measure the final weight of the propane tank by placing it on the scale and recording the mass.
8. Repeat steps 2-7 at 10, 15, 20, and 25 psi propane valve pressures and record the data.
9. Determine a correlation between the propane valve pressure and the change in mass over the five minutes the engine was running.

10. Weld Test

Type: Quality assurance check
Test Objective: Ensure that both the voltage and wire feed parameters are properly set for safe and strong welds.
Equipment Needed:

- MIG Welder
- Welding Jackets
- Welding Helmets
- Closed-toe shoes
- Spare 1.5 x 1.5-inch angle iron pieces approximately 4 inches long each
- Hydraulic press
- Grinding wheel

Location: Mercer University, Weld Shop
Date/Total Time: January 10, 2022, 1 hour
Personnel: Ryan McMillan, Barrett McDonald, and Mary Lichtenwalner
Criteria for success: Welded sample does not break at welding location during bend test
Procedure:

1. Turn on the welding exhaust vent.
2. Turn on the MIG welder.
3. Open the welding gas valve.
4. Using the grinding wheel, clean the surface of the angle iron.
5. Set the MIG welder to 18 Volts and the feed rate to 240.
6. But weld the two pieces of angle iron together.
7. Using the hydraulic press, bend the welded sample.
8. If the test fails, alter the settings in step 5 and repeat the procedure starting with step 6.

11. Cart Aesthetics

Type: Human acceptance test

Test Objective: To ensure that the cart is aesthetically pleasing for operators and spectators.
Equipment Needed:

- Five impartial participants
- Cart

Location: Senior design lab – SEB 128
Date/Total Time: March 12, 2022, 10 min
Personnel: Ryan McMillan, Barrett McDonald, Mary Lichtenwalner, and five impartial participants
Criteria for success: An average score of 7 or more out of 10 on aesthetics
Procedure:

1. Gather five impartial participants.
2. Have them view the gas turbine engine cart.
3. Have them rank how aesthetically pleasing the cart is on a 1-10 scale.
4. Collect the data from the contestants and average the scores.

12. GUI Interface

Type: Human acceptance test
Test Objective: To confirm that the graphical user interface for the recording software meets the physical, mental, and emotional requirements of the user.
Equipment Needed:

- Recording software
- Sensor array with Arduino
- Laptop

Location: Senior design lab – SEB (Science and Engineering Building) Room 128
Date/Total Time: March 12, 2022, 1 hour
Personnel: Mary Lichtenwalner, Ryan McMillan, Barrett McDonald, and Dr. Sumner
Criteria for success: Our client, Dr. Sumner, conveys that he is satisfied with the graphical user interface.
Procedure:

1. Connect the sensor array to the laptop.
2. Open the recording software.
3. Let Barrett McDonald, Ryan McMillan, and Dr. Sumner use the graphical user interface.
4. Collect feedback from Barrett McDonald, Ryan McMillan, and Dr. Sumner, and make changes accordingly.
5. Repeat the process until the GUI is deemed satisfactory.

Appendix B: Revised Supporting Calculations

The original proposed design had six vertical supports on the cart. The final design has four vertical supports. Therefore, SES ran more analysis using ANSYS to ensure that the four vertical supports would sufficiently support the gas turbine engine. The ANSYS calculations done by SES that demonstrate the ability of only having four vertical supports on the cart are demonstrated below. This change lowered manufacturing costs as well as opened the cart to be modified internally much easier. These results are displayed in Figure 32 and Figure 33.

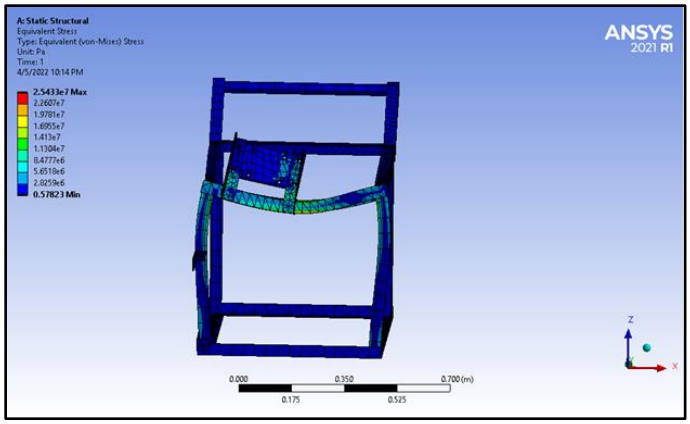


Figure 32. Equivalent Von Mises stress with only four vertical supports.

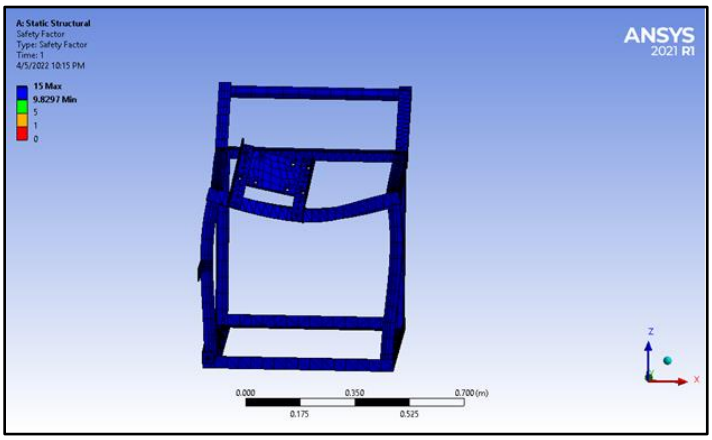


Figure 33. Factor of safety with only four vertical supports.

Appendix C: Updated/Final Working Drawings

This section contains the models of the final cart design.

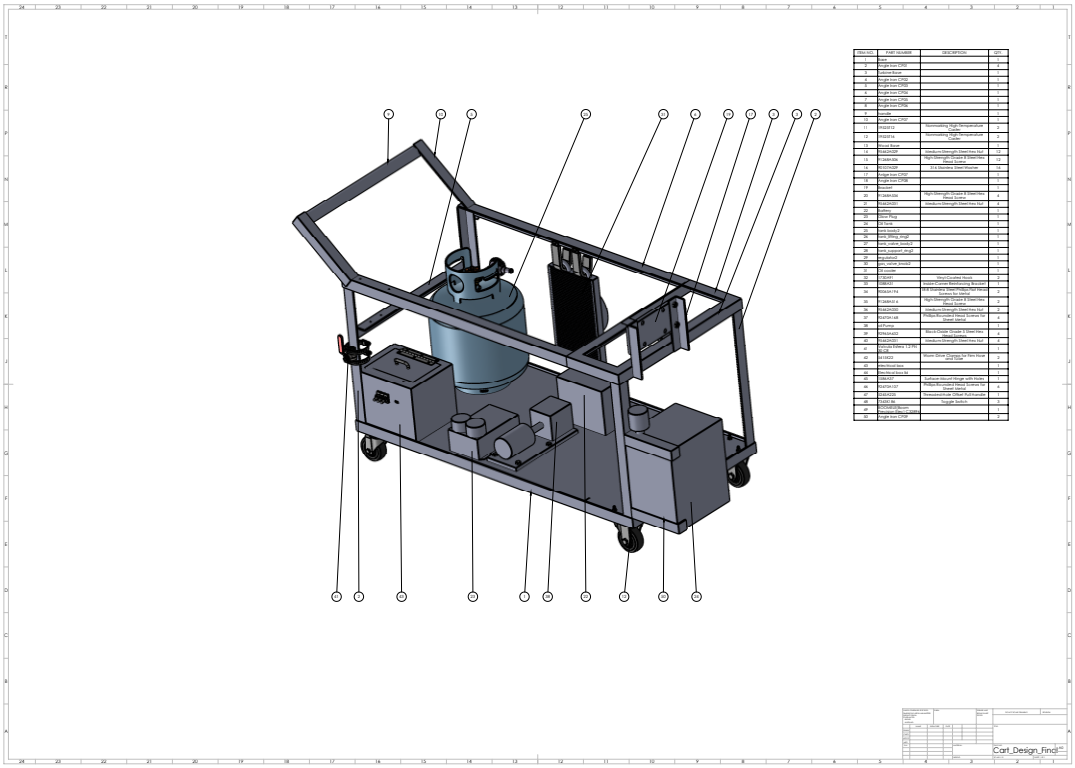


Figure 34. Final cart design.

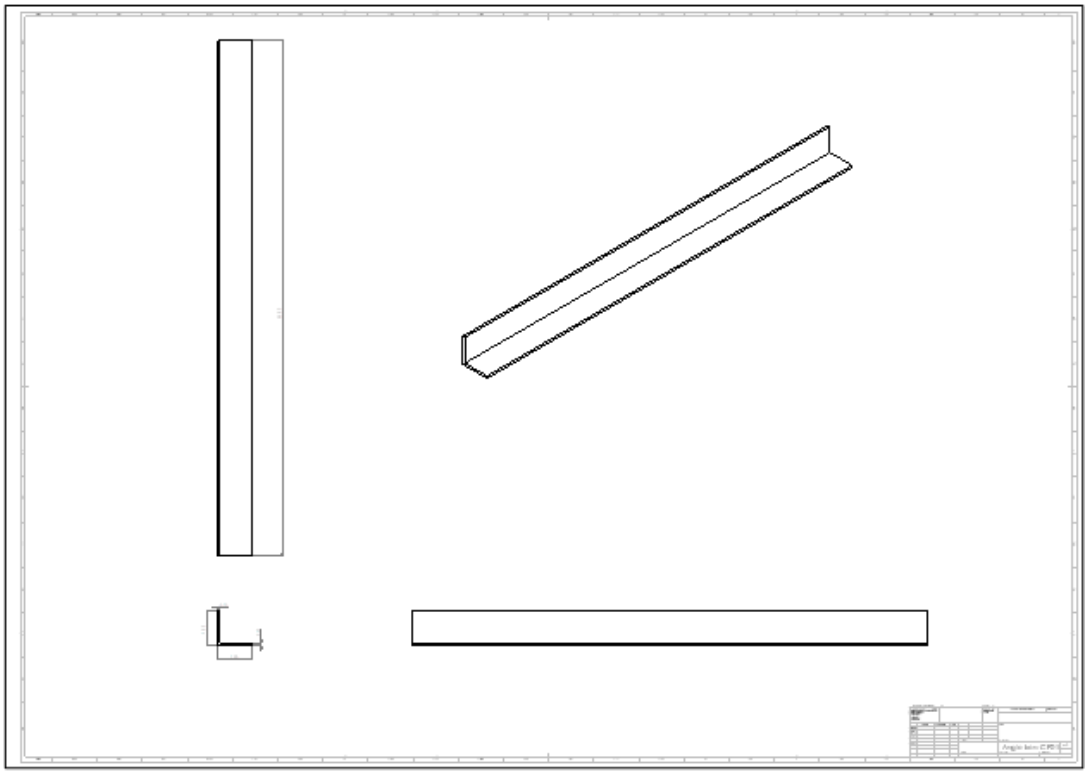


Figure 35. Angle iron CP01.

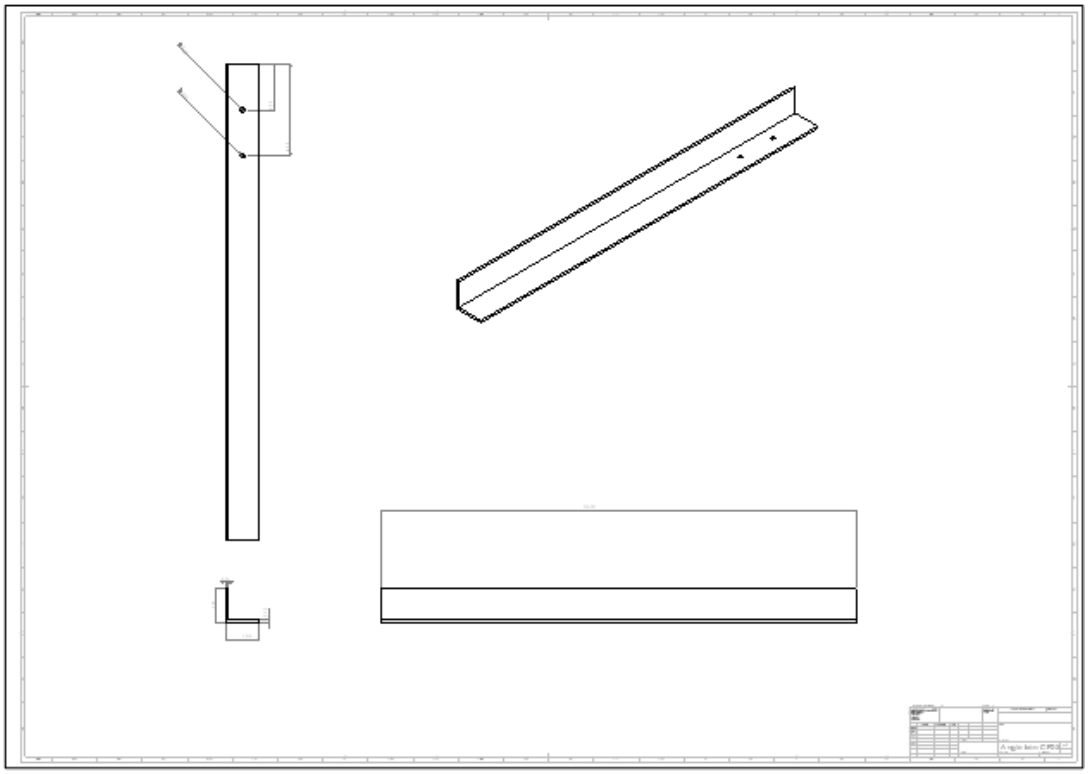


Figure 36. Angle iron CP03.

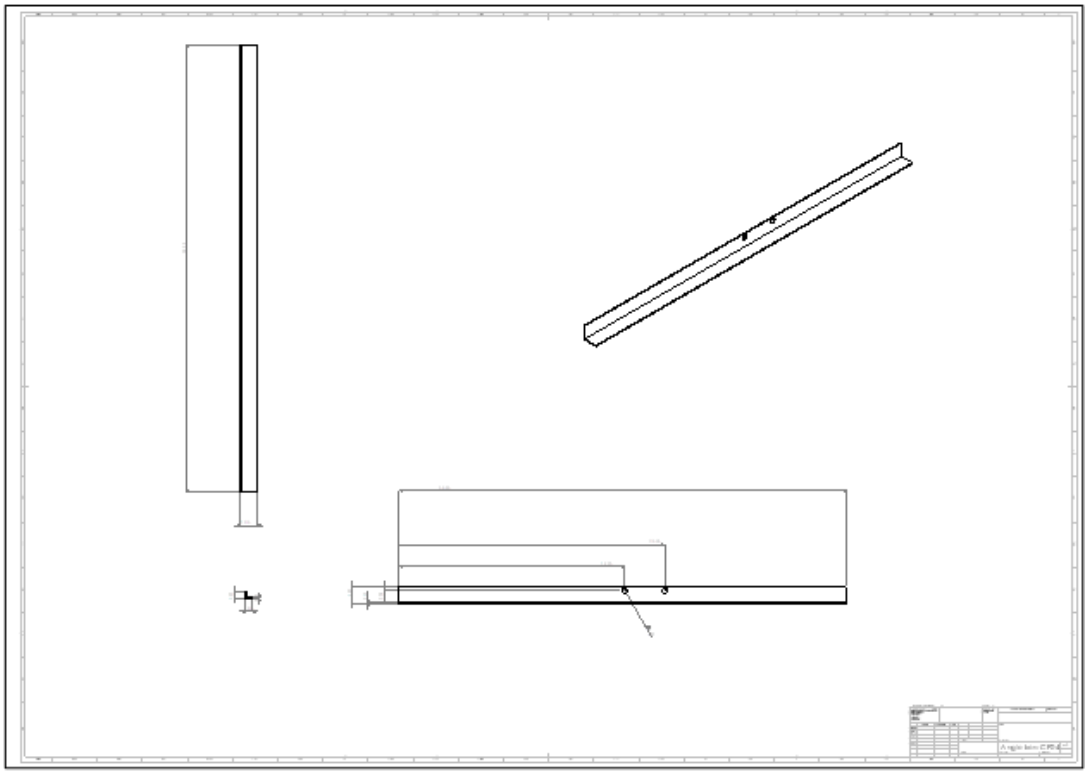


Figure 37. Angle iron CP04.

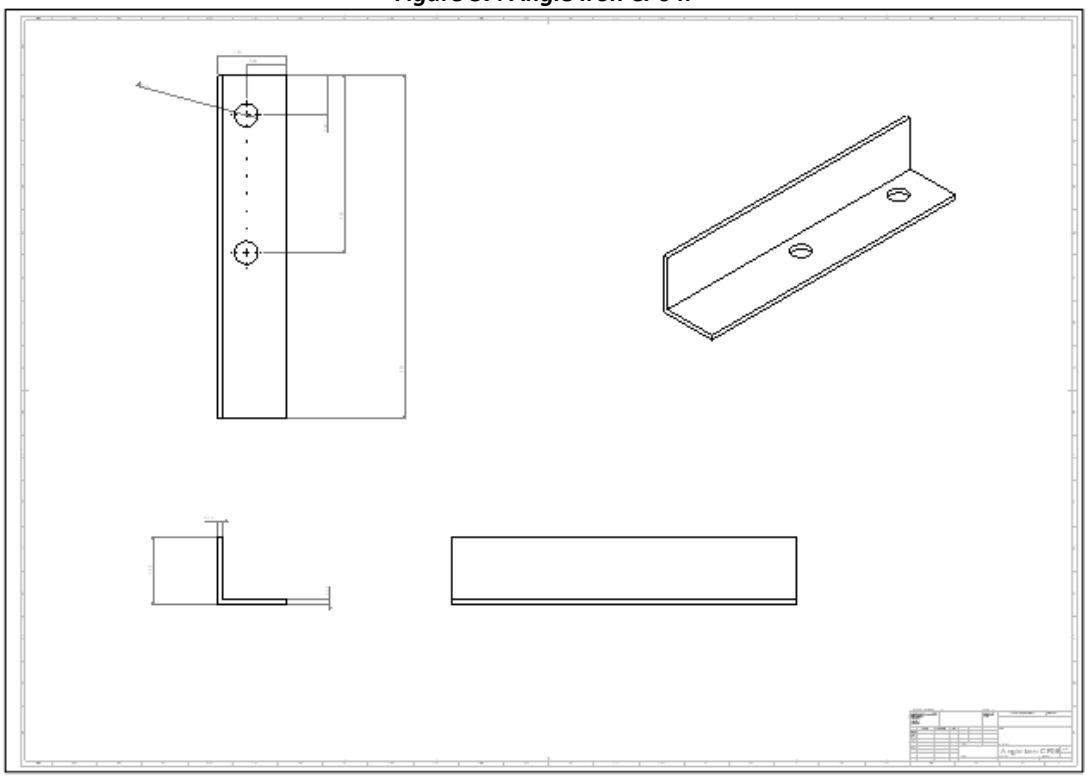


Figure 38. Angle iron CP08.

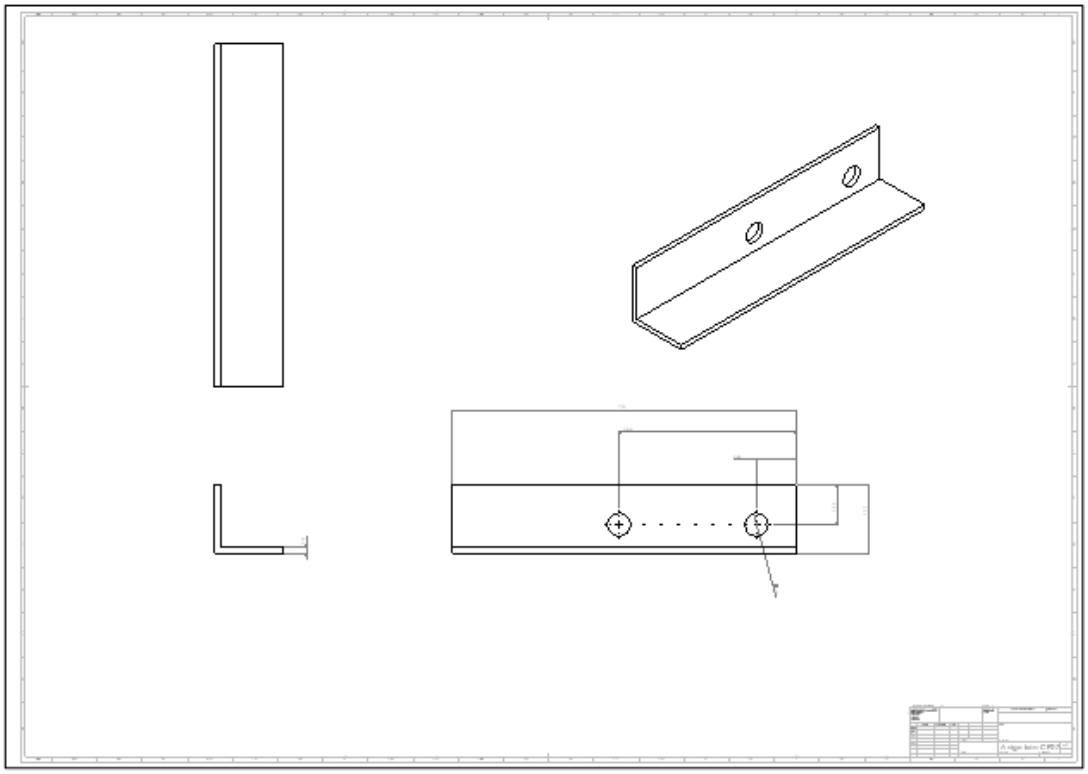


Figure 39. Angle iron CP07.

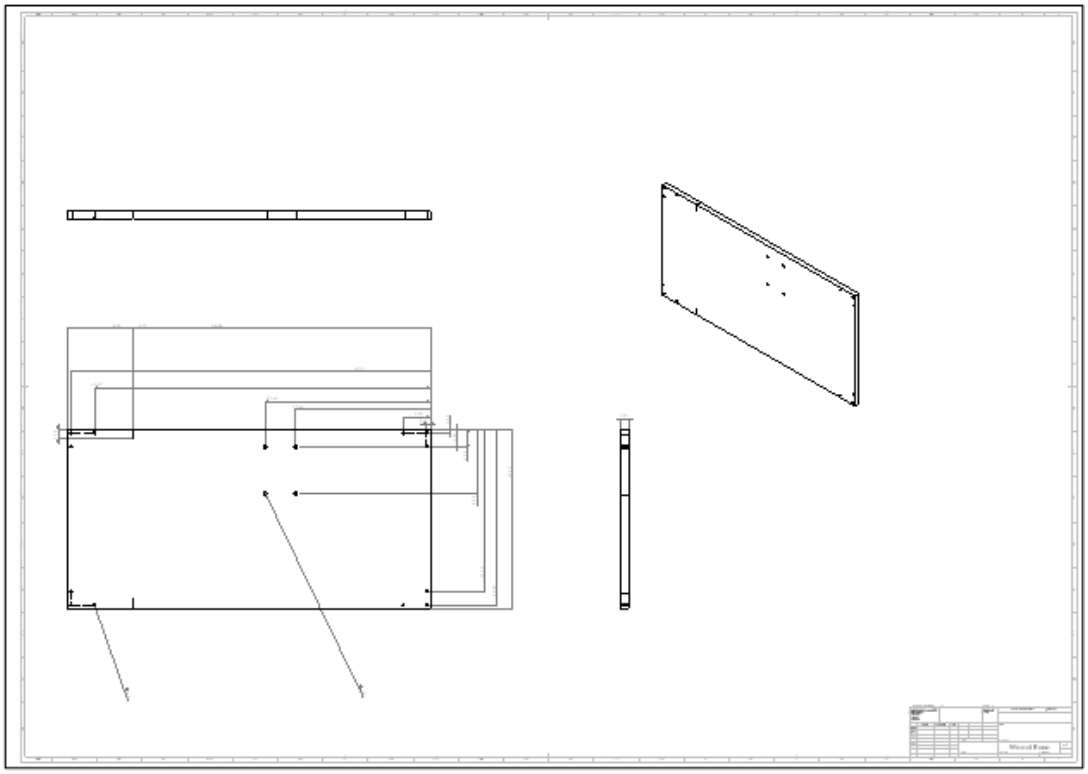


Figure 40. Wood base.

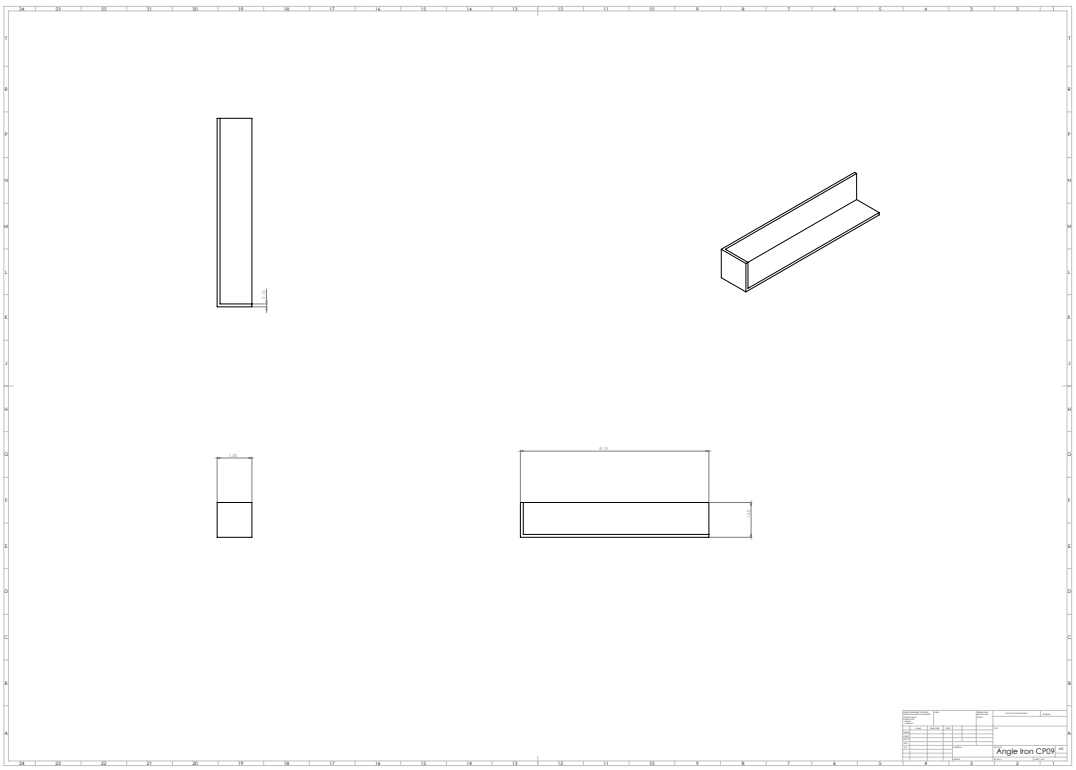


Figure 41. Angle iron CP09.

Appendix D: Arduino Circuit Diagram

Figure 42, shown below, shows the Arduino circuit connections. The wires are color coded to match the wires installed in the electrical box on the gas turbine engine system.

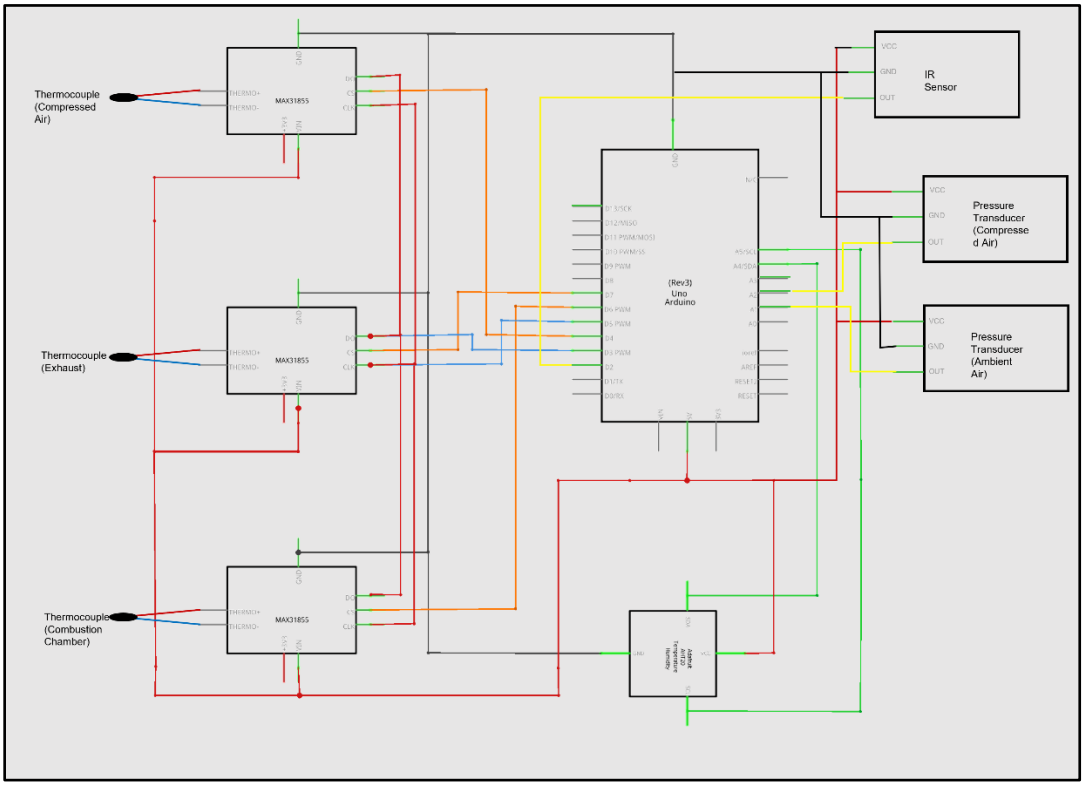


Figure 42. Arduino circuit diagram.

Appendix E: Original Budget

Table 9 shows the original budget proposed during the design phase of this project. SES was able to complete the project at a lower cost than initially anticipated.

Table 9. Original budget.

Part	Quantity	Price Per Part	Cost
Swivel Casters	2	$37.01	$74.02
Fixed Casters	2	$20.62	$41.24
Propane Valve	1	$14.01	$14.01
0.125" Angle Iron - 6ft	2	$16.95	$33.90
Pressure Transducer	2	$15.99	$31.98
Temperature & Humidity Sensor	1	$4.50	$4.50
K-Type Thermocouple	3	$6.09	$18.27
Toggle Switch	2	$9.99	$19.98
Thermocouple Amplifier Breakout Board	3	$14.95	$44.85
IR Infrared Sensor Module	1	$9.58	$9.58
15-Feet USB Cable	1	$2.45	$2.45
Wood 1/2' thick	1	$20.00	$20.00
Medium-Strength Steel Hex Nut 1/4"-20 Thread Size	1	$5.56	$5.56
High-Strength Grade 8 Steel Hex Head Screw 1/4"-20	2	$8.61	$17.22
316 Stainless Steel Washer	1	$7.11	$7.11
Slotted Rounded Head Screws 18-8 Stainless Steel	1	$7.35	$7.35
Inside-Corner Reinforcing Bracket with 2" Long Sides	10	$4.70	$47.00
18-8 Stainless Steel Phillips Flat Head Screws	1	$5.91	$5.91
Surface-Mount Hinge with Holes 6" x 3/4"	1	$5.59	$5.59
Threaded-Hole Offset Pull Handle	1	$5.70	$5.70
Phillips Rounded Head Screws, 18-8 Stainless Steel	1	$3.55	$3.55
Worm-Drive Clamps for Firm Hose and Tub	1	$11.61	$11.61
Brass On/Off Valve with Lever Handle	1	$9.01	$9.01
Toggle Switch	2	$5.31	$10.62
USB Port	1	$1.52	$1.52
Heavy Duty Hooks	2	$3.51	$7.02
Pitot Tube Sensor	1	$38.00	$38.00
Deacon 4011 High Temperature Sealant	1	$75.70	$75.70
Easy-to-Form Stainless Steel Wire Cloth	1	$11.92	$11.92
Total			**$585.17**

Appendix F. Arduino Program Source Code

The following is the source code controlling the Arduino Uno.

```
/*
 * Combined Program: Takes measurements from each sensor.
 * Includes:
 * two pressure sensors
 * three thermocouples with MAX31855 thermocouple amplifier breakout
   boards
 * one AHT20 temperature/humidity sensor
 * one EK1245 Gikfun obstacle avoidance infrared sensor module for shaft
   speed measurement
 *
 * Written by Mary Lichtenwalner
 *
 * Last
   Update:
   April 5,
   2022 */


//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%
// Section 0. Arduino Uno connections
   //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %%%%%%%%%%%%%%%%%%%%%%%

/*
 * All sensors
 * Connect VCC to 5V, GND to ground
 *
 * MAX31855 Thermocouple amplifier breakout board
 * Thermocouple 1: Compressed air, DO to pin 3, CS to pin 4, CLK to
   pin 5
 * Thermocouple 2: Chamber, DO to pin 3, CS to pin 6, CLK to pin 5
 * Thermocouple 3: Exhaust, DO to pin 3, CS to pin 7, CLK to pin 5
 *
 * Pressure transducers
```

```
 * Transducer 1: Ambient air, red to VCC, black to GND, yellow to pin A1
       * Transducer 2: Compressed air, red to VCC, black to GND, yellow to
         pin A2
 *
 * AHT20 temperature/humidity sensor
       * SDA to SDA, SCL to SCL
 *
 * Gikfun infrared sensor module
 * red to VCC, black to GND, yellow
 to pin 2 */

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%
   //   Section 1. Set up sensors

        //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        %%%%%%%%%%%%%%%%%%%%%%

   //   Libraries to include
#include <SPI.h>
#include "Adafruit_MAX31855.h"
#include <Adafruit_AHTX0.h>

//***************************************************************************
   //   Set up thermocouples
//***************************************************************************

   //   First thermocouple
#define MAXDO     3
#define MAXCS1    4
#define MAXCLK    5

Adafruit_MAX31855 thermocouple_1(MAXCLK, MAXCS1, MAXDO);

   //   Second thermocouple (reuse
        DO and CLK) #define MAXCS2
        6

Adafruit_MAX31855 thermocouple_2(MAXCLK, MAXCS2, MAXDO);
```

```
    //  Third thermocouple (reuse
        DO and CLK) #define
        MAXCS3 7

Adafruit_MAX31855 thermocouple_3(MAXCLK, MAXCS3, MAXDO);

//**********************************************************************
    //  Set up pressure transducers
//**********************************************************************

    //  Constants. Set pin numbers for pressure readings.
const int pressurePin1 = A1;
const int pressurePin2 = A2;

// Variables
double pressureReading1 = 0.0;
double pressureReading2 = 0.0;

//**********************************************************************
    //  Set up temperature/humidity sensor
        //**********************************************************************

    //  Initialize sensor
Adafruit_AHTX0 aht;

//**********************************************************************
    //  Set up IR sensor
//**********************************************************************


    //  Variables used in rpm calculation
float value=0;
float rev=0;
int rpm;
int oldtime=0;
int time;
```

```
void isr() //interrupt service routine
{
rev++;
}

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    //      Section 2. Set up program
            //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %%%

void setup() {

    //  Designate two analog pins as input for the pressure transducers
         pinMode(pressurePin1, INPUT);
  pinMode(pressurePin2, INPUT);

    //  Infrared sensor goes in digital pin 2
         attachInterrupt(0,isr,RISING); //attaching the interrupt

    //  Initialize baud rate
  Serial.begin(9600);

  delay(1000);

    //  Initialize
         thermocouples
         thermocouple_1
         .begin();
         thermocouple_2
         .begin();
         thermocouple_3
         .begin();

    //  Initialize temperature/humidity
         sensor aht.begin();
  delay(10);
```

```
}
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    //      Section 3. Part that runs on a loop
          //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

void loop() {

//*********************************************************************
//Goes through and takes a reading from each sensor
//*********************************************************************

    //  Read thermocouples in Fahrenheit
  String temp1 = String(thermocouple_1.readFahrenheit());
  String temp2 = String(thermocouple_2.readFahrenheit());
  String temp3 = String(thermocouple_3.readFahrenheit());

    //  Read pressure transducers
      pressureReading1 =
      analogRead(pressurePin1);
      pressureReading2 =
      analogRead(pressurePin2);


    //  Convert pressures to proper units, want output in psi
    //  Pressure transducer 1
  float voltage1 =
  (pressureReading1*5.0)/1023.0; float
  calibration1 = 0.17;      // Calibrate
  here.
Adjust so that ambient pressure is correct.
  float pressure_pascal1 = (3.0*(voltage1-
  calibration1))*1000000.0; float pressure_bar1 =
  pressure_pascal1/10e5; float pressure_psi1 =
  pressure_bar1*14.5038;

  // Pressure transducer 2
```

```
   float voltage2 =
    (pressureReading2*5.0)/1023.0; float
    calibration2 = 0.17;       // Calibrate ere.
Adjust so that ambient pressure is correct.
  float pressure_pascal2 = (3.0*(voltage2-
   calibration2))*1000000.0; float pressure_bar2 =
   pressure_pascal2/10e5; float pressure_psi2 =
   pressure_bar2*14.5038;

            //   Final output from both
                 pressure transducers
                 String pressure1 =
                 String(pressure_psi1);
                 String pressure2 =
                 String(pressure_psi2);

     //  Read temperature/humidity sensor
    sensors_event_t humidity, temp;
   aht.getEvent(&humidity, &temp); // Populate temp and humidity objects
   with fresh data

    // Read infrared shaft speed sensor
    detachInterrupt(0); //detaches the interrupt
    time=millis()-oldtime; //finds the time
    rpm=(rev/time)*60000; //calculates rpm

    oldtime=millis();
     rev=0;
//saves the current time

//*********************************************************************
     //  Send the output to the serial port
          //*********************************************************************

     //  Combine all measurements into one string with spaces in between
          values String output = temp1 + " " + temp2 + " " + temp3 + " " +
          pressure1 + " " +
```

```
pressure2 + " " + temp.temperature + " " + humidity.relative_humidity + " " + rpm;

        //  Output single string containing
            each measurement
            Serial.println(output);

        //  Uncomment following section to see error codes if thermocouples
            are reading NAN
   //  Serial.print(thermocouple_1.readError());
   //  Serial.print(thermocouple_2.readError());
   //  Serial.print(thermocouple_3.readError());
   //  Serial.print('\n');

//**********************************************************************
   //  Delay before next reading and restart rpm interrupt
       //**********************************************************************

   //  Delay one second before next reading
  delay(1000);
 attachInterrupt(0,isr,RISING);

}
```

Appendix G: Engine Control Tool Source Code

The following is the source code that produces Engine Control Tool.

```
... Tool\EngineControlTool\EngineControlTool\MainWindow.xaml                1
 1 <!--
     ********************************************************************************
     ***************
 2     Main window designer
 3     Written by Mary Lichtenwalner
 4     Last Update: April 10, 2022
 5 ********************************************************************************
     **************-->
 6
 7 <Window
 8         xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 9         xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
10         xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
11         xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
12         xmlns:local="clr-namespace:EngineControlTool"
13         xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
             x:Class="EngineControlTool.MainWindow"
14         mc:Ignorable="d"
15         Title="Engine Control Tool" Height="600" Width="1100" Top="50" Left="50"
16         xmlns:gauge ="clr-
             namespace:Syncfusion.UI.Xaml.Gauges;assembly=Syncfusion.SfGauge.Wpf"
17         Loaded="Window_Loaded">
18
19     <!--
         ********************************************************************************
         ***************
20     Section 1. Window resources. Includes viewModel settings, style settings,
         etc.
21
   ********************************************************************************
   **********-->
22
23     <Window.Resources>
24         <local:ViewModel x:Key="viewModel"/>
25
26         <!--Textbox style-->
27         <Style x:Key="TextBoxTheme" TargetType="TextBox">
28             <Style.Triggers>
29                 <DataTrigger Binding="{Binding Color_Theme}" Value="0">
30                     <Setter Property="Background" Value="#4a4a4a"/>
31                     <Setter Property="IsEnabled" Value="True"/>
32                     <Setter Property="Foreground" Value="White"/>
33                     <Setter Property="BorderBrush" Value="White"/>
34                     <Setter Property="BorderThickness" Value="1"/>
35                 </DataTrigger>
36                 <DataTrigger Binding="{Binding Color_Theme}" Value="1">
37                     <Setter Property="Background" Value="#bfbfbf"/>
38                     <Setter Property="Foreground" Value="Black"/>
39                     <Setter Property="IsEnabled" Value="False"/>
40                     <Setter Property="BorderBrush" Value="Black"/>
41                     <Setter Property="BorderThickness" Value="1"/>
42                 </DataTrigger>
```

... Tool\EngineControlTool\EngineControlTool\MainWindow.xaml 2

```
43              </Style.Triggers>
44          </Style>
45
46          <!--Grid style-->
47          <Style x:Key="BackgroundTheme" TargetType="Grid">
48              <Style.Triggers>
49                  <DataTrigger Binding="{Binding Color_Theme}" Value="0">
50                      <Setter Property="Background" Value="#2e2e2e"/>
51                  </DataTrigger>
52                  <DataTrigger Binding="{Binding Color_Theme}" Value="1">
53                      <Setter Property="Background" Value="White"/>
54                  </DataTrigger>
55              </Style.Triggers>
56          </Style>
57
58          <!--Label style-->
59          <Style x:Key="LabelTheme" TargetType="Label">
60              <Style.Triggers>
61                  <DataTrigger Binding="{Binding Color_Theme}" Value="0">
62                      <Setter Property="Foreground" Value="White"/>
63                  </DataTrigger>
64                  <DataTrigger Binding="{Binding Color_Theme}" Value="1">
65                      <Setter Property="Foreground" Value="Black"/>
66                  </DataTrigger>
67              </Style.Triggers>
68          </Style>
69
70          <!--Button style-->
71          <Style x:Key="ButtonTheme" TargetType="Button">
72              <Setter Property="BorderThickness" Value="3"/>
73              <Style.Triggers>
74                  <DataTrigger Binding="{Binding Color_Theme}" Value="0">
75                      <Setter Property="Foreground" Value="White"/>
76                      <Setter Property="Background" Value="#4a4a4a"/>
77                      <Setter Property="BorderBrush" Value="White"/>
78                  </DataTrigger>
79                  <DataTrigger Binding="{Binding Color_Theme}" Value="1">
80                      <Setter Property="Foreground" Value="Black"/>
81                      <Setter Property="Background" Value="#bfbfbf"/>
82                      <Setter Property="BorderBrush" Value="Black"/>
83                  </DataTrigger>
84                  <Trigger Property="IsMouseOver" Value="True">
85                      <Setter Property="Background" Value="{Binding
                         Accent_Color_String}"/>
86                  </Trigger>
87                  <Trigger Property="IsPressed" Value="True">
88                      <Setter Property="Background" Value="#FFD0A663"/>
89                      <Setter Property="BorderBrush" Value="#FF794937"/>
90                  </Trigger>
91              </Style.Triggers>
92          </Style>
93
```

... Tool\EngineControlTool\EngineControlTool\MainWindow.xaml 3

```
94          <!--GroupBox style-->
95          <Style x:Key="GroupBoxTheme" TargetType="GroupBox">
96              <Style.Triggers>
97                  <DataTrigger Binding="{Binding Color_Theme}" Value="0">
98                      <Setter Property="Foreground" Value="White"/>
99                      <Setter Property="Margin" Value="5"/>
100                 </DataTrigger>
101                 <DataTrigger Binding="{Binding Color_Theme}" Value="1">
102                     <Setter Property="Foreground" Value="Black"/>
103                     <Setter Property="Margin" Value="5"/>
104                 </DataTrigger>
105             </Style.Triggers>
106         </Style>
107
108         <!--Gauge pointer style-->
109         <Style x:Key="PointerTheme" TargetType="gauge:CircularPointer">
110             <Style.Triggers>
111                 <DataTrigger Binding="{Binding Color_Theme}" Value="0">
112                     <Setter Property="NeedlePointerStroke" Value="White"/>
113                 </DataTrigger>
114                 <DataTrigger Binding="{Binding Color_Theme}" Value="1">
115                     <Setter Property="NeedlePointerStroke" Value="Black"/>
116                 </DataTrigger>
117             </Style.Triggers>
118         </Style>
119
120         <!--Gauge style-->
121         <Style x:Key="GaugeTheme" TargetType="gauge:CircularScale">
122             <Setter Property="RimStrokeThickness" Value="3"/>
123             <Setter Property="FontFamily" Value="Spectral"/>
124             <Setter Property="MinorTicksPerInterval" Value="5"/>
125             <Setter Property="LabelPosition" Value="Custom"/>
126             <Setter Property="LabelOffset" Value="0.6"/>
127             <Setter Property="TickStrokeThickness" Value="1"/>
128             <Setter Property="SmallTickStrokeThickness" Value="0.5"/>
129             <Setter Property="TickLength" Value="10"/>
130             <Setter Property="SmallTickLength" Value="5"/>
131             <Setter Property="LabelStroke" Value="#FFACACAC"/>
132             <Style.Triggers>
133                 <DataTrigger Binding="{Binding Accent_Color}" Value="0">
134                     <Setter Property="RimStroke" Value="Red"/>
135                 </DataTrigger>
136                 <DataTrigger Binding="{Binding Accent_Color}" Value="1">
137                     <Setter Property="RimStroke" Value="Orange"/>
138                 </DataTrigger>
139                 <DataTrigger Binding="{Binding Accent_Color}" Value="2">
140                     <Setter Property="RimStroke" Value="Yellow"/>
141                 </DataTrigger>
142                 <DataTrigger Binding="{Binding Accent_Color}" Value="3">
143                     <Setter Property="RimStroke" Value="Green"/>
144                 </DataTrigger>
145                 <DataTrigger Binding="{Binding Accent_Color}" Value="4">
```

... Tool\EngineControlTool\EngineControlTool\MainWindow.xaml 4

```
146                     <Setter Property="RimStroke" Value="#36f5ff"/>
147                 </DataTrigger>
148                 <DataTrigger Binding="{Binding Accent_Color}" Value="5">
149                     <Setter Property="RimStroke" Value="Purple"/>
150                 </DataTrigger>
151                 <DataTrigger Binding="{Binding Accent_Color}" Value="6">
152                     <Setter Property="RimStroke" Value="#ff00e6"/>
153                 </DataTrigger>
154                 <DataTrigger Binding="{Binding Accent_Color}" Value="7">
155                     <Setter Property="RimStroke" Value="#bfa900"/>
156                 </DataTrigger>
157             </Style.Triggers>
158         </Style>
159 
160         <!--Menu style-->
161         <Style x:Key="MenuTheme" TargetType="Menu">
162             <Style.Triggers>
163                 <DataTrigger Binding="{Binding Accent_Color}" Value="0">
164                     <Setter Property="Background" Value="Red"/>
165                 </DataTrigger>
166                 <DataTrigger Binding="{Binding Accent_Color}" Value="1">
167                     <Setter Property="Background" Value="Orange"/>
168                 </DataTrigger>
169                 <DataTrigger Binding="{Binding Accent_Color}" Value="2">
170                     <Setter Property="Background" Value="Yellow"/>
171                 </DataTrigger>
172                 <DataTrigger Binding="{Binding Accent_Color}" Value="3">
173                     <Setter Property="Background" Value="Green"/>
174                 </DataTrigger>
175                 <DataTrigger Binding="{Binding Accent_Color}" Value="4">
176                     <Setter Property="Background" Value="#36f5ff"/>
177                 </DataTrigger>
178                 <DataTrigger Binding="{Binding Accent_Color}" Value="5">
179                     <Setter Property="Background" Value="Purple"/>
180                 </DataTrigger>
181                 <DataTrigger Binding="{Binding Accent_Color}" Value="6">
182                     <Setter Property="Background" Value="#ff00e6"/>
183                 </DataTrigger>
184                 <DataTrigger Binding="{Binding Accent_Color}" Value="7">
185                     <Setter Property="Background" Value="#bfa900"/>
186                 </DataTrigger>
187             </Style.Triggers>
188         </Style>
189 
190         <!--First level menu item style-->
191         <Style x:Key="MenuItemTheme" TargetType="MenuItem">
192             <Setter Property="FontSize" Value="20"/>
193             <Style.Triggers>
194                 <Trigger Property="IsMouseOver" Value="True">
195                     <Setter Property="Foreground" Value="White"/>
196                     <Setter Property="Background" Value="Black"/>
197                 </Trigger>
```

```
... Tool\EngineControlTool\EngineControlTool\MainWindow.xaml                5
198             </Style.Triggers>
199         </Style>
200
201         <!--Second level menu item style-->
202         <Style x:Key="SubMenuItemTheme" TargetType="MenuItem">
203             <Setter Property="FontSize" Value="15"/>
204             <Setter Property="Foreground" Value="Black"/>
205         </Style>
206
207         <!--Control template for button, mouse hover and click commented out and
              replaced with custom
208         setting in button style above-->
209         <ControlTemplate x:Key="ButtonBaseControlTemplate1" TargetType="{x:Type
              ButtonBase}">
210             <Border x:Name="border" BorderBrush="{TemplateBinding BorderBrush}"
                  BorderThickness="{TemplateBinding BorderThickness}"
                  Background="{TemplateBinding Background}"
                  SnapsToDevicePixels="True">
211                 <ContentPresenter x:Name="contentPresenter"
                      ContentTemplate="{TemplateBinding ContentTemplate}"
                      Content="{TemplateBinding Content}"
                      ContentStringFormat="{TemplateBinding ContentStringFormat}"
                      Focusable="False" HorizontalAlignment="{TemplateBinding
                      HorizontalContentAlignment}" Margin="{TemplateBinding Padding}"
                       RecognizesAccessKey="True"
                      SnapsToDevicePixels="{TemplateBinding SnapsToDevicePixels}"
                      VerticalAlignment="{TemplateBinding VerticalContentAlignment}"/
                      >
212             </Border>
213             <ControlTemplate.Triggers>
214                 <Trigger Property="Button.IsDefaulted" Value="True">
215                     <Setter Property="BorderBrush" TargetName="border"
                          Value="{DynamicResource {x:Static
                          SystemColors.HighlightBrushKey}}"/>
216                 </Trigger>
217                 <!--<Trigger Property="IsMouseOver" Value="True">
218                     <Setter Property="Background" TargetName="border"
                          Value="#FFC4E5F6"/>
219                     <Setter Property="BorderBrush" TargetName="border"
                          Value="#FF3C7FB1"/>
220                 </Trigger>-->
221                 <!--<Trigger Property="IsPressed" Value="True">
222                     <Setter Property="Background" TargetName="border"
                          Value="#FFC4E5F6"/>
223                     <Setter Property="BorderBrush" TargetName="border"
                          Value="#FF2C628B"/>
224                 </Trigger>-->
225                 <Trigger Property="ToggleButton.IsChecked" Value="True">
226                     <Setter Property="Background" TargetName="border"
                          Value="#FFBCDDEE"/>
227                     <Setter Property="BorderBrush" TargetName="border"
                          Value="#FF245A83"/>
```

```
... Tool\EngineControlTool\EngineControlTool\MainWindow.xaml                6
228                 </Trigger>
229                 <Trigger Property="IsEnabled" Value="False">
230                     <Setter Property="Background" TargetName="border"
                        Value="#FFF4F4F4"/>
231                     <Setter Property="BorderBrush" TargetName="border"
                        Value="#FFADB2B5"/>
232                     <Setter Property="Foreground" Value="#FF838383"/>
233                 </Trigger>
234             </ControlTemplate.Triggers>
235         </ControlTemplate>
236     </Window.Resources>
237
238     <!--
          ***************************************************************************
          ***************
239     Section 2. Main grid begins here. DataContext set to viewModel
240
    ***************************************************************************
    **********-->
241     <Grid DataContext="{StaticResource viewModel}" Style="{StaticResource
          BackgroundTheme}">
242         <Grid.RowDefinitions>
243             <RowDefinition Height="1*"/>
244             <RowDefinition Height="19*"/>
245         </Grid.RowDefinitions>
246
247         <!--
              ***************************************************************************
              ***************
248         Section 2.1 Menu bar
249
    ***************************************************************************
    *******-->
250         <DockPanel Grid.Row="0">
251             <Menu DockPanel.Dock="Top" Height="Auto" Style="{StaticResource
                  MenuTheme}">
252                 <MenuItem x:Name="Excel_Button" Header="Excel" FontWeight="Black"
                       Style="{StaticResource MenuItemTheme}"
                      Click="Excel_Button_Click"/>
253                 <MenuItem x:Name="Theme_ComboBox" Header="Mode"
                      Style="{StaticResource MenuItemTheme}">
254                     <MenuItem x:Name="DarkItem" Style="{StaticResource
                        SubMenuItemTheme}" Header="Dark Mode"
                        Click="DarkItem_Selected"/>
255                     <MenuItem x:Name="LightItem" Style="{StaticResource
                        SubMenuItemTheme}" Header="Light Mode"
                        Click="LightItem_Selected"/>
256                 </MenuItem>
257                 <MenuItem x:Name="Accent_ComboBox" Header="Color"
                      Style="{StaticResource MenuItemTheme}">
258                     <MenuItem x:Name="RedItem" Style="{StaticResource
                        SubMenuItemTheme}" Header="Red" Click="RedItem_Selected"/>
```

```
... Tool\EngineControlTool\EngineControlTool\MainWindow.xaml                    7
259                     <MenuItem x:Name="OrangeItem" Style="{StaticResource
                         SubMenuItemTheme}" Header="Orange"
                         Click="OrangeItem_Selected"/>
260                     <MenuItem x:Name="YellowItem" Style="{StaticResource
                         SubMenuItemTheme}" Header="Yellow"
                         Click="YellowItem_Selected"/>
261                     <MenuItem x:Name="GreenItem" Style="{StaticResource
                         SubMenuItemTheme}" Header="Green" Click="GreenItem_Selected"/
                         >
262                     <MenuItem x:Name="BlueItem" Style="{StaticResource
                         SubMenuItemTheme}" Header="Blue" Click="BlueItem_Selected"/>
263                     <MenuItem x:Name="PurpleItem" Style="{StaticResource
                         SubMenuItemTheme}" Header="Purple"
                         Click="PurpleItem_Selected"/>
264                     <MenuItem x:Name="PinkItem" Style="{StaticResource
                         SubMenuItemTheme}" Header="Pink" Click="PinkItem_Selected"/>
265                     <MenuItem x:Name="GoldItem" Style="{StaticResource
                         SubMenuItemTheme}" Header="Gold" Click="GoldItem_Selected"/>
266                 </MenuItem>
267             </Menu>
268         </DockPanel>
269
270         <!--
              ************************************************************************
              ***************
271         Section 2.2 Body grid
272
    ************************************************************************
    *******-->
273         <Grid Grid.Row="1">
274             <Grid.ColumnDefinitions>
275                 <ColumnDefinition/>
276                 <ColumnDefinition/>
277                 <ColumnDefinition/>
278                 <ColumnDefinition/>
279             </Grid.ColumnDefinitions>
280             <Grid.RowDefinitions>
281                 <RowDefinition/>
282                 <RowDefinition/>
283             </Grid.RowDefinitions>
284
285             <!--
                  ************************************************************
                  ***************
286             The body grid contains 6 individual GroupBoxes. They are initialized
                  below
287
    ************************************************************************
    ***-->
288
289             <!--Temperature groupbox-->
290             <GroupBox Grid.Row="0" Grid.Column="0" Grid.RowSpan="2"
```

```
... Tool\EngineControlTool\EngineControlTool\MainWindow.xaml                    8
                Header="Temperature (ºF)" Style="{StaticResource GroupBoxTheme}">
291               <Grid>
292                   <Grid.RowDefinitions>
293                       <RowDefinition/>
294                       <RowDefinition/>
295                       <RowDefinition/>
296                       <RowDefinition/>
297
298                       <RowDefinition/>
299                       <RowDefinition/>
300                       <RowDefinition/>
301                       <RowDefinition/>
302
303                       <RowDefinition/>
304                       <RowDefinition/>
305                       <RowDefinition/>
306                       <RowDefinition/>
307
308                       <RowDefinition/>
309                       <RowDefinition/>
310                       <RowDefinition/>
311                       <RowDefinition/>
312                   </Grid.RowDefinitions>
313
314                   <Grid.ColumnDefinitions>
315                       <ColumnDefinition Width="2*"/>
316                       <ColumnDefinition Width="1*"/>
317                   </Grid.ColumnDefinitions>
318
319                   <Label Grid.Row="1" Grid.Column="1" Content="Ambient"        ➲
                    VerticalAlignment="Center" Style="{StaticResource           ➲
                    LabelTheme}"/>
320                 <Label Grid.Row="5" Grid.Column="1" Content="Compressed"     ➲
                    VerticalAlignment="Center" Style="{StaticResource           ➲
                    LabelTheme}"/>
321                 <Label Grid.Row="9" Grid.Column="1" Content="Chamber"        ➲
                    VerticalAlignment="Center" Style="{StaticResource           ➲
                    LabelTheme}"/>
322                 <Label Grid.Row="13" Grid.Column="1" Content="Exhaust"       ➲
                    VerticalAlignment="Center" Style="{StaticResource           ➲
                    LabelTheme}"/>
323
324                 <Label Grid.Row="2" Grid.Column="1" x:Name="AmbientTemp_Box" ➲
                    Height="30" Style="{StaticResource LabelTheme}"             ➲
                    Content="{Binding Path = tempAmbient,                       ➲
                    Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}"/>
325                 <Label Grid.Row="6" Grid.Column="1"                          ➲
                    x:Name="CompressedTemp_Box" Height="30"                     ➲
                    Style="{StaticResource LabelTheme}" Content="{Binding Path = ➲
                    tempCompressed,                                             ➲
                    Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}"/>
326                 <Label Grid.Row="10" Grid.Column="1" x:Name="ChamberTemp_Box" ➲
```

```
... Tool\EngineControlTool\EngineControlTool\MainWindow.xaml                    9
                        Height="30" Style="{StaticResource LabelTheme}"
                       Content="{Binding Path = tempChamber,
                       Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}"/>
327                    <Label Grid.Row="14" Grid.Column="1" x:Name="ExhaustTemp_Box"
                        Height="30" Style="{StaticResource LabelTheme}"
                       Content="{Binding Path = tempExhaust,
                       Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}"/>
328
329                    <!--Ambient temperature gauge-->
330                    <gauge:SfCircularGauge Grid.Row="0" Grid.RowSpan="4"
                       x:Name="AmbientTemp_Gauge" HeaderAlignment="Custom"
                       GaugeHeader="Ambient Temp (ºF)" GaugeHeaderPosition="0.23,
                       1.1" HorizontalAlignment="Center">
331
332                        <gauge:SfCircularGauge.Scales>
333                            <gauge:CircularScale RadiusFactor="1" StartValue="60"
                        EndValue="90" Interval="5" Style="{StaticResource
                       GaugeTheme}">
334                                <gauge:CircularScale.Pointers>
335                                    <gauge:CircularPointer Style="{StaticResource
                        PointerTheme}" Value="{Binding Path=tempAmbient,
                       Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}" />
336                                </gauge:CircularScale.Pointers>
337                            </gauge:CircularScale>
338                        </gauge:SfCircularGauge.Scales>
339
340                    </gauge:SfCircularGauge>
341
342                    <!--Compressed temperature gauge-->
343                    <gauge:SfCircularGauge Grid.Row="4" Grid.RowSpan="4"
                       x:Name="CompressedTemp_Gauge" HeaderAlignment="Custom"
                       GaugeHeader="Ambient Temp (ºF)" GaugeHeaderPosition="0.23,
                       1.1" HorizontalAlignment="Center">
344
345                        <gauge:SfCircularGauge.Scales>
346                            <gauge:CircularScale RadiusFactor="1" StartValue="60"
                        EndValue="200" Interval="20" Style="{StaticResource
                       GaugeTheme}">
347                                <gauge:CircularScale.Pointers>
348                                    <gauge:CircularPointer Style="{StaticResource
                        PointerTheme}" Value="{Binding Path=tempCompressed,
                       Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}" />
349                                </gauge:CircularScale.Pointers>
350                            </gauge:CircularScale>
351                        </gauge:SfCircularGauge.Scales>
352
353                    </gauge:SfCircularGauge>
354
355                    <!--Chamber temperature gauge-->
356                    <gauge:SfCircularGauge Grid.Row="8" Grid.RowSpan="4"
                       x:Name="ChamberTemp_Gauge" HeaderAlignment="Custom"
                       GaugeHeader="Ambient Temp (ºF)" GaugeHeaderPosition="0.23,
```

```
... Tool\EngineControlTool\EngineControlTool\MainWindow.xaml                10
                        1.1" HorizontalAlignment="Center">
357
358                         <gauge:SfCircularGauge.Scales>
359                             <gauge:CircularScale RadiusFactor="1" StartValue="60" ↵
                         EndValue="2200" Interval="500" Style="{StaticResource       ↵
                        GaugeTheme}">
360                                 <gauge:CircularScale.Pointers>
361                                     <gauge:CircularPointer Style="{StaticResource ↵
                         PointerTheme}" Value="{Binding Path=tempChamber,             ↵
                        Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}" />
362                                 </gauge:CircularScale.Pointers>
363                             </gauge:CircularScale>
364                         </gauge:SfCircularGauge.Scales>
365
366                     </gauge:SfCircularGauge>
367
368                     <!--Exhaust temperature gauge-->
369                     <gauge:SfCircularGauge Grid.Row="12" Grid.RowSpan="4"        ↵
                        x:Name="ExhaustTemp_Gauge" HeaderAlignment="Custom"          ↵
                        GaugeHeader="Ambient Temp (ºF)" GaugeHeaderPosition="0.23,   ↵
                        1.1" HorizontalAlignment="Center">
370
371                         <gauge:SfCircularGauge.Scales>
372                             <gauge:CircularScale RadiusFactor="1" StartValue="60" ↵
                         EndValue="2200" Interval="500" Style="{StaticResource       ↵
                        GaugeTheme}">
373                                 <gauge:CircularScale.Pointers>
374                                     <gauge:CircularPointer Style="{StaticResource ↵
                         PointerTheme}" Value="{Binding Path=tempExhaust,             ↵
                        Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}" />
375                                 </gauge:CircularScale.Pointers>
376                             </gauge:CircularScale>
377                         </gauge:SfCircularGauge.Scales>
378
379                     </gauge:SfCircularGauge>
380
381                 </Grid>
382             </GroupBox>
383
384             <!--Pressure groupbox-->
385             <GroupBox Grid.Column="1" Grid.RowSpan="2" Header="Pressure (psi)"    ↵
                  Style="{StaticResource GroupBoxTheme}">
386                 <Grid>
387                     <Grid.RowDefinitions>
388                         <RowDefinition/>
389                         <RowDefinition/>
390                         <RowDefinition/>
391                         <RowDefinition/>
392
393                         <RowDefinition/>
394                         <RowDefinition/>
395                         <RowDefinition/>
```

```
... Tool\EngineControlTool\EngineControlTool\MainWindow.xaml                    11
396                         <RowDefinition/>
397
398                         <RowDefinition/>
399                         <RowDefinition/>
400                         <RowDefinition/>
401                         <RowDefinition/>
402
403                         <RowDefinition/>
404                         <RowDefinition/>
405                         <RowDefinition/>
406                         <RowDefinition/>
407                     </Grid.RowDefinitions>
408
409                     <Grid.ColumnDefinitions>
410                         <ColumnDefinition Width="2*"/>
411                         <ColumnDefinition Width="1*"/>
412                     </Grid.ColumnDefinitions>
413
414                     <Label Grid.Row="1" Grid.Column="1" Content="Ambient"          ↵
                        VerticalAlignment="Center" Style="{StaticResource             ↵
                        LabelTheme}"/>
415                     <Label Grid.Row="5" Grid.Column="1" Content="Compressed"       ↵
                        VerticalAlignment="Center" Style="{StaticResource             ↵
                        LabelTheme}"/>
416
417                     <Label Grid.Row="2" Grid.Column="1"                           ↵
                        x:Name="AmbientPressure_Box" Height="30"                      ↵
                        Style="{StaticResource LabelTheme}" Content="{Binding Path =  ↵
                        pressureAmbient,                                              ↵
                        Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}"/>
418                     <Label Grid.Row="6" Grid.Column="1"                           ↵
                        x:Name="CompressedPressure_Box" Height="30"                   ↵
                        Style="{StaticResource LabelTheme}" Content="{Binding Path =  ↵
                        pressureCompressed,                                           ↵
                        Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}"/>
419
420                     <!--Ambient pressure gauge-->
421                     <gauge:SfCircularGauge Grid.Row="0" Grid.RowSpan="4"          ↵
                        x:Name="AmbientPressure_Gauge" HeaderAlignment="Custom"       ↵
                        GaugeHeader="Ambient Pressure (psi)"                          ↵
                        GaugeHeaderPosition="0.20, 1.1" HorizontalAlignment="Center">
422
423                         <gauge:SfCircularGauge.Scales>
424                             <gauge:CircularScale RadiusFactor="1" StartValue="0"  ↵
                        EndValue="50" Interval="10" Style="{StaticResource            ↵
                        GaugeTheme}">
425                                 <gauge:CircularScale.Pointers>
426                                     <gauge:CircularPointer Style="{StaticResource ↵
                         PointerTheme}" Value="{Binding Path=pressureAmbient,          ↵
                        Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}" />
427                                 </gauge:CircularScale.Pointers>
428                             </gauge:CircularScale>
```

```
... Tool\EngineControlTool\EngineControlTool\MainWindow.xaml                    12
429                         </gauge:SfCircularGauge.Scales>
430
431                     </gauge:SfCircularGauge>
432
433                     <!--Compressed pressure gauge-->
434                     <gauge:SfCircularGauge Grid.Row="4" Grid.RowSpan="4"
                        x:Name="CompressedPressure_Gauge" HeaderAlignment="Custom"
                        GaugeHeader="Ambient Pressure (psi)"
                        GaugeHeaderPosition="0.20, 1.1" HorizontalAlignment="Center">
435
436                         <gauge:SfCircularGauge.Scales>
437                             <gauge:CircularScale RadiusFactor="1" StartValue="0"
                        EndValue="50" Interval="10" Style="{StaticResource
                        GaugeTheme}">
438                                 <gauge:CircularScale.Pointers>
439                                     <gauge:CircularPointer Style="{StaticResource
                         PointerTheme}" Value="{Binding Path=pressureCompressed,
                        Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}" />
440                                 </gauge:CircularScale.Pointers>
441                             </gauge:CircularScale>
442                         </gauge:SfCircularGauge.Scales>
443
444                     </gauge:SfCircularGauge>
445
446                 </Grid>
447             </GroupBox>
448
449             <!--Humidity groupbox-->
450             <GroupBox Grid.Column="2" Header="Humidity (%)"
                  Style="{StaticResource GroupBoxTheme}">
451                 <Grid>
452                     <Grid.RowDefinitions>
453                         <RowDefinition/>
454                         <RowDefinition/>
455                         <RowDefinition/>
456                         <RowDefinition/>
457
458                         <RowDefinition/>
459                         <RowDefinition/>
460                         <RowDefinition/>
461                         <RowDefinition/>
462                     </Grid.RowDefinitions>
463
464                     <Grid.ColumnDefinitions>
465                         <ColumnDefinition Width="2*"/>
466                         <ColumnDefinition Width="1*"/>
467                     </Grid.ColumnDefinitions>
468
469                     <Label Grid.Row="1" Grid.Column="1" Content="Ambient"
                        VerticalAlignment="Center" Style="{StaticResource
                        LabelTheme}"/>
470
```

```
... Tool\EngineControlTool\EngineControlTool\MainWindow.xaml                    13
471                     <Label Grid.Row="2" Grid.Column="1" x:Name="AmbientHum_Box"
                        Height="30" Style="{StaticResource LabelTheme}"
                        Content="{Binding humidity}"/>
472
473                     <!--Humidity gauge-->
474                     <gauge:SfCircularGauge Grid.Row="0" Grid.RowSpan="4"
                        x:Name="Humidity_Gauge" HeaderAlignment="Custom"
                        GaugeHeader="Ambient Humidity (%)" GaugeHeaderPosition="0.2,
                        1.1" HorizontalAlignment="Center">
475
476                         <gauge:SfCircularGauge.Scales>
477                             <gauge:CircularScale RadiusFactor="1" StartValue="0"
                        EndValue="100" Interval="20" Style="{StaticResource
                        GaugeTheme}">
478                                 <gauge:CircularScale.Pointers>
479                                     <gauge:CircularPointer Style="{StaticResource
                         PointerTheme}" Value="{Binding Path=humidity, Mode=TwoWay,
                        UpdateSourceTrigger=PropertyChanged}" />
480                                 </gauge:CircularScale.Pointers>
481                             </gauge:CircularScale>
482                         </gauge:SfCircularGauge.Scales>
483
484                     </gauge:SfCircularGauge>
485
486                 </Grid>
487             </GroupBox>
488
489             <!--Shaft speed groupbox-->
490             <GroupBox Grid.Column="3" Header="Shaft Speed (RPM)"
                  Style="{StaticResource GroupBoxTheme}">
491                 <Grid>
492                     <Grid.RowDefinitions>
493                         <RowDefinition/>
494                         <RowDefinition/>
495                         <RowDefinition/>
496                         <RowDefinition/>
497
498                         <RowDefinition/>
499                         <RowDefinition/>
500                         <RowDefinition/>
501                         <RowDefinition/>
502                     </Grid.RowDefinitions>
503
504                     <Grid.ColumnDefinitions>
505                         <ColumnDefinition Width="2*"/>
506                         <ColumnDefinition Width="1*"/>
507                     </Grid.ColumnDefinitions>
508
509                     <Label Grid.Row="1" Grid.Column="1" Content="Turbine"
                        VerticalAlignment="Center" Style="{StaticResource
                        LabelTheme}"/>
510                     <Label Grid.Row="2" Grid.Column="1" x:Name="ShaftSpeed_Box"
```

```
... Tool\EngineControlTool\EngineControlTool\MainWindow.xaml                14
                        Height="30" Style="{StaticResource LabelTheme}"
                        Content="{Binding shaftSpeed}"/>
511
512                     <!--Shaft speed gauge-->
513                     <gauge:SfCircularGauge Grid.Row="0" Grid.RowSpan="4"
                        x:Name="ShaftSpeed_Gauge" HeaderAlignment="Custom"
                        GaugeHeader="Shaft Speed (RPM)" GaugeHeaderPosition="0.23,
                        1.1" HorizontalAlignment="Center">
514
515                         <gauge:SfCircularGauge.Scales>
516                             <gauge:CircularScale RadiusFactor="1" StartValue="0"
                        EndValue="6000" Interval="2000" Style="{StaticResource
                        GaugeTheme}">
517                                 <gauge:CircularScale.Pointers>
518                                     <gauge:CircularPointer Style="{StaticResource
                         PointerTheme}" Value="{Binding Path=shaftSpeed, Mode=TwoWay,
                         UpdateSourceTrigger=PropertyChanged}" />
519                                 </gauge:CircularScale.Pointers>
520                             </gauge:CircularScale>
521                         </gauge:SfCircularGauge.Scales>
522
523                     </gauge:SfCircularGauge>
524
525                 </Grid>
526             </GroupBox>
527
528             <!--Flow Rate groupbox-->
529             <GroupBox Grid.Row="1" Grid.Column="2" Header="Flow Rate (lb/in)"
                  Style="{StaticResource GroupBoxTheme}">
530                 <Grid>
531                     <Grid.RowDefinitions>
532                         <RowDefinition/>
533                         <RowDefinition/>
534                         <RowDefinition/>
535                     </Grid.RowDefinitions>
536
537                     <Grid.ColumnDefinitions>
538                         <ColumnDefinition/>
539                         <ColumnDefinition/>
540                         <ColumnDefinition/>
541                     </Grid.ColumnDefinitions>
542
543                     <Label Grid.Row="1" Grid.Column="0" Content="Flow Rate"
                        VerticalAlignment="Center" Style="{StaticResource
                        LabelTheme}"/>
544                     <TextBox Grid.Row="1" Grid.Column="1" x:Name="FlowRate_Box"
                        Height="30" Style="{StaticResource TextBoxTheme}"
                        TextChanged="FlowRate_Box_TextChanged"/>
545
546                 </Grid>
547             </GroupBox>
548
```

```
... Tool\EngineControlTool\EngineControlTool\MainWindow.xaml                15
549             <!--Log groupbox-->
550             <GroupBox Grid.Row="1" Grid.Column="3" Header="Log"
                  Style="{StaticResource GroupBoxTheme}">
551                 <Grid>
552                     <Grid.RowDefinitions>
553                         <RowDefinition/>
554                         <RowDefinition/>
555                         <RowDefinition/>
556                     </Grid.RowDefinitions>
557
558                     <Grid.ColumnDefinitions>
559                         <ColumnDefinition/>
560                         <ColumnDefinition/>
561                         <ColumnDefinition/>
562                     </Grid.ColumnDefinitions>
563
564                     <Label Grid.Row="0" Grid.Column="0" Grid.ColumnSpan="3"
                      Content="Enter log message." HorizontalAlignment="Center"
                      VerticalAlignment="Center" Style="{StaticResource
                      LabelTheme}" />
565                     <Button x:Name="Log_Button" Grid.Row="2" Grid.Column="1"
                      Content="Log" VerticalAlignment="Center"
                      Style="{StaticResource ButtonTheme}"
                      Template="{DynamicResource ButtonBaseControlTemplate1}"
                      Click="Log_Button_Click"/>
566                     <TextBox Grid.Row="1" Grid.Column="0" Grid.ColumnSpan="3"
                      x:Name="Log_Box" Height="30" Style="{StaticResource
                      TextBoxTheme}" Text="{Binding Path = logNote,
                      Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}"/>
567
568                 </Grid>
569             </GroupBox>
570
571             <!--
                  ************************************************************
                  ***************
572             Section 3. Warning label to overlay screen if necessary
573
    ****************************************************************************
    ***-->
574
575             <!--Grid containing warning label-->
576             <Grid x:Name="WarningGrid" Margin="200" Grid.Row="0" Grid.Column="0"
                  Grid.RowSpan="2" Grid.ColumnSpan="4" Background="#fffb7d">
577                 <Grid.Style>
578                     <Style TargetType="Grid">
579                         <Style.Triggers>
580                             <DataTrigger Binding="{Binding showWarning}"
                          Value="0">
581                                 <Setter Property="Visibility" Value="Hidden"/>
582                             </DataTrigger>
583                             <DataTrigger Binding="{Binding showWarning}"
```

```
... Tool\EngineControlTool\EngineControlTool\MainWindow.xaml                16
                        Value="1">
584                             <Setter Property="Visibility" Value="Visible"/>
585                         </DataTrigger>
586                     </Style.Triggers>
587                 </Style>
588             </Grid.Style>
589             <Grid.RowDefinitions>
590                 <RowDefinition Height="2.6*"/>
591                 <RowDefinition Height="2*"/>
592                 <RowDefinition Height="1.8*"/>
593             </Grid.RowDefinitions>
594
595             <Label Grid.Row="0" FontSize="40" Content="WARNING!"            ↵
                  HorizontalAlignment="Center" VerticalAlignment="Center"       ↵
                  Foreground="Red"/>
596
597             <Label Grid.Row="1" FontSize="25" Content="Shut off propane."   ↵
                  HorizontalAlignment="Center" Foreground="Red"/>
598             <Label Grid.Row="2" FontSize="20" HorizontalAlignment="Center"  ↵
                  Foreground="Red">
599                 <Label.Style>
600                     <Style TargetType="Label">
601                         <Style.Triggers>
602                             <DataTrigger Binding="{Binding tempOrPressure}"  ↵
                        Value="0">
603                                 <Setter Property="Content" Value="High      ↵
                        temperature detected!"/>
604                             </DataTrigger>
605                             <DataTrigger Binding="{Binding tempOrPressure}"  ↵
                        Value="1">
606                                 <Setter Property="Content" Value="High      ↵
                        pressure detected!"/>
607                             </DataTrigger>
608                         </Style.Triggers>
609                     </Style>
610                 </Label.Style>
611             </Label>
612
613         </Grid>
614
615     </Grid>
616
617 </Grid>
618 </Window>
619
```

...ol\EngineControlTool\EngineControlTool\MainWindow.xaml.cs 1

```
1  ///
2  /// Engine Control Tool
3  ///     Receives data from Arduino and displays it in GUI
4  ///     Records measurements and logs in an Excel spreadsheet when user clicks
     "Excel" button
5  ///
6  /// MainWindow.xaml.cs
7  ///     Class that controls program
8  ///     Sets colors on GUI
9  ///     Receives data from serial port and passes data between View Model
10 ///     Checks for dangerous temperature or pressure values
11 ///     Initializes Excel creation
12 ///
13 /// Written by Mary Lichtenwalner
14 ///
15 /// Last Update: April 10, 2022
16 ///
17
18 using System;
19 using System.IO.Ports;
20 using System.Threading;
21 using System.Windows;
22
23
24 namespace EngineControlTool
25 {
26     /// <summary>
27     /// Interaction logic for MainWindow.xaml
28     /// </summary>
29     public partial class MainWindow : System.Windows.Window
30     {
31         // Initialize possible ports
32         private SerialPort port;
33         private SerialPort port1 = new SerialPort("COM3", 9600, Parity.None, 8,
           StopBits.One);
34         private SerialPort port2 = new SerialPort("COM4", 9600, Parity.None, 8,
           StopBits.One);
35         private SerialPort port3 = new SerialPort("COM5", 9600, Parity.None, 8,
           StopBits.One);
36
37         // Initialize view model and Excel classes
38         public ViewModel viewModel;
39         public CreateExcel createExcel;
40
41         // Initialize thread to check for dangerous temperature/pressure values
42         private Thread warningCheck;
43
44         // Variables to save current sensor readings
45         double tempCompressed;
46         double tempChamber;
47         double tempExhaust;
48         double pressureAmbient;
```

...ol\EngineControlTool\EngineControlTool\MainWindow.xaml.cs 2

```
49          double pressureCompressed;
50          double tempAmbient;
51          double humidity;
52          double shaftSpeed;
53
54          //
              ************************************************************************
              ******************
55          // Method 1. Construct MainWindow
56          //
              ************************************************************************
              ******************
57          public MainWindow()
58          {
59              InitializeComponent();
60
61              // Set viewModel and createExcel to an instance of respective objects
62              viewModel = this.FindResource("viewModel") as ViewModel;
63              createExcel = new CreateExcel(viewModel);
64
65              // Start the thread to check for dangerous pressures/temperatures
66              warningCheck = new Thread(new ThreadStart(WarningChecker));
67              warningCheck.IsBackground = true;
68              warningCheck.Start();
69          }
70
71          //
              ************************************************************************
              ******************
72          // Methods 2-11. Methods to change colors in GUI. Dark or light mode,
              accent color
73          //
              ************************************************************************
              ******************
74          // Set correct variable in viewModel to corresponding color based on user
               selection
75          //
              ************************************************************************
              ******************
76          private void DarkItem_Selected(object sender, RoutedEventArgs e)
77          {
78              viewModel.Color_Theme = 0;
79          }
80
81          private void LightItem_Selected(object sender, RoutedEventArgs e)
82          {
83              viewModel.Color_Theme = 1;
84          }
85
86          private void RedItem_Selected(object sender, RoutedEventArgs e)
87          {
88              viewModel.Accent_Color = 0;
```

...ol\EngineControlTool\EngineControlTool\MainWindow.xaml.cs 3

```
 89              viewModel.Accent_Color_String = "Red";
 90          }
 91
 92          private void OrangeItem_Selected(object sender, RoutedEventArgs e)
 93          {
 94              viewModel.Accent_Color = 1;
 95              viewModel.Accent_Color_String = "Orange";
 96          }
 97
 98          private void YellowItem_Selected(object sender, RoutedEventArgs e)
 99          {
100              viewModel.Accent_Color = 2;
101              viewModel.Accent_Color_String = "Yellow";
102          }
103
104          private void GreenItem_Selected(object sender, RoutedEventArgs e)
105          {
106              viewModel.Accent_Color = 3;
107              viewModel.Accent_Color_String = "Green";
108          }
109
110          private void BlueItem_Selected(object sender, RoutedEventArgs e)
111          {
112              viewModel.Accent_Color = 4;
113              viewModel.Accent_Color_String = "#36f5ff";
114          }
115
116          private void PurpleItem_Selected(object sender, RoutedEventArgs e)
117          {
118              viewModel.Accent_Color = 5;
119              viewModel.Accent_Color_String = "Purple";
120          }
121
122          private void PinkItem_Selected(object sender, RoutedEventArgs e)
123          {
124              viewModel.Accent_Color = 6;
125              viewModel.Accent_Color_String = "#ff00e6";
126          }
127
128          private void GoldItem_Selected(object sender, RoutedEventArgs e)
129          {
130              viewModel.Accent_Color = 7;
131              viewModel.Accent_Color_String = "#bfa900";
132          }
133
134          //
              ************************************************************************
              *****************
135          // Method 12. Method runs when window loads. Find the correct port and
              begin reading data
136          //
              ************************************************************************
```

...ol\EngineControlTool\EngineControlTool\MainWindow.xaml.cs 4

```
            ******************
137         [STAThread]
138         private void Window_Loaded(object sender, RoutedEventArgs e)
139         {
140             // Try reading from serial port "COM3"
141             try
142             {
143                 port1.Open();
144                 port = port1;
145             }
146             catch
147             {
148                 // Try reading from serial port "COM4"
149                 try
150                 {
151                     port2.Open();
152                     port = port2;
153                 }
154                 catch
155                 {
156                     // Try reading from serial port "COM5"
157                     try
158                     {
159                         port3.Open();
160                         port = port3;
161                     }
162                     // If no port successfully connects, just set port to port1 ↵
                         to avoid crashing
163                     catch
164                     {
165                         port = port1;
166                     }
167                 }
168             }
169
170             // Calls method to receive serial port data
171             SerialPortProgram();
172         }
173
174         //                                                                    ↵
              ********************************************************************* ↵
              ******************
175         // Method 13. Serial Port Program reads incoming data from serial port
176         //                                                                    ↵
              ********************************************************************* ↵
              ******************
177         private void SerialPortProgram()
178         {
179             Console.WriteLine("Incoming Data:");
180
181             // Receiving data triggers method "port_DataReceived"
182             port.DataReceived += new SerialDataReceivedEventHandler            ↵
```

...ol\EngineControlTool\EngineControlTool\MainWindow.xaml.cs 5

```
                (port_DataReceived);
183             try
184             {
185                 port.Open();
186             }
187             catch
188             {
189                 // If port does not receive data, do nothing
190             }
191             Console.ReadLine();
192         }
193
194         //
              ************************************************************************
              *****************
195         // Method 14. Uses the data received from the serial port
196         //
              ************************************************************************
              *****************
197         private void port_DataReceived(object sender, SerialDataReceivedEventArgs
              e)
198         {
199             // Reads the string from the Arduino and splits the values at the
                spaces
200             string measurements = port.ReadLine();
201             string[] meas = measurements.Split(' ');
202
203             // Grab a current time stamp
204             DateTime now = DateTime.Now;
205
206             // Convert all the measurements from strings to doubles
207             // If not possible (ex: thermocouple had error so reading was NAN),
                just set value to 0
208             try
209             {
210                 tempCompressed = Convert.ToDouble(meas[0]);
211             }
212             catch
213             {
214                 tempCompressed = 0;
215             }
216             try
217             {
218                 tempChamber = Convert.ToDouble(meas[1]);
219             }
220             catch
221             {
222                 tempChamber = 0;
223             }
224             try
225             {
226                 tempExhaust = Convert.ToDouble(meas[2]);
```

...ol\EngineControlTool\EngineControlTool\MainWindow.xaml.cs 6

```
227             }
228             catch
229             {
230                 tempExhaust = 0;
231             }
232             try
233             {
234                 pressureAmbient = Convert.ToDouble(meas[3]);
235             }
236             catch
237             {
238                 pressureAmbient = 0;
239             }
240             try
241             {
242                 pressureCompressed = Convert.ToDouble(meas[4]);
243             }
244             catch
245             {
246                 pressureCompressed = 0;
247             }
248             try
249             {
250                 tempAmbient = 1.8 * Convert.ToDouble(meas[5]) + 32;
251             }
252             catch
253             {
254                 tempAmbient = 0;
255             }
256             try
257             {
258                 humidity = Convert.ToDouble(meas[6]);
259             }
260             catch
261             {
262                 humidity = 0;
263             }
264             try
265             {
266                 shaftSpeed = Convert.ToDouble(meas[7]);
267             }
268             catch
269             {
270                 shaftSpeed = 0;
271             }
272 
273             // Take all the measurements and place in correct spot in viewModel
274             // These values are bound to correct label/gauge in GUI
275             viewModel.tempCompressed = tempCompressed;
276             viewModel.tempChamber = tempChamber;
277             viewModel.tempExhaust = tempExhaust;
278             viewModel.pressureAmbient = pressureAmbient;
```

...ol\EngineControlTool\EngineControlTool\MainWindow.xaml.cs 7

```
279             viewModel.pressureCompressed = pressureCompressed;
280             viewModel.tempAmbient = tempAmbient;
281             viewModel.humidity = humidity;
282             viewModel.shaftSpeed = shaftSpeed;
283
284             // Add the current measurement to the correct list in viewModel
285             // These lists will be used in the Excel spreadsheet later
286             viewModel.timeList.Add(Convert.ToString(now));
287             viewModel.tCompressedList.Add(tempCompressed);
288             viewModel.tChamberList.Add(tempChamber);
289             viewModel.tExhaustList.Add(tempExhaust);
290             viewModel.pAmbientList.Add(pressureAmbient);
291             viewModel.pCompressedList.Add(pressureCompressed);
292             viewModel.tAmbientList.Add(tempAmbient);
293             viewModel.humidityList.Add(humidity);
294             viewModel.shaftSpeedList.Add(shaftSpeed);
295
296             // Check to see if the log button was clicked
297             // If so, log the inputted message or just mark the time if no
                  message inputted
298             if (viewModel.Log)
299             {
300                 // Add the log message to the list in the viewModel
301                 viewModel.logList.Add("Log: " + viewModel.logNote);
302
303                 // Reset the log checker to false
304                 viewModel.Log = false;
305
306                 // Clear the log message
307                 viewModel.logNote = "";
308             }
309
310             // If log button was not clicked, just add a blank to the log list
311             else
312             {
313                 viewModel.logList.Add("");
314             }
315         }
316
317         //
              ************************************************************************
              *****************
318         // Method 15. Run when user clicks "Excel" button
319         //
              ************************************************************************
              *****************
320         private void Excel_Button_Click(object sender, RoutedEventArgs e)
321         {
322             try
323             {
324                 // Call method to initialize the Excel application
325                 createExcel.startUpExcel();
```

...ol\EngineControlTool\EngineControlTool\MainWindow.xaml.cs 8

```
326
327                 // Create the first spreadsheet with recorded values taken from
                      the viewModel
328                 createExcel.generateExcel(viewModel.timeList,
                      viewModel.tCompressedList, viewModel.tChamberList,
                      viewModel.tExhaustList,
329                     viewModel.pAmbientList, viewModel.pCompressedList,
                        viewModel.tAmbientList, viewModel.humidityList,
                        viewModel.shaftSpeedList,
330                     viewModel.logList);
331
332                 // Create a second spreadsheet that has data in metric units
                      (Celsius and pascals)
333                 createExcel.generateMetricExcel(viewModel.timeList,
                      viewModel.tCompressedList, viewModel.tChamberList,
                      viewModel.tExhaustList,
334                     viewModel.pAmbientList, viewModel.pCompressedList,
                        viewModel.tAmbientList, viewModel.humidityList,
                        viewModel.shaftSpeedList,
335                     viewModel.logList);
336             }
337             catch
338             {
339                 // Do nothing
340             }
341
342         }
343
344         //
              ************************************************************************
              *****************
345         // Method 16. Run if user changes flow rate value
346         //
              ************************************************************************
              *****************
347         private void FlowRate_Box_TextChanged(object sender, RoutedEventArgs e)
348         {
349             // Try to convert the user input to a double
350             try
351             {
352                 viewModel.flowRate = Convert.ToDouble(FlowRate_Box.Text);
353             }
354
355             // If unable to convert to double, set text box input to 0
356             catch
357             {
358
359                 viewModel.flowRate = 0;
360                 FlowRate_Box.Text = "0";
361             }
362         }
363
```

...ol\EngineControlTool\EngineControlTool\MainWindow.xaml.cs 9

```
364         //                                                                          ➥
              ************************************************************************ ➥
              *****************
365         // Method 17. Checks for dangerous temperature or pressure values
366         //                                                                          ➥
              ************************************************************************ ➥
              *****************
367         private void WarningChecker()
368         {
369             // Limits on temperature and pressure (degrees Fahrenheit and psi)
370             double tempLimit = 10000;
371             double pressureLimit = 10000;
372
373             // Constantly checking
374             while (true)
375             {
376                 // If any temperature value is above limit, set                    ➥
                    viewModel.isDangerous to true
377                 if (viewModel.tempAmbient > tempLimit || viewModel.tempCompressed ➥
                       > tempLimit ||
378                     viewModel.tempChamber > tempLimit || viewModel.tempExhaust >  ➥
                     tempLimit)
379                 {
380                     viewModel.isDangerous = true;
381                     viewModel.tempOrPressure = 0;
382                 }
383
384                 // If any pressure value is above limit, set                       ➥
                    viewModel.isDangerous to true
385                 else if (viewModel.pressureAmbient > pressureLimit ||              ➥
                    viewModel.pressureCompressed > pressureLimit)
386                 {
387                     viewModel.isDangerous = true;
388                     viewModel.tempOrPressure = 1;
389                 }
390
391                 // If everything is within safe limits, set the boolean to false
392                 else
393                 {
394                     viewModel.isDangerous = false;
395                 }
396
397                 // If the boolen is set to true, flash the warning label so that  ➥
                    the user can stop the propane flow
398                 if (viewModel.isDangerous)
399                 {
400                     for (int i = 0; i < 3; i++)
401                     {
402                         viewModel.showWarning = 1;
403                         Thread.Sleep(1000);
404                         viewModel.showWarning = 0;
405                         Thread.Sleep(1000);
```

...ol\EngineControlTool\EngineControlTool\MainWindow.xaml.cs 10

```
406                     }
407                 }
408             }
409         }
410
411         //                                                                  ➥
              ********************************************************************* ➥
              *****************
412         // Method 18. Runs when user clicks the log button
413         //                                                                  ➥
              ********************************************************************* ➥
              *****************
414         private void Log_Button_Click(object sender, RoutedEventArgs e)
415         {
416             // Sets the log value in viewModel to true so that a marker is placed ➥
                   in the Excel
417             // Message is directly bound from text box to viewModel
418             viewModel.Log = true;
419         }
420     }
421 }
422
```

...l Tool\EngineControlTool\EngineControlTool\CreateExcel.cs 1

```
1  ///
2  /// CreateExcel.cs
3  ///     Class that creates Excel spreadsheets of recorded values
4  ///
5  /// Written by Mary Lichtenwalner
6  ///
7  /// Last Update: April 5, 2022
8  ///
9
10 using System;
11 using System.Collections.Generic;
12 using System.Linq;
13 using Excel = Microsoft.Office.Interop.Excel;
14
15 namespace EngineControlTool
16 {
17     public class CreateExcel
18     {
19         // Initialize the viewModel
20         public ViewModel viewModel;
21
22         // Initialize the Excel application, workbook, and spreadsheets
23         Excel.Application xlApp;
24         Excel.Workbook xlBook;
25         Excel.Worksheet xlSheet1;
26         Excel.Worksheet xlSheet2;
27
28         //
             ************************************************************************
             ******************
29         // Method 1. Create Excel constructor. Pass in viewModel from
             MainWindow.xaml.cs
30         //
             ************************************************************************
             ******************
31         public CreateExcel(ViewModel vm)
32         {
33             viewModel = vm;
34         }
35
36         //
             ************************************************************************
             ******************
37         // Method 2. Start up Excel application and insert desired number of
             spreadsheets
38         //
             ************************************************************************
             ******************
39         public void startUpExcel()
40         {
41             // Start Excel and input two spreadsheets
42             xlApp = new Excel.Application();
```

...l Tool\EngineControlTool\EngineControlTool\CreateExcel.cs 2

```
43              xlApp.Visible = true;
44              xlApp.DisplayAlerts = false;
45              xlBook = xlApp.Workbooks.Add();
46              xlSheet1 = xlBook.ActiveSheet;
47              xlSheet2 = xlBook.Worksheets.Add();
48
49              // Name the spreadsheets
50              xlSheet1.Name = "Metric";
51              xlSheet2.Name = "Measured Values";
52
53          }
54
55          //
              ************************************************************************
              *****************
56          // Method 3. Generate Excel spreadsheet with metric values (Celsius and
              pascals)
57          //
              ************************************************************************
              *****************
58          public void generateMetricExcel(List<string> timeList, List<double>
              tCompressed, List<double> tChamber, List<double> tExhaust,
59              List<double> pAmbient, List<double> pCompressed, List<double>
                  tAmbient, List<double> humidity, List<double> shaftSpeed,
60              List<string> logList)
61          {
62              // Insert desired headers
63              xlSheet1.Range["C1", "F1"].Merge();
64              xlSheet1.Range["G1", "H1"].Merge();
65              xlSheet1.Range["A1", "B1"].Merge();
66              xlSheet1.Cells[1, 1] = "Time Stamps";
67              xlSheet1.Cells[1, 3] = "Temperature (ºC)";
68              xlSheet1.Cells[1, 7] = "Pressure (Pa)";
69              xlSheet1.Cells[1, 9] = "Humidity (%)";
70              xlSheet1.Cells[1, 10] = "Shaft Speed (RPM)";
71              xlSheet1.Cells[1, 11] = "Flow Rate (idk)";
72              xlSheet1.Cells[2, 1] = "Time";
73              xlSheet1.Cells[2, 2] = "Log";
74              xlSheet1.Cells[2, 3] = "Ambient";
75              xlSheet1.Cells[2, 4] = "Compressed";
76              xlSheet1.Cells[2, 5] = "Chamber";
77              xlSheet1.Cells[2, 6] = "Exhaust";
78              xlSheet1.Cells[2, 7] = "Ambient";
79              xlSheet1.Cells[2, 8] = "Compressed";
80              xlSheet1.Cells[2, 9] = "Ambient";
81              xlSheet1.Cells[2, 10] = "Turbine";
82              xlSheet1.Cells[2, 11] = "Exhaust";
83
84              // Insert the data from the viewModel lists into the correct column
85              // Convert temperature readings from Fahrenheit to Celsius
86              // Convert pressure readings from psi to pascals
87              for (int i = 0; i < timeList.Count(); i++ )
```

...l Tool\EngineControlTool\EngineControlTool\CreateExcel.cs 3

```
 88              {
 89                  xlSheet1.Cells[i + 3, 1] = timeList[i];
 90                  xlSheet1.Cells[i + 3, 2] = logList[i];
 91                  xlSheet1.Cells[i + 3, 4] = (tCompressed[i] - 32) * 0.5556;
 92                  xlSheet1.Cells[i + 3, 5] = (tChamber[i] - 32) * 0.5556;
 93                  xlSheet1.Cells[i + 3, 6] = (tExhaust[i] - 32) * 0.5556;
 94                  xlSheet1.Cells[i + 3, 7] = pAmbient[i] * 6894.76;
 95                  xlSheet1.Cells[i + 3, 8] = pCompressed[i] * 6894.76;
 96                  xlSheet1.Cells[i + 3, 3] = (tAmbient[i] - 32) * 0.5556;
 97                  xlSheet1.Cells[i + 3, 9] = humidity[i];
 98                  xlSheet1.Cells[i + 3, 10] = shaftSpeed[i];
 99              }
100
101             // Just insert flow rate into first time stamp
102             xlSheet1.Cells[3, 11] = viewModel.flowRate;
103
104             //                                                                ➥
                  ****************************************************************➥
                  *****************
105             // Style the sheet
106             //                                                                ➥
                  ****************************************************************➥
                  *****************
107
108             // Set up ranges that will be used
109             int count = timeList.Count() + 2;
110             Excel.Range headers = xlSheet1.Range[xlSheet1.Cells[1, 1],         ➥
                  xlSheet1.Cells[1, 11]];
111             Excel.Range headers2 = xlSheet1.Range[xlSheet1.Cells[2, 1],        ➥
                  xlSheet1.Cells[2, 11]];
112             Excel.Range timeCol = xlSheet1.Range[xlSheet1.Cells[3, 1],         ➥
                  xlSheet1.Cells[count, 1]];
113             Excel.Range tAmbientCol = xlSheet1.Range[xlSheet1.Cells[3, 3],     ➥
                  xlSheet1.Cells[count, 3]];
114             Excel.Range tCompressedCol = xlSheet1.Range[xlSheet1.Cells[3, 4],  ➥
                  xlSheet1.Cells[count, 4]];
115             Excel.Range tChamberCol = xlSheet1.Range[xlSheet1.Cells[3, 5],     ➥
                  xlSheet1.Cells[count, 5]];
116             Excel.Range tExhaustCol = xlSheet1.Range[xlSheet1.Cells[3, 6],     ➥
                  xlSheet1.Cells[count, 6]];
117             Excel.Range pAmbientCol = xlSheet1.Range[xlSheet1.Cells[3, 7],     ➥
                  xlSheet1.Cells[count, 7]];
118             Excel.Range pCompressedCol = xlSheet1.Range[xlSheet1.Cells[3, 8],  ➥
                  xlSheet1.Cells[count, 8]];
119             Excel.Range humidityCol = xlSheet1.Range[xlSheet1.Cells[3, 9],     ➥
                  xlSheet1.Cells[count, 9]];
120             Excel.Range shaftCol = xlSheet1.Range[xlSheet1.Cells[3, 10],       ➥
                  xlSheet1.Cells[count, 10]];
121             Excel.Range flowCol = xlSheet1.Range[xlSheet1.Cells[3, 11],        ➥
                  xlSheet1.Cells[count, 11]];
122             Excel.Range logCol = xlSheet1.Range[xlSheet1.Cells[3, 2],          ➥
                  xlSheet1.Cells[count, 2]];
```

...l Tool\EngineControlTool\EngineControlTool\CreateExcel.cs 4

```
123
124             Excel.Range tempRange = xlSheet1.Range[xlSheet1.Cells[3, 3],
                  xlSheet1.Cells[count, 6]];
125             Excel.Range pressureRange = xlSheet1.Range[xlSheet1.Cells[3, 7],
                  xlSheet1.Cells[count, 8]];
126
127             // Turn on desired borders
128             timeCol.Borders.LineStyle = Excel.XlLineStyle.xlContinuous;
129             tempRange.Borders.LineStyle = Excel.XlLineStyle.xlContinuous;
130             pressureRange.Borders.LineStyle = Excel.XlLineStyle.xlContinuous;
131             humidityCol.Borders.LineStyle = Excel.XlLineStyle.xlContinuous;
132             shaftCol.Borders.LineStyle = Excel.XlLineStyle.xlContinuous;
133             flowCol.Borders.LineStyle = Excel.XlLineStyle.xlContinuous;
134             logCol.Borders.LineStyle = Excel.XlLineStyle.xlContinuous;
135
136             headers.Borders.LineStyle = Excel.XlLineStyle.xlContinuous;
137             headers.Borders.Weight = Excel.XlBorderWeight.xlThick;
138             headers2.Borders.LineStyle = Excel.XlLineStyle.xlContinuous;
139             headers2.Borders.Weight = Excel.XlBorderWeight.xlThick;
140
141             DrawThickBorderAround(timeCol);
142             DrawThickBorderAround(logCol);
143             DrawThickBorderAround(tAmbientCol);
144             DrawThickBorderAround(tCompressedCol);
145             DrawThickBorderAround(tChamberCol);
146             DrawThickBorderAround(tExhaustCol);
147             DrawThickBorderAround(pAmbientCol);
148             DrawThickBorderAround(pCompressedCol);
149             DrawThickBorderAround(humidityCol);
150             DrawThickBorderAround(shaftCol);
151             DrawThickBorderAround(flowCol);
152
153             // Set different background colors
154             xlSheet1.Range["C1", "F2"].Interior.Color = 0xe7c6b4;
155             tempRange.Interior.Color = 0xf2e1d9;
156             xlSheet1.Range["G1", "H2"].Interior.Color = 0xadcbf8;
157             xlSheet1.Range["I1", "I2"].Interior.Color = 0x99e6ff;
158             xlSheet1.Range["J1", "J2"].Interior.Color = 0xb4e0c6;
159             xlSheet1.Range["K1", "K2"].Interior.Color = 0xe597b8;
160             xlSheet1.Range["A1", "B2"].Interior.Color = 0xa9a9a9;
161             timeCol.Interior.Color = 0xd9d9d9;
162             logCol.Interior.Color = 0xd9d9d9;
163             pressureRange.Interior.Color = 0xd6e4fc;
164             humidityCol.Interior.Color = 0xccf2ff;
165             shaftCol.Interior.Color = 0xdaefe2;
166             flowCol.Interior.Color = 0xf3cde2;
167
168             // Bold fonts
169             headers.Font.Bold = true;
170             headers2.Font.Bold = true;
171
172             // Miscellaneous: autofit columns, date/time format, center text
```

...l Tool\EngineControlTool\EngineControlTool\CreateExcel.cs 5

```
173             timeCol.NumberFormat = "mm/dd/yy  hh:mm:ss";
174             xlSheet1.Columns.AutoFit();
175             xlSheet1.Cells.HorizontalAlignment = Excel.XlHAlign.xlHAlignCenter;
176         }
177
178         //
              ********************************************************************
              ******************
179         // Method 4. Generate Excel spreadsheet with measured values (Fahrenheit
              and psi)
180         //
              ********************************************************************
              ******************
181         public void generateExcel(List<string> timeList, List<double>
              tCompressed, List<double> tChamber, List<double> tExhaust,
182             List<double> pAmbient, List<double> pCompressed, List<double>
                tAmbient, List<double> humidity, List<double> shaftSpeed,
183             List<string> logList)
184         {
185             // Insert desired headers
186             xlSheet2.Range["C1", "F1"].Merge();
187             xlSheet2.Range["G1", "H1"].Merge();
188             xlSheet2.Range["A1", "B1"].Merge();
189             xlSheet2.Cells[1, 1] = "Time Stamps";
190             xlSheet2.Cells[1, 3] = "Temperature (ºF)";
191             xlSheet2.Cells[1, 7] = "Pressure (psi)";
192             xlSheet2.Cells[1, 9] = "Humidity (%)";
193             xlSheet2.Cells[1, 10] = "Shaft Speed (RPM)";
194             xlSheet2.Cells[1, 11] = "Flow Rate (idk)";
195             xlSheet2.Cells[2, 1] = "Time";
196             xlSheet2.Cells[2, 2] = "Log";
197             xlSheet2.Cells[2, 3] = "Ambient";
198             xlSheet2.Cells[2, 4] = "Compressed";
199             xlSheet2.Cells[2, 5] = "Chamber";
200             xlSheet2.Cells[2, 6] = "Exhaust";
201             xlSheet2.Cells[2, 7] = "Ambient";
202             xlSheet2.Cells[2, 8] = "Compressed";
203             xlSheet2.Cells[2, 9] = "Ambient";
204             xlSheet2.Cells[2, 10] = "Turbine";
205             xlSheet2.Cells[2, 11] = "Exhaust";
206
207             // Insert the data from viewModel lists into the correct column
208             for (int i = 0; i < timeList.Count(); i++)
209             {
210                 xlSheet2.Cells[i + 3, 1] = timeList[i];
211                 xlSheet2.Cells[i + 3, 2] = logList[i];
212                 xlSheet2.Cells[i + 3, 4] = tCompressed[i];
213                 xlSheet2.Cells[i + 3, 5] = tChamber[i];
214                 xlSheet2.Cells[i + 3, 6] = tExhaust[i];
215                 xlSheet2.Cells[i + 3, 7] = pAmbient[i];
216                 xlSheet2.Cells[i + 3, 8] = pCompressed[i];
217                 xlSheet2.Cells[i + 3, 3] = tAmbient[i];
```

...l Tool\EngineControlTool\EngineControlTool\CreateExcel.cs 6

```
218                 xlSheet2.Cells[i + 3, 9] = humidity[i];
219                 xlSheet2.Cells[i + 3, 10] = shaftSpeed[i];
220             }
221
222             // Just insert flow rate into first time stamp
223             xlSheet2.Cells[3, 11] = viewModel.flowRate;
224
225             //
                  ********************************************************************
                  *****************
226             // Style the sheet
227             //
                  ********************************************************************
                  *****************
228
229             // Set up ranges that will be used
230             int count = timeList.Count() + 2;
231             Excel.Range headers = xlSheet2.Range[xlSheet2.Cells[1, 1],
                  xlSheet2.Cells[1, 11]];
232             Excel.Range headers2 = xlSheet2.Range[xlSheet2.Cells[2, 1],
                  xlSheet2.Cells[2, 11]];
233             Excel.Range timeCol = xlSheet2.Range[xlSheet2.Cells[3, 1],
                  xlSheet2.Cells[count, 1]];
234             Excel.Range tAmbientCol = xlSheet2.Range[xlSheet2.Cells[3, 3],
                  xlSheet2.Cells[count, 3]];
235             Excel.Range tCompressedCol = xlSheet2.Range[xlSheet2.Cells[3, 4],
                  xlSheet2.Cells[count, 4]];
236             Excel.Range tChamberCol = xlSheet2.Range[xlSheet2.Cells[3, 5],
                  xlSheet2.Cells[count, 5]];
237             Excel.Range tExhaustCol = xlSheet2.Range[xlSheet2.Cells[3, 6],
                  xlSheet2.Cells[count, 6]];
238             Excel.Range pAmbientCol = xlSheet2.Range[xlSheet2.Cells[3, 7],
                  xlSheet2.Cells[count, 7]];
239             Excel.Range pCompressedCol = xlSheet2.Range[xlSheet2.Cells[3, 8],
                  xlSheet2.Cells[count, 8]];
240             Excel.Range humidityCol = xlSheet2.Range[xlSheet2.Cells[3, 9],
                  xlSheet2.Cells[count, 9]];
241             Excel.Range shaftCol = xlSheet2.Range[xlSheet2.Cells[3, 10],
                  xlSheet2.Cells[count, 10]];
242             Excel.Range flowCol = xlSheet2.Range[xlSheet2.Cells[3, 11],
                  xlSheet2.Cells[count, 11]];
243             Excel.Range logCol = xlSheet2.Range[xlSheet2.Cells[3, 2],
                  xlSheet2.Cells[count, 2]];
244
245             Excel.Range tempRange = xlSheet2.Range[xlSheet2.Cells[3, 3],
                  xlSheet2.Cells[count, 6]];
246             Excel.Range pressureRange = xlSheet2.Range[xlSheet2.Cells[3, 7],
                  xlSheet2.Cells[count, 8]];
247
248             // Turn on desired borders
249             timeCol.Borders.LineStyle = Excel.XlLineStyle.xlContinuous;
250             tempRange.Borders.LineStyle = Excel.XlLineStyle.xlContinuous;
```

...l Tool\EngineControlTool\EngineControlTool\CreateExcel.cs 7

```
251             pressureRange.Borders.LineStyle = Excel.XlLineStyle.xlContinuous;
252             humidityCol.Borders.LineStyle = Excel.XlLineStyle.xlContinuous;
253             shaftCol.Borders.LineStyle = Excel.XlLineStyle.xlContinuous;
254             flowCol.Borders.LineStyle = Excel.XlLineStyle.xlContinuous;
255             logCol.Borders.LineStyle = Excel.XlLineStyle.xlContinuous;
256
257             headers.Borders.LineStyle = Excel.XlLineStyle.xlContinuous;
258             headers.Borders.Weight = Excel.XlBorderWeight.xlThick;
259             headers2.Borders.LineStyle = Excel.XlLineStyle.xlContinuous;
260             headers2.Borders.Weight = Excel.XlBorderWeight.xlThick;
261
262             DrawThickBorderAround(timeCol);
263             DrawThickBorderAround(logCol);
264             DrawThickBorderAround(tAmbientCol);
265             DrawThickBorderAround(tCompressedCol);
266             DrawThickBorderAround(tChamberCol);
267             DrawThickBorderAround(tExhaustCol);
268             DrawThickBorderAround(pAmbientCol);
269             DrawThickBorderAround(pCompressedCol);
270             DrawThickBorderAround(humidityCol);
271             DrawThickBorderAround(shaftCol);
272             DrawThickBorderAround(flowCol);
273
274             // Set different background colors
275             xlSheet2.Range["C1", "F2"].Interior.Color = 0xe7c6b4;
276             tempRange.Interior.Color = 0xf2e1d9;
277             xlSheet2.Range["G1", "H2"].Interior.Color = 0xadcbf8;
278             xlSheet2.Range["I1", "I2"].Interior.Color = 0x99e6ff;
279             xlSheet2.Range["J1", "J2"].Interior.Color = 0xb4e0c6;
280             xlSheet2.Range["K1", "K2"].Interior.Color = 0xe597b8;
281             xlSheet2.Range["A1", "B2"].Interior.Color = 0xa9a9a9;
282             timeCol.Interior.Color = 0xd9d9d9;
283             logCol.Interior.Color = 0xd9d9d9;
284             pressureRange.Interior.Color = 0xd6e4fc;
285             humidityCol.Interior.Color = 0xccf2ff;
286             shaftCol.Interior.Color = 0xdaefe2;
287             flowCol.Interior.Color = 0xf3cde2;
288
289             // Bold fonts
290             headers.Font.Bold = true;
291             headers2.Font.Bold = true;
292
293             // Miscellaneous: autofit columns, date/time format, center text
294             timeCol.NumberFormat = "mm/dd/yy  hh:mm:ss";
295             xlSheet2.Columns.AutoFit();
296             xlSheet2.Cells.HorizontalAlignment = Excel.XlHAlign.xlHAlignCenter;
297         }
298
299         //
              ************************************************************************
              *****************
300         // Method 5. Draw a thick border around some Excel range
```

...l Tool\EngineControlTool\EngineControlTool\CreateExcel.cs 8

```
301         //                                                                  ➥
              ********************************************************************* ➥
              *****************
302         private void DrawThickBorderAround(Excel.Range cells)
303         {
304             cells.BorderAround(Type.Missing, Excel.XlBorderWeight.xlThick,  ➥
                  Excel.XlColorIndex.xlColorIndexAutomatic, Type.Missing);
305         }
306     }
307 }
308
```

...rol Tool\EngineControlTool\EngineControlTool\ViewModel.cs 1

```
1  ///
2  /// ViewModel.cs
3  ///     Class that passes data between other classes as necessary
4  ///
5  /// Written by Mary Lichtenwalner
6  ///
7  /// Last Update: April 10, 2022
8  ///
9
10 using System.Collections.Generic;
11 using System.ComponentModel;
12 using System.Runtime.CompilerServices;
13
14 namespace EngineControlTool
15 {
16     public class ViewModel : INotifyPropertyChanged
17     {
18         //
           ************************************************************************
           ******************
19         // Section 1. Methods to control changes in data
20         //
           ************************************************************************
           ******************
21         public ViewModel MainModel { get; internal set; }
22
23         public event PropertyChangedEventHandler PropertyChanged;
24
25         public void InvokePropertyChanged(PropertyChangedEventArgs e)
26         {
27             PropertyChangedEventHandler handler = PropertyChanged;
28             if (handler != null) handler(this, e);
29         }
30
31         protected void onPropertyChanged(string propertyName)
32         {
33             PropertyChanged?.Invoke(this, new PropertyChangedEventArgs
                 (propertyName));
34         }
35
36         public void onPropertyChange([CallerMemberName] string propertyName =
             null)
37         {
38             PropertyChanged?.Invoke(this, new PropertyChangedEventArgs
                 (propertyName));
39         }
40
41         //
           ************************************************************************
           *************
42         // Section 2. Methods to control which color is selected in GUI
43         //
```

...rol Tool\EngineControlTool\EngineControlTool\ViewModel.cs 2

```
            ************************************************************************
            ************

        // Integer to set dark or light theme
        private int color_theme = 0;
        public int Color_Theme
        {
            get { return color_theme; }
            set { color_theme = value; onPropertyChange(nameof(Color_Theme)); }
        }

        // Integer that determines which accent color is selected
        private int accent_color = 0;
        public int Accent_Color
        {
            get { return accent_color; }
            set { accent_color = value; onPropertyChange(nameof(Accent_Color)); }
        }

        // String that determines which accent color is selected
        private string accent_color_string = "Red";
        public string Accent_Color_String
        {
            get { return accent_color_string; }
            set { accent_color_string = value; onPropertyChange(nameof
              (Accent_Color_String)); }
        }

        //
          ************************************************************************
          ************
        // Section 3. All real-time measurements read from serial port
        //
          ************************************************************************
          ************

        // Ambient pressure reading
        private double _pressureAmbient;
        public double pressureAmbient
        {
            get { return _pressureAmbient; }
            set{ _pressureAmbient = value; onPropertyChange(nameof
              (pressureAmbient)); }

        }

        // Compressed pressure reading
        private double _pressureCompressed;
        public double pressureCompressed
        {
            get { return _pressureCompressed; }
            set { _pressureCompressed = value; onPropertyChange(nameof
```

...rol Tool\EngineControlTool\EngineControlTool\ViewModel.cs 3

```
                (pressureCompressed)); }
88
89          }
90
91          //Ambient temperature reading
92          private double _tempAmbient;
93          public double tempAmbient
94          {
95              get { return _tempAmbient; }
96              set { _tempAmbient = value; onPropertyChange(nameof(tempAmbient)); }
97
98          }
99
100         //Compressed temperature reading
101         private double _tempCompressed;
102         public double tempCompressed
103         {
104             get { return _tempCompressed; }
105             set { _tempCompressed = value; onPropertyChange(nameof
                  (tempCompressed)); }
106
107         }
108
109         //Chamber temperature reading
110         private double _tempChamber;
111         public double tempChamber
112         {
113             get { return _tempChamber; }
114             set { _tempChamber = value; onPropertyChange(nameof(tempChamber)); }
115
116         }
117
118         //Exhaust temperature reading
119         private double _tempExhaust;
120         public double tempExhaust
121         {
122             get { return _tempExhaust; }
123             set { _tempExhaust = value; onPropertyChange(nameof(tempExhaust)); }
124
125         }
126
127         //Ambient humidity reading
128         private double _humidity;
129         public double humidity
130         {
131             get { return _humidity; }
132             set { _humidity = value; onPropertyChange(nameof(humidity)); }
133         }
134
135         //Shaft speed reading
136         private double _shaftSpeed;
137         public double shaftSpeed
```

...rol Tool\EngineControlTool\EngineControlTool\ViewModel.cs 4

```
138         {
139             get { return _shaftSpeed; }
140             set { _shaftSpeed = value; onPropertyChange(nameof(shaftSpeed)); }
141         }
142
143         //Flow rate from user input
144         private double _flowRate;
145         public double flowRate
146         {
147             get { return _flowRate; }
148             set { _flowRate = value; onPropertyChange(nameof(flowRate)); }
149         }
150
151         // Bool determining if log button was clicked by user
152         private bool _log;
153         public bool Log
154         {
155             get { return _log; }
156             set { _log = value; onPropertyChange(nameof(Log)); }
157         }
158
159         // Optional message to log inputted by user
160         private string _logNote;
161         public string logNote
162         {
163             get { return _logNote; }
164             set { _logNote = value; onPropertyChange(nameof(logNote)); }
165         }
166
167         //
              ********************************************************************
              ************
168         // Section 4. Lists that hold all the data that has been received
169         //
              ********************************************************************
              ************
170
171         // Ambient pressure list
172         private List<double> _pAmbientList = new List<double>();
173         public List<double> pAmbientList
174         {
175             get { return _pAmbientList; }
176             set { _pAmbientList = value; onPropertyChange(nameof
                  (pAmbientList)); }
177
178         }
179
180         // Compressed pressure list
181         private List<double> _pCompressedList = new List<double>();
182         public List<double> pCompressedList
183         {
184             get { return _pCompressedList; }
```

...rol Tool\EngineControlTool\EngineControlTool\ViewModel.cs 5

```
185             set { _pCompressedList = value; onPropertyChange(nameof
                  (pCompressedList)); }
186
187         }
188
189         // Ambient temperature list
190         private List<double> _tAmbientList = new List<double>();
191         public List<double> tAmbientList
192         {
193             get { return _tAmbientList; }
194             set { _tAmbientList = value; onPropertyChange(nameof
                  (tAmbientList)); }
195
196         }
197
198         // Compressed temperature list
199         private List<double> _tCompressedList = new List<double>();
200         public List<double> tCompressedList
201         {
202             get { return _tCompressedList; }
203             set { _tCompressedList = value; onPropertyChange(nameof
                  (tCompressedList)); }
204
205         }
206
207         // Chamber temperature list
208         private List<double> _tChamberList = new List<double>();
209         public List<double> tChamberList
210         {
211             get { return _tChamberList; }
212             set { _tChamberList = value; onPropertyChange(nameof
                  (tChamberList)); }
213
214         }
215
216         // Exhaust temperature list
217         private List<double> _tExhaustList = new List<double>();
218         public List<double> tExhaustList
219         {
220             get { return _tExhaustList; }
221             set { _tExhaustList = value; onPropertyChange(nameof
                  (tExhaustList)); }
222
223         }
224
225         // Ambient humidity list
226         private List<double> _humidityList = new List<double>();
227         public List<double> humidityList
228         {
229             get { return _humidityList; }
230             set { _humidityList = value; onPropertyChange(nameof
                  (humidityList)); }
```

...rol Tool\EngineControlTool\EngineControlTool\ViewModel.cs 6

```
231
232         }
233
234         // Shaft speed list
235         private List<double> _shaftSpeedList = new List<double>();
236         public List<double> shaftSpeedList
237         {
238             get { return _shaftSpeedList; }
239             set { _shaftSpeedList = value; onPropertyChange(nameof
                  (shaftSpeedList)); }
240
241         }
242
243         // List with all time stamps
244         private List<string> _timeList = new List<string>();
245         public List<string> timeList
246         {
247             get { return _timeList; }
248             set { _timeList = value; onPropertyChange(nameof(timeList)); }
249         }
250
251         // List containing the log messages/instances of log button clicked
252         private List<string> _logList = new List<string>();
253         public List<string> logList
254         {
255             get { return _logList; }
256             set { _logList = value; onPropertyChange(nameof(logList)); }
257         }
258
259         //
            ************************************************************************
            ************
260         // Section 5. Values used to control warning label
261         //
            ************************************************************************
            ************
262
263         // Boolean showing if dangerous values are being recorded
264         private bool _isDangerous = false;
265         public bool isDangerous
266         {
267             get { return _isDangerous; }
268             set { _isDangerous = value; onPropertyChange(nameof(isDangerous)); }
269         }
270
271         // Integer determining if warning label is displayed
272         private int _showWarning = 0;
273         public int showWarning
274         {
275             get { return _showWarning; }
276             set { _showWarning = value; onPropertyChange(nameof(showWarning)); }
277         }
```

...rol Tool\EngineControlTool\EngineControlTool\ViewModel.cs 7

```
278
279         // Integer that designates if it is a temperature or a pressure that is
              too high
280         private int _tempOrPressure;
281         public int tempOrPressure
282         {
283             get { return _tempOrPressure; }
284             set { _tempOrPressure = value; onPropertyChange(nameof
                  (tempOrPressure)); }
285         }
286     }
287 }
288
```

Appendix H: Ideas for Future Projects

The following outlines some ideas for future projects involving the Mercer University gas turbine engine.

H.1 Sustainable Combustion

SES measured the pressure ratio between state 1 and state 2 (ambient air pressure and compressed air pressure) to be 0.95. This suggests that no work is being done by the compressor on the air flow. The expected pressure ratio should be in the range of 2 to 3. This could be caused by improper pressure gauge readings; however, testing performed on the pressure gauges suggests that the gauges are working as intended.

Future senior design teams could investigate this issue further to determine if there is an error in sensor measurement and implementation, or they could investigate why no work is being done on the air by the compressor.

Additionally, the combustion chamber is oversized for the current air flow coming through the engine. Because of the size of the chamber, unwanted turbulence could have adverse effects on the flame generated by the combustion of propane. A smaller combustion chamber would reduce the amount of turbulence and give students more experience using high-level fluid dynamics principles.

H.2 Nozzle the Exhaust

Assuming the gas turbine engine can sustain stable combustion, students could add a nozzle to the exhaust of the engine to generate thrust. Students could add a live anemometer reading to the exhaust to measure the velocity of the air exiting the engine. Using this value, students could measure the work done on the air by the engine.

H.3 Starter Mechanism

Future senior design teams could design a starting mechanism that spools up the compressor to high revolutions per minute before ignition. A starter from a car could be placed onto a gear train to achieve a higher gear ratio. This starter could reduce the amount of time required for the turbocharger assembly to reach a shaft speed high enough to sustain stable combustion.

H.4 Sensor Array Changes

Currently, the sensor array installed is a temporary setup. The Arduino is connected to the sensors through a breadboard, so it is possible to move sensors and reconfigure the connections. In the future, a senior design team could solder all these connections to make the sensor array more permanent. A future project could also install a better fitting thermocouple in the combustion chamber to fix the problem of grounding between the combustion chamber thermocouple and exhaust thermocouple.

Other features could be added to the Engine Control Tool as well. For example, once the engine can maintain stable combustion, Engine Control Tool could be programmed to automatically calculate net work and efficiency while recording data.

Appendix I: Resumes

Mary Lichtenwalner

(678) 313-8958 • Cumming, GA 30041
maryelichtenwalner@gmail.com
www.linkedin.com/in/marylichtenwalner

Professional Goal

Gain experience and contribute to the development of the gas turbine engine through the senior design process

Education

Bachelor of Science in Engineering (Electrical) & Bachelor of Science in Mathematics *Class of 2022*
Mercer University – GPA 4.0

Relevant Coursework
Mathematical Modeling, Electronics, Linear Algebra, Electromagnetic Applications, Electronics Laboratory

Professional Qualities

- Knowledge of C#, C++, MATLAB, LaTeX, Visual Studio, Microsoft Office suite
- Hard-working and self-motivated, thrives in a challenging work environment
- Passionate about learning new skills and technologies

Experience

Mercer Engineering Research Center – Warner Robins, GA *May 2020 – Present*
Electronic Warfare Intern
- Completed projects both individually and as part of a team
- Learned fundamental radar concepts, radar signal processing, and the basics of electronic warfare

Mercer Math Department – Macon, GA *August 2020 – May 2021*
LaTeX Assistant
- Worked with a math professor to prepare LaTeX documents for class

Projects

Electronic Warfare Intern
MATLAB Matched Filter
- Used MATLAB to simulate BPSK and LFM radar signals
- Programmed a matched filter to search for the transmitted signal in the simulated received signal
- Added jamming signals and observed changes in results

Amplifier Characterization Tool
- Wrote a C# software tool to automate mapping of amplifier gain profiles
- Interfaced with RF measurement equipment

Instrument Checkout Tool
- Created a C# program to automate the testing of new equipment
- Program records results of each test to document the status of new equipment

Radar Antenna Testing
- Scheduled and performed gain tests on radar antennas
- Determined a method for taking precise measurements of antenna angles
- Collaborated with a team to present results

Latex Assistant
Calculus II Beamer Presentations
- Generated Beamer presentations using LaTeX for a calculus II class and discrete mathematics class

Honors and Awards

Mercer University President's List *August 2018 – Present*
Tau Beta Pi Engineering Honor Society *November 2020 – Present*
Mercer University Presidential Scholarship *August 2018 – Present*

Barrett McDonald

Pooler, GA • barrettmc77@gmail.com • (912) 658-9123 • https://www.linkedin.com/in/barrett-mcdonald/

EDUCATION

Mercer University | Macon, GA | May 2022
Bachelor of Science in Engineering (B.S.E.) – Mechanical Engineering
Cumulative GPA: 4.0/4.0, Major GPA: 4.0/4.0
President's List All Semesters

Relevant Coursework: Solid Mechanics 2 · Fluid Mechanics · Heat Transfer · Finite Element Analysis · Machine Design

ENGINEERING PROJECTS

Jet Engine – Senior Design Project | *April 2021 - Present*
- Collaborating with a team of two other engineering students to fabricate a jet engine using a turbo assembly from a bulldozer engine
- Rebuilt the previous design using extra parts found in the Mercer University machine shop in less than 6 weeks
- Project Goal: redesign the existing combustion chamber to facilitate stable combustion and increase efficiency

Other Relevant Class Projects
- Used C++ and Numerical Methods to create a program to estimate roots of equations and model heat transfer
- Designed a gear box using shaft analysis techniques from Machine Design that will transfer power to a generator from a wind turbine

EXPERIENCE

***Student Researcher,* Mercer University | Macon, GA** | *January 2020 – Present*
- Researched the uptake of carbon dioxide into water over time using a modified mechanical balance
- Designed and fabricated a test bench that will measure a solid rocket motor's force output versus time
- Measured material properties of polymers before and after metallic coatings applied using sputtering deposition

***Product Engineering Intern,* Ridge Corporation | Pataskala, OH** | *June 2020 – August 2020*
- Led initial research and development efforts for polymer-based ballistic protection
- Designed prototype composite layups and created bills of materials using SolidWorks
- Used Six Sigma principles to assist in the creation of new manufacturing processes
- Performed ASTM and NIJ materials tests
- Generated PowerPoints and progress reports

***Process Engineering Intern,* Peterbilt Motors Company | Denton, TX** | May 2021 – Present
- Designed and Fabricated tooling used every day during production
- Performed and Analyzed time study data to make informed decisions about line balancing
- Worked with the Continuous Improvement group to conduct various Six Sigma projects
- Presented time study results to the plant management team and offered potential solutions that will allow for 25 additional trucks to be built per shift

LEADERSHIP ACTIVITIES

***President,* Kappa Alpha Order** | *November 2020 - Present*
- Introduced updated risk management policies
- Filled IRS 990 tax forms for expenses totaling just below $40,000
- Introduced new community involvement initiatives through the local animal shelter

***Vice President,* Order of Omega Honors Society** | *April 2020 – April 2021*
- Plan Greek Letter Organization Week to promote the Greek Life community on campus
- Manage committees and perform all functions of the President in her absence

ADDITIONAL INFORMATION

Skills: Problem Solving, Composite Design, Composite Manufacturing, SolidWorks, Creo, MATLAB, ANSYS, Microsoft Office Suite, Six Sigma Green Belt, Machining and Fabrication, MIG Welding, Technical Communication, Leadership
Volunteering: Raised over $4,000 for the Muscular Dystrophy Association and over $5,000 for the Independence Fund
Honors Societies: Tau Beta Pi, Order of Omega, Phi Eta Sigma, Phi Kappa Phi

References

Dr. Matt Marone | marone_mj@mercer.edu | (478) 301-2597 – Associate Professor, Mercer University
Bret Moss | bret.moss@ridgecorp.com | (614) 421-7434 - Director of Product Engineering, Ridge Corporation
Casey McAden | casey.mcaden@paccar.com | (940) 205-8440 – Process Engineer, Peterbilt Motors Company

Ryan McMillan

mcmillanryan55@gmail.com | (309)256-8571 |6037 East Agave Circle Cave Creek, AZ 85331

EDUCATION

MERCER UNIVERSITY | School of Engineering — **Cumulative GPA: 3.9/4.0**
Major: Bachelor of Science in Mechanical Engineering — Expected Graduation: May 2022
Minor: Physics
Scholarship: Tuition Exchange
Honors: Presidents List (Spring 2020), Dean's List (Fall 2018, Spring 2019, Fall 2019, Fall 2020, Spring 2021)

PROFESSIONAL EXPERIENCE

Caterpillar Inc. | *Co-Op Parallel intern– Engineering Integrated Component Solution – Aftertreatment* | **May 2021 – Present**

- Worked directly with lead design engineers to design and model new caterpillar emissions module (CEM) for prototype builds based on EPA and customer requirements
- Negotiated with 4 separate suppliers on optimizing designs for manufacturability and cost of components to be used on new developing CEM's
- Developed/reviewed 100+ engineering drawings of CEM components for suppliers and submitted purchase orders for prototype builds

Caterpillar Inc. | *Corporate Intern – Engineering Large Power Systems Growth – Cooling Division* | **June 2020 – May 2021**

- Authored a design guide for modular radiator system through analyzing models and testing data on radiator designs as well as collaborating with the design teams on their strategies
- Developed an appearance specification for a cooling cores supplier by closely collaborating with supplier and Caterpillar's internal cooling products team on acceptable standards of the product
- Developed and managed DFMEA's for radiator, ATAAC, condenser, fuel cooler, oil cooler, and fan systems to correctly record and prioritize design risks within the cooling modules

RESEARCH EXPERIENCE

Mercer University, Mechanical Engineering Department | *Research Assistant* | **January 2021 – May 2021**

- Investigated the effectiveness of coating acrylonitrile-butadiene-styrene (ABS) structures with silver for maintaining the specimen's mechanical properties after long exposure to harsh environments for Dr. Dorina Mihut
- Aged the 3-D printed ABS specimen using a Q-Lab QUV Accelerated Weathering Tester and then tested the mechanical properties maintained by those specimens with a Mark-10 ESM303 Tensile Compression Force Tester
- Presented findings to around 500 people at Mercer University's annual Bear Fair and submitted my research poster to Society of Vacuum Coaters while participating in their annual conference

LEADERSHIP & ACTIVITIES

American Society of Mechanical Engineers | *Collegiate Member and Vice President* | **August 2018 – August 2019**

- Organized campus fundraising events, including a booth to raise financial support at the university football games
- Elected as a freshman to serve as VP by giving a speech to collegiate members of the organization displaying my values of teamwork and passion for community involvement
- Networked with multiple companies on a yearly basis for potential tours and visits to cultivate the club members understanding of the mechanical engineering field and emerge members with real world shadowing opportunities

Sigma Alpha Epsilon | *Vice President and Philanthropy Chair* | **August 2019 – Present**

- Maintained and managed collegiate members to increase Fraternity productivity by tracking and sending out updates of accomplishments, fundraising, and current activities to 375+ contacts using Mailchimp
- Worked alongside another campus organization in establishing a golf fundraiser for Children's Miracle Network Hospital
- Served as representative and external voice for the members of the Fraternity to the alumni and local community members

TECHNICAL SKILLS: Solid Modeling (Creo/Solid Works), Technical Writing, Project Management, Data Analysis, Communication, Problem Solving, Leadership, and Time management

References Available Upon Request

Appendix J: Instruction Manual

(This page intentionally left blank.)

2022

Saturn Engineering Solutions

Gas Turbine Engine and Software

INSTRUCTION MANUAL

Mary Lichtenwalner
Barrett McDonald
Ryan McMillan

Table of Contents

How to Use This Guide

This instruction manual has been created so that whether you are a casual user or an advanced programmer, you will be able to effectively use all features installed on the Mercer University gas turbine engine. This includes running the engine, measuring data with the provided Arduino, and recording data with the program Engine Control Tool.

This guide is divided into two main sections. The hardware section provides information on running the engine, that is, starting the engine, controlling the propane flow, etc. The second section is devoted to helping you understand the software being used to record data while running the engine. There are two programs that will be discussed: the Arduino code and Engine Control Tool.

If you have any difficulties while running the engine, you may reach the Saturn Engineering Solutions team at any of the following email addresses.

- Mary Lichtenwalner maryelichtenwalner@gmail.com
- Barrett McDonald barrettmc77@gmail.com
- Ryan McMillan mcmillanryan55@gmail.com

Navigating the Hardware Section

If you are looking for instructions on how to safely operate the gas turbine engine, refer to the operation section of this manual, located under the Hardware Section. This content breaks operation into a variety of steps for each subsystem of the engine.

If you are looking for guidance with troubleshooting mechanical issues with the engine, refer to the Troubleshooting section under Hardware. This content focuses on issues with the spark plug and combustion instability.

Navigating the Software Section

If this is your first time using Engine Control Tool on your laptop, refer to the Installation section to learn how to download Engine Control Tool onto your laptop.

If you have already downloaded Engine Control Tool, we recommend that you begin by familiarizing yourself with the components of the Engine Control Tool GUI. These are shown in the Engine Control Tool Diagram section. Once you are familiar with the layout of Engine Control Tool, you may begin collecting data while running the engine by referring to the Engine Control Tool Quick Start section.

If you would like to learn about each component of Engine Control Tool in more detail, refer to the Engine Control Tool Component Descriptions section.

If you are a programmer and would like to make changes to Engine Control Tool or the Arduino source code, refer to the Programmer's Manual section. Finally, if you are experiencing a specific problem with either Engine Control Tool or the Arduino sensor array, refer to the Troubleshooting section.

Hardware

Engine

Operation

The following section outlines the proper steps required to safely start the gas turbine engine.

Setup

There are several key factors to consider before starting the engine. These considerations are outlined in the following sections.

1. Using a voltmeter, measure the potential difference generated from the battery. For successful operation, the battery should be producing between 10 and 12 volts.
    a. If the voltage reading is outside this range, connect the battery to a trickle charger and leave overnight.
2. Ensure that there is oil in the oil tank using the built-in dipstick.
    a. If there is not enough oil in the tank, consult Dr. Sumner.
3. Ensure there is propane in the propane tank.
    a. If you cannot hear propane sloshing in the tank when moving the tank, replace the propane tank with a filled propane tank from a store like Lowe's or Home Depot.
    b. Confirm the valve is completely closed.
4. Collect the following equipment:
    a. Leaf Blower(s)
    b. Extension Cord
    c. Windows Laptop with USB connections
    d. Fire Extinguisher
5. Locate the test site.
    a. Saturn Engineering Solutions recommends the sidewalk underneath the bridge connecting the Mercer Science and Engineering Building to the parking lot.
6. Move the engine, along with the equipment listed in Step 4, to the test site.
7. Remove the propane tank from the engine and position it as far away from the combustion chamber as possible.

Oil System and Heat Exchanger

The following section outlines the steps required to properly start the flow of oil through the engine.

1. Turn the Oil Pump and Heat Exchanger switch to the ON position.
    a. This switch is found on the electrical box.
2. Ensure that the fan on the Heat Exchanger is blowing.
3. After about 30 seconds, check the oil pressure gauge.
    a. The oil pressure should be in the range of 30-45 psi.

Glow Plug

The following section outlines the proper use of the spark generation system.

1. Turn the glow plug switch to the ON position.

a. The operators should hear a winding sound coming from the glow plug followed by a spark.
b. If the time between sparks is greater than two to three seconds, do not run the engine.
    i. The battery will need to be charged to reduce the amount of time between sparks.

Note: Be ready to turn the Glow plug switch to the OFF position shortly after ignition.

Spool-Up

The following section outlines the proper use of leaf blowers to start the engine.

1. For an electric leaf blower, plug in the leaf blower to an outlet or an extension cord.
2. For a gas leaf blower, start the blower.
3. When the operator controlling the fuel is in position, start blowing air into the compressor intake.
    a. The end of the leaf blower should butt against the compressor intake housing.
        i. NOT THE COMPRESSOR BLADES
4. Do not remove the leaf blower until a blue flame is visible through the combustion chamber window.

Propane

The following section outlines the proper use of the propane system.

1. Ensure propone regulator is turned all the way off.
2. Once the air flow is induced and glow plug is on, slowly twist regulator until engine starts around 2 psi.
3. Shortly after introducing the propane, ignition should occur.
4. Once the engine starts running begin closing the emergency shut off valve to throttle down the fuel consumption.
5. Turn off the glow plug switch to conserve the charge on the battery.

Shut Down

The following section outlines to proper method required to stop the engine.

1. Close the propane emergency shut off valve.
2. Close the propane regulator valve.
3. Close the valve on the propane tank.
4. Confirm there is no combustion in the combustion chamber.
5. Leave the Oil Pump and Heat Exchanger ON for 5 minutes to assist in engine cooling.
6. Turn off the oil system.
7. Let the cart cool for 15 to 30 minutes depending on the duration of the test run.

CAUTION: The engine components will be hot after the test is complete and can burn the operator.

Safety

The following section outlines potential safety concerns regarding operation and modification of the gas turbine engine.

Operation

Gas turbine engines are inherently dangerous. High revolutions per minute, high temperature, and combustible fuels require close attention for safe operation.

Propane

1. Storage
    a. Do not store propane in areas where the tank can be exposed to temperatures significantly higher than atmospheric temperature. Store inside as much as possible.
    b. Do not store the propane tank on the cart.
        i. After the cart has been moved to a storage location, remove the tank from the cart to eliminate any issues with electrical components arcing to the tank.
2. Operation
    a. Do not operate the engine with the tank on the cart.
        i. Move the tank as far from the engine as the tubing will allow.
    b. Note that the regulator valve does not completely cut off flow.
        i. Do not rely on this valve alone to completely restrict the flow of propane.
    c. Do not turn on the flow of propane without the glow plug turned on and air flowing through the engine.
        i. This will cause an explosion.

Compressor

1. Do not reach any part of your body into the compressor housing during operation.
    a. The compressor could suck body parts or clothing into the compressor blades which can cause bodily harm.
2. Do not stand in the radial direction of the compressor.
    a. Compressor blades could break off from the rotor and launch through the compressor housing, sending debris in the radial direction.

Turbine

1. The exhaust from the turbine is VERY HOT.
2. Do not stand behind the turbine exhaust.
    a. Debris could shoot out of the exhaust.
    b. Exhaust gasses are extremely hot.
3. Do not stand in the radial direction of the turbine.
    a. Turbine blades could break off from the rotor and launch through the turbine housing, sending debris in the radial direction.

Other Engine Components

1. Do not touch the combustion chamber, turbine housing, or combustion chamber adaptor.

a. These components are VERY HOT.

Modification

Some subsystems on the cart pose additional safety concerns to operators.

Electrical

1. Do not work on the wiring of the cart when the wiring is connected to the battery.
    a. The wiring can shock you.
2. Be careful of wiring connections that have been recently soldered.
    a. These joints might still be hot.

Glow Plug

The glow plug system is known to have significant grounding issues. Do not work on the glow plug when the battery is connected to the wiring for any reason.

1. Again, do not work on the glow plug when the battery is connected.
2. Before working on the glow plug, ground the body of the main electrical box to the cart to dissipate any charge build up in the system.

Oil System

1. Do not work on the oil system while the oil pump is on.
2. Do not work on the oil system while the battery is connected to the wiring.
3. Be cautious of the oil in the oil lines.
    a. Oil will be in the lines connecting components of the oil system.
        i. Be ready for oil to leak from the open connections.

Troubleshooting

This section covers common problems that may arise during engine operation.

Propane

1. If there is an odor of unburnt propane:
    a. Utilize the shut off valve to throttle the amount of propane entering the system.
    b. Ensure that all connections are leak free within the propane lines.
        i. Use Teflon tape on all propane connections to ensure leak free connections and tighten with wrenches, not by hand.
2. No propane fuel entering the system:
    a. Ensure that the propane tank is not empty
        i. If propane tank is empty, replace tank refill tank at Lowes, Home Depot, etc.
        ii. If the propane tank is not empty, ensure there are no blockages within propane lines

Glow Plug

1. No spark being generated by glow plug.
    a. Turn off all electrical components connected to the battery.
    b. Connect batter to a trickle charger for 3 hours to charge the battery.
    c. Reconnect battery and try glow plug again.
2. Still no spark being generated by glow plug.
    a. Disconnect battery from the system.
    b. Remove the elbow connection connecting the glow plug to the combustion chamber.
        i. Ensure that the wires do not get twisted and entangle during this process which could result in harming the wire connections.
    c. Remove electrical tape around the elbow connection and ensure that all wire connections are intact.
        i. The metal sheath encasing the wires grounds the system. Ensure that this metal sheath is grounded to the elbow connection. This connection is currently made by a wire soldered between the metal sheath and the elbow connection.
    d. Once all wire connections are intact, reconnect the battery and glow plug. A spark should ensue every couple seconds.

Software

Engine Control Tool

Arduino Code

Installation

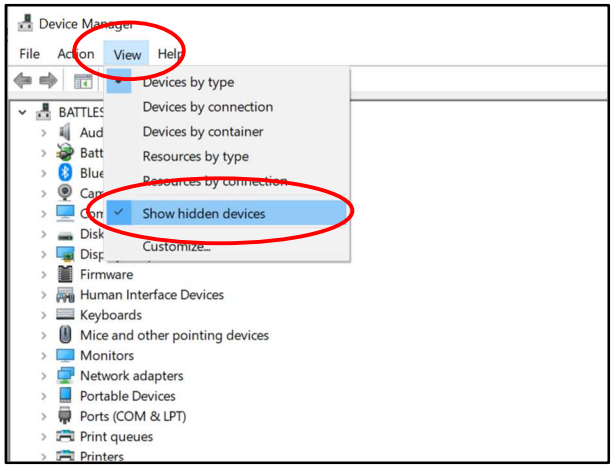
The following sections describe how to access the Arduino program and Engine Control Tool from either the provided USB drive or a Github repository.

USB Option

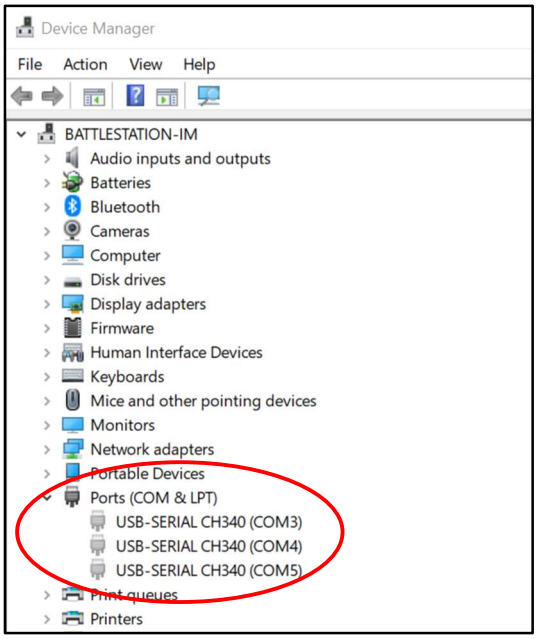
The following steps describe the installation process when using the provided USB drive.

Arduino Source Code

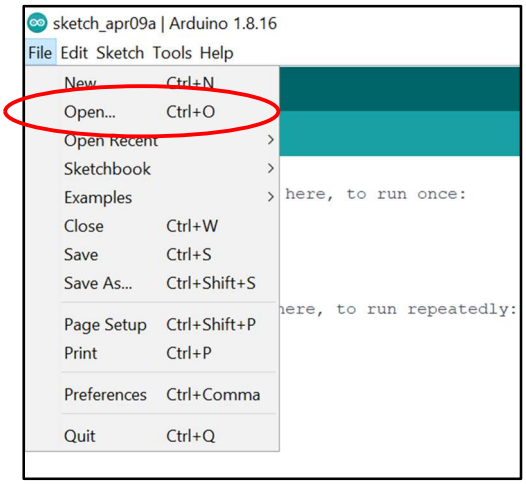
1. First, download the correct version of the Arduino IDE for your operating system from https://www.arduino.cc/en/software.
2. Follow the instructions to install the Arduino IDE.
3. The Arduino must be connected to USB serial port COM3, COM4, or COM5. To confirm that your laptop has one of these ports open, go to device manager by searching “Device Manager” in the start menu. Once Device Manager is open, select View < Show hidden devices.



4. Open the drop down next to “Ports (COM & LPT)” and confirm that either COM3, COM4, or COM5 is available. You may connect to the Arduino with any of these ports.



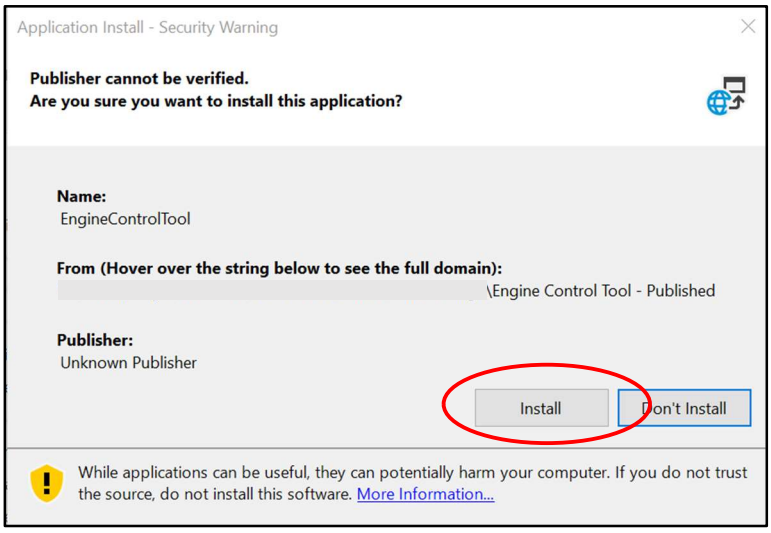
5. If none of these ports are available, you will need to change the source code for both the Arduino and Engine Control Tool to match the port you have available. This problem is addressed in the Troubleshooting section of this manual. If you have one of these ports available, simply connect to the Arduino through one of these ports with the provided USB cable and the Arduino will be ready to send data to Engine Control Tool.
6. To access the Arduino source code, open the USB drive in File Explorer, and select the folder *CombinedProgram*. Copy this folder to the desired location on your device.
7. Once the Arudino IDE is open, select File < Open and navigate to the location where you saved the folder *CombinedProgram*. Open this folder and select the file *CombinedProgram.ino*.



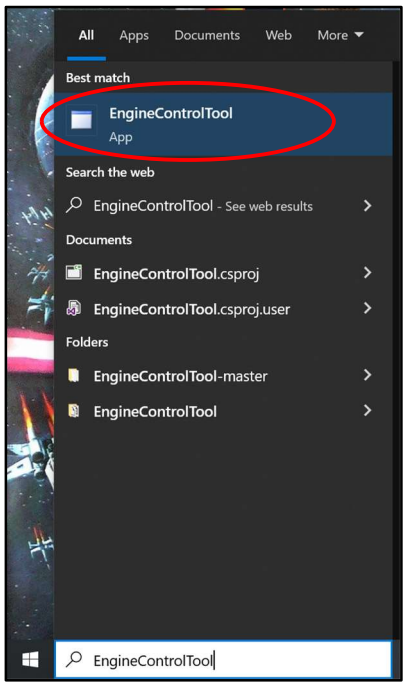
8. The Arduino source code is now opened. Information on the structure of the source code may be found in the section Programmer's ManualProgrammer's Manual.

Engine Control Tool – No Source Code

1. To install Engine Control Tool to your laptop, copy the folder *Engine Control Tool – Published* from the USB drive to the desired location on your laptop.
2. Navigate to the location where you moved this folder and open the file *EngineControlTool.application*. Click "Install" on the notice that pops up.

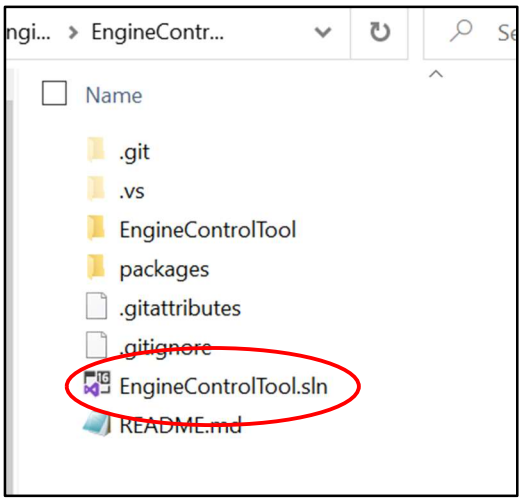


3. After the program installs, you may open the program by searching "EngineControlTool" in the start menu. It may be uninstalled at any time in Settings or Control Panel.

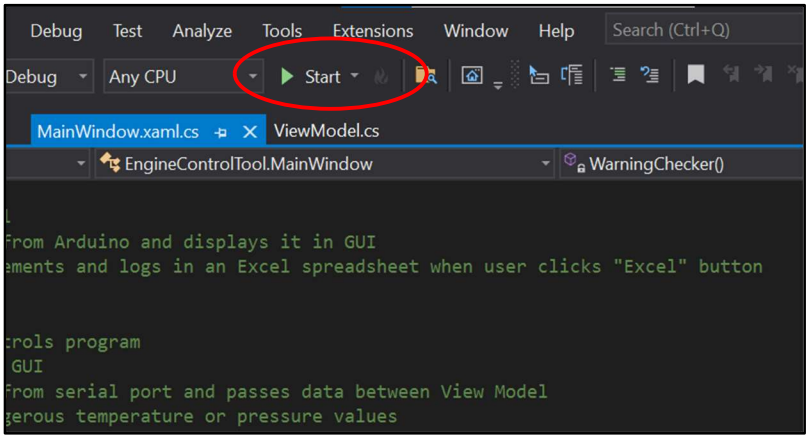


Engine Control Tool – Source Code

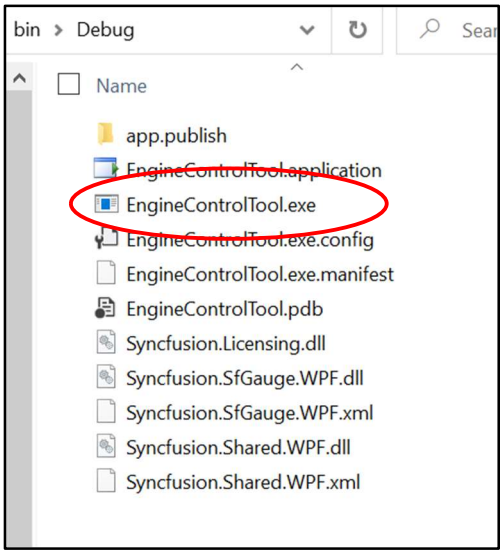
1. If you would like access to the Engine Control Tool source code, you will need to first download Microsoft Visual Studio from https://visualstudio.microsoft.com/. Select the correct version for your operating system. The community version of Visual Studio is free to download and use.
2. Once Visual Studio is downloaded, copy the folder *Engine Control Tool* from the USB drive to the desired location on your laptop.
3. To open in Visual Studio, go into the folder *Engine Control Tool* wherever you saved it, and then select …\EngineControlTool\EngineControlTool.sln. This is the Visual Studio solution file, and it will open the program in Visual Studio.



4. To run the program directly from the source code without installing it to your device, you have two options. With the program open in Visual Studio, you may select the "Start" button at the top center of Visual Studio.



5. Alternatively, you may run the program by selecting the file *EngineControlTool.exe* found at the file path …\EngineControlTool\EngineControlTool\bin\Debug\EngineControlTool.exe where you begin inside the folder *Engine Control Tool* originally copied from the USB drive.

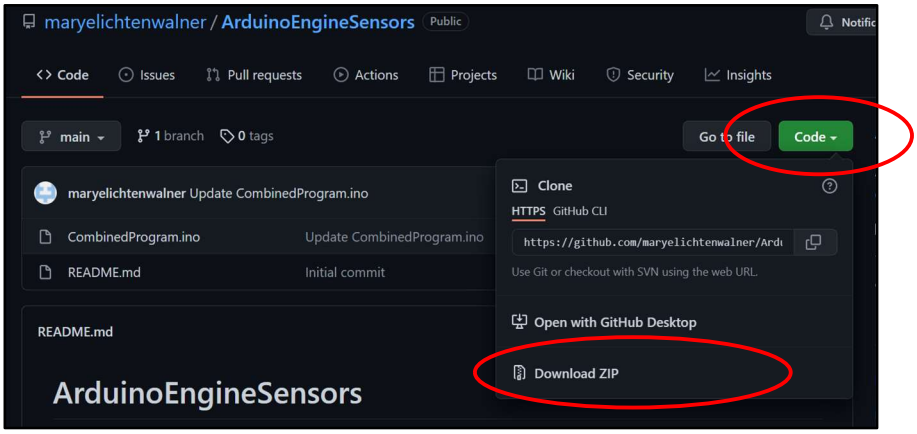


Github Option

If preferred, you may access both the Arduino code and Engine Control Tool through Github. To use this method, you will need both the Arduino IDE (https://www.arduino.cc/en/software) and Visual Studio (https://visualstudio.microsoft.com/) installed on your laptop. Install these programs before proceeding with this method.

Arduino Source Code

1. You will again need to confirm that you have the correct ports open to use the Arduino. Follow the first five steps of the USB installation portion above to confirm that you have connected the Arduino to a compatible USB port.
2. The Arduino source code may be accessed at https://github.com/maryelichtenwalner/ArduinoEngineSensors.
3. Once on this page, select Code < Download ZIP and save the ZIP folder to your desired location.

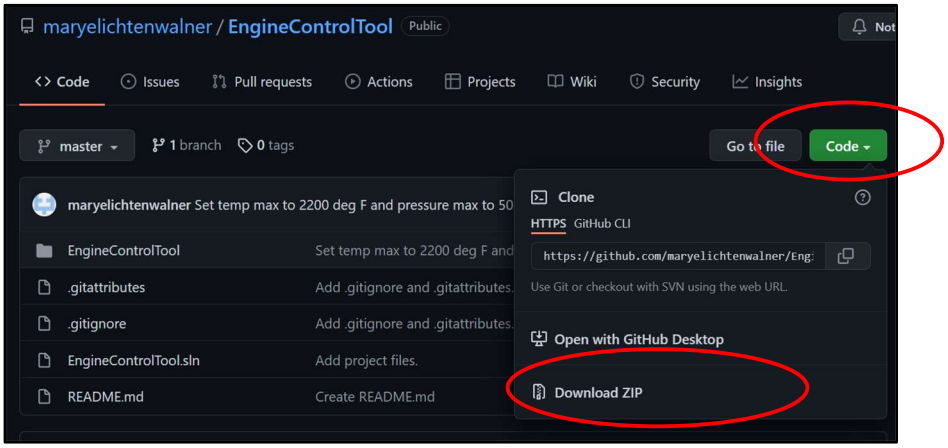


4. Extract the files from the ZIP folder once it has downloaded.
5. Open the Arduino IDE and select File < Open. Navigate to the file *CombinedProgram.ino*.

6. You now have the Arduino source code opened.

Engine Control Tool – Source Code

1. You may access the Engine Control Tool Github repository at https://github.com/maryelichtenwalner/EngineControlTool.
2. Once on this page, select Code < Download ZIP and download a ZIP folder of the repository to your desired location.



3. Extract the files from the ZIP folder once it has downloaded.
4. Open the Visual Studio solution file by navigating to …\EngineControlTool-master\EngineControlTool-master\EngineControlTool.sln.
6. The source code is now open in Visual Studio, and Engine Control Tool may be run by selecting the "Start" button at the top center of Visual Studio.
7. Alternatively, you may run the program by selecting the file *EngineControlTool.exe* found at the file path …\EngineControlTool-master\EngineControlTool-master\EngineControlTool\bin\Debug\EngineControlTool.exe.

Engine Control Tool Diagram

The following image shows the Engine Control Tool GUI. The names for these visual elements given below will be used throughout the rest of this manual.

The following list corresponds to the numbered items on the figure above.

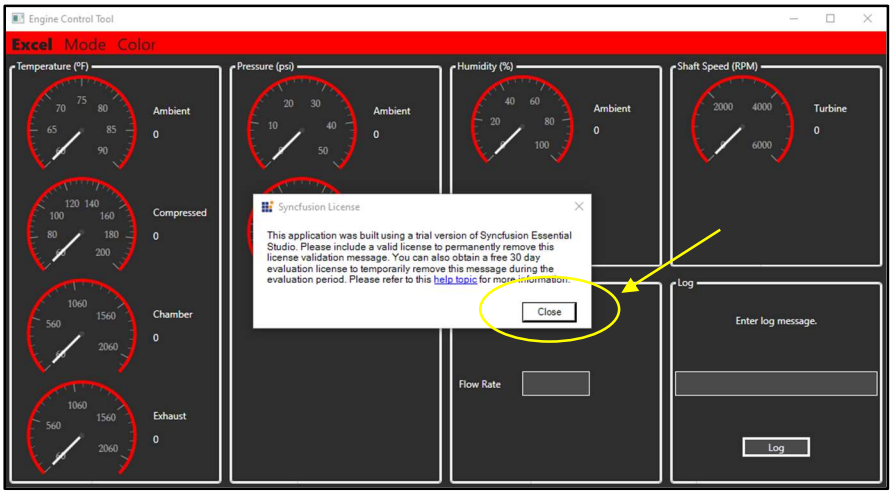
1. Menu Bar
   1.1. Excel Button
   1.2. Mode Button
   1.3. Color Button
2. Temperature Groupbox
   2.1. Ambient Air Temperature Gauge/Label
   2.2. Compressed Air Temperature Gauge/Label
   2.3. Chamber Temperature Gauge/Label
   2.4. Exhaust Temperature Gauge/Label
3. Pressure Groupbox
   3.1. Ambient Air Pressure Gauge/Label
   3.2. Compressed Air Pressure Gauge/Label
4. Humidity Groupbox
   4.1. Ambient Air Humidity Gauge/Label
5. Shaft Speed Groupbox
   5.1. Turbine Shaft Speed Gauge/Label
6. Flow Rate Groupbox
   6.1. Flow Rate Textbox
7. Log Groupbox
   7.1. Log Message Textbox
   7.2. Log Button

Engine Control Tool Component Descriptions

The following sections detail the different components of the Engine Control Tool GUI. All numbers refer to the corresponding labels in the Engine Control Tool Diagram section above.

Startup

Upon startup, Engine Control Tool will pop up a notice about the “Syncfusion License.” This refers to the C# library used to initialize the gauges used to display measurements. You may simply close this message.

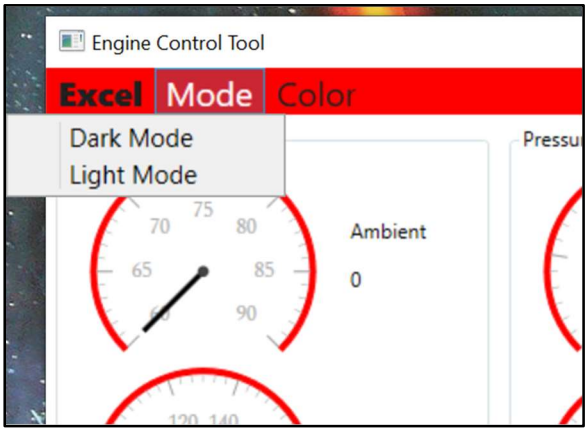


Menu Bar

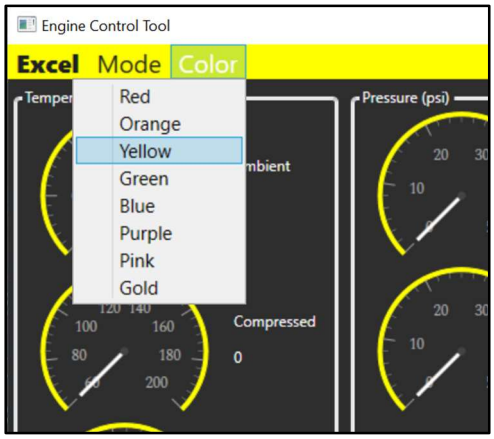
Engine Control Tool has a Menu Bar (1) across the top of the window with three buttons: Excel Button (1.1), Mode Button (1.2), and Color Button (1.3).

The Excel Button exports the recorded data to an Excel spreadsheet. Engine Control Tool continues to operate and record data even after this button is pressed, so it is possible to generate multiple Excel spreadsheets during one engine test run.

The Mode Button opens a drop-down menu where you may choose to display Engine Control Tool in dark mode or light mode.



The Color Button opens a drop-down menu where you may choose from a list of eight accent colors. The selected accent color will change the color of the menu bar and the rims on the gauges.



Both the Mode Button and Color Button make purely aesthetic changes to Engine Control Tool and do not affect the functionality.

Temperature Groupbox

The Temperature Groupbox (2) contains gauges and labels to read four temperature measurements: ambient air (2.1), compressed air (2.2), combustion chamber (2.3), and exhaust temperatures (2.4). The measurements are displayed in degrees Fahrenheit, and each gauge has an upper limit that is high enough to display the expected temperature for that location. For example, the Ambient Air Temperature Gauge has an upper limit of 90°F while the Chamber Temperature Gauge has an upper limit of 2200°F.

Pressure Groupbox

The Pressure Groupbox (3) contains gauges and labels to read two pressure measurements: ambient air (3.1) and compressed air (3.2). Both pressure measurements are displayed in psi.

Humidity Groupbox

The Humidity Groupbox (4) contains a gauge and label (4.1) to display percent humidity of the ambient air. This gauge has a range of 0 to 100 %.

Shaft Speed Groupbox

The Shaft Speed Groupbox (5) contains a gauge and label (5.1) to display the shaft speed of the turbine.

Flow Rate Groupbox

The Flow Rate Groupbox (6) contains the Flow Rate Textbox (6.1). You must record the flow rate of the exhaust using the provided handheld anemometer. Enter this value into this textbox. This textbox will only accept a numerical value. If a nonnumerical input is attempted, Engine Control Tool will change the value in this textbox to 0.

Log Groupbox

The Log Groupbox (7) contains the Log Message Textbox (7.1) where you may enter a custom message to mark some specific timestamp in the Excel output. If you would like to mark some timestamp with a custom message, type your message into the Log Message Textbox. Click the Log Button (7.2) at the moment you would like to log, and the textbox input will be added to that timestamp in your final Excel spreadsheet. If you would like to mark some timestamp but not include a custom message, simply click the Log Button at the moment you would like to mark while leaving the textbox empty, and that timestamp will be marked in the Excel output.

Engine Control Tool Quick Start

The following steps describe how to use Engine Control Tool assuming it has already been installed on your laptop. Numbered items correspond to the numbering used in the Engine Control Tool Diagram section above.

1. Connect the provided USB cable to the Arduino and your laptop. The lights on the Arduino should turn on.
2. Open Engine Control Tool and click “Close” on the Syncfusion License notice that pops up.
3. At this point, Engine Control Tool should be displaying the data from the sensors. If not, refer to the troubleshooting section of this guide.
4. Run the engine. Engine Control Tool will be reading in new data from the sensors once every second. You will see this real time data displayed on the Engine Control Tool screen.
5. Once the engine is running, measure the flow rate of the exhaust using a handheld anemometer, and enter this value into the Flow Rate Textbox (6.1) on Engine Control Tool.
6. When ready to export the recorded data to Excel, click the Excel Button (1.1).

Programmer’s Manual

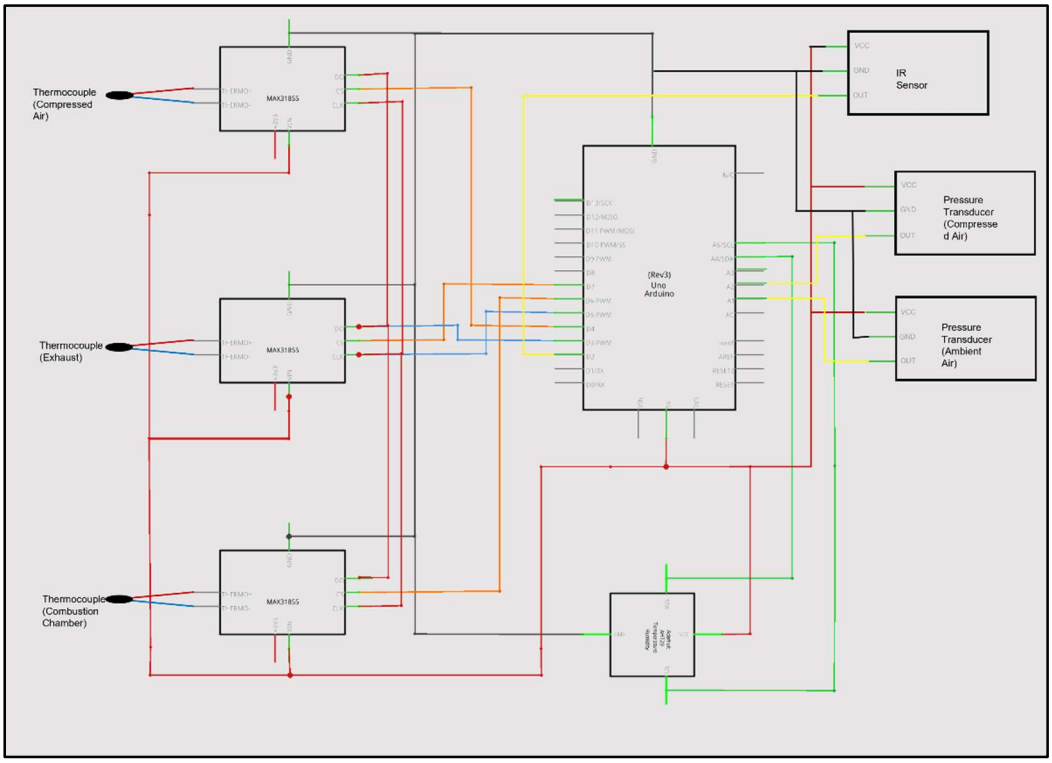
The following sections explain the different sections in both the Arduino source code and Engine Control Tool source code. You may use this information to alter the source code to fit your requirements.

Arduino Source Code

This section should be referenced if you are changing the source code used by the Arduino. The Arduino source code is written in the file *CombinedProgram.ino*. You may access this file by following either method described in the Installation section of this manual. The Arduino source code is divided into four sections. In the source code, these are labeled Section 0 to Section 3 with comments. So, you will be able to easily find the section being discussed by using the find function (CTRL + F). We begin with Section 0 in *CombinedProgram.ino*.

Section 0. Arduino Uno Connections

This section contains only comments. It describes the required pin connections for each sensor used on the engine. A diagram of these connections can be seen below. The wires in the diagram are color coded to match the actual wires installed in the electrical box on the cart.



Section 1. Set Up Sensors

This section contains libraries and definitions that are required to set up each sensor. First, three libraries are included. These are:

- SPI.h is the serial peripheral interface library and is used by the Arduino to communicate with the connected sensors.
- Adafruit_MAX31855.h is a library that allows the Arduino to use the built-in functions associated with the Adafruit MAX31855 thermocouple amplifier breakout board.
- Adafruit_AHTX0.h is a library that allows access to built-in functions that control the Adafruit AHT20 temperature/humidity sensor.

After the library definitions, the thermocouples are the first sensors that are initialized. Each thermocouple is connected to its own Adafruit MAX31855 thermocouple amplifier breakout board. All three MAX31855 boards share a clock and data out pin, and they each have their own chip select pin.

Next, the pressure transducers are set up by allocating two analog pins and setting their initial reading to 0. Following the pressure transducers pin assignment, there is one line of code that creates an instance of the Adafruit AHT20 temperature/humidity sensor. Lastly, the shaft speed sensor is prepared

by defining variables and an interrupt routine that will be used to count the revolutions made by the turbine shaft.

Section 2. Set Up Program

This section contains the setup() function included in any Arduino program. First, the two analog pins for the pressure transducers are designated as input pins. This allows the Arduino to read the value from these transducers.

Next, the attachInterrupt function is called. This causes the infrared shaft speed sensor to begin counting revolutions. Finally, the three MAX31855 boards and the AHT20 sensor are all started based on how they were defined in the previous section.

Section 3. Loop Section

This is the final section of source code in *CombinedProgram.ino*. This section contains the loop() function, another requirement in Arduino code. This section is the code that continuously runs if the Arduino is connected to a power source, in this case, if the Arudino is connected to your laptop.

First, a measurement is taken from each of the three thermocouples and stored in three strings. Next, a reading from each pressure transducer is taken. After receiving this analog reading for each transducer, there are five lines of code that convert the reading to psi. The pressure transducers are calibrated to read ambient pressure correctly, and from there, a linear relationship between the transducer's voltage output and pressure is assumed. If the pressure sensor seems to be reading incorrectly, you may need to change the calibration so that it correctly reads ambient pressure. To do this, there are two variables that you may adjust. The first is a float called "calibration1" and the second is another float named "calibration2". The value "calibration1" corresponds to the ambient air pressure transducer, and "calibration2" corresponds to the compressed air pressure transducer. To increase the reading from the pressure transducer, decrease its calibration variable and vice versa.

Next, a reading is taken from the AHT20 temperature/humidity sensor, and finally, the shaft speed sensor is read and converted to an RPM measurement.

Lastly, all these measurements are combined into one string with a space between each measurement, and this string is sent to the serial port, where it is received by Engine Control Tool. A one second delay occurs, and then, the Arduino repeats this section.

Engine Control Tool Source Code

This section should be used if you would like to make changes to the Engine Control Tool source code. Engine Control Tool source code is in four files, three are written in C# and one is written in Extensible Application Markup Language, or XAML. Each file will be examined in the following sections. Like the Arduino source code, the sections discussed below are commented with the same titles in the source code, so you may easily locate the desired section with CTRL + F.

File 1. MainWindow.xaml

The file *MainWindow.xaml* is written in XAML, and it defines the visual elements of Engine Control Tool. The source code for this file is divided into three main sections.

Section Title	Description	Approximate Lines in Code
Section 1	**Defines multiple style for the components on the GUI. These style definitions are used in the later sections to determine the visual aspects of different components.**	**19 – 236**
Section 2	**Contains the source code to define the Menu Bar (Section 2.1) and the Body Grid (Section 2.2). The Menu Bar contains the Excel Button, Mode Button, and Color Button while the Body Grid contains the six groupboxes that display measurements and receive user input**	**238 – 569**
Section 3	**Defines the label that will flash on the screen if a dangerous temperature or pressure level is measured**	**571 – 613**

File 2. MainWindow.xaml.cs

The class *MainWindow.xaml.cs* contains the C# code controlling the overall functioning of Engine Control Tool. When the program is run, this file creates instances of the other two classes, *ViewModel.cs* and *CreateExcel.cs*. It also contains methods that are called whenever the user clicks a button on the GUI. These include the Mode Button, Color Button, Excel Button, and Log Button. This class also opens the correct serial port and begins reading data from the Arduino as soon as the program is started. Finally, it starts a separate thread that continuously monitors the temperature and pressure measurements. If either of these reach a dangerously high level, it flashes a warning label to the user and instructs them to terminate the flow of propane. Rather than having comments dividing this file into sections, comments divide this file into each method that it contains. The methods contained in this file are discussed in the table below. You may find these methods in the source code by using CTRL + F and searching for the method number.

Method Title	Description	Approximate Lines in Code
Method 1	**Constructor for the MainWindow class. Initializes GUI, connects MainWindow view to view model for passing data between classes, starts a background thread to check for high temps/pressures**	**54 – 69**
Methods 2 – 11	**Called whenever the user selects from the Mode Button or the Color Button. Simply set correct variable in view model to change colors on GUI**	**71 – 132**
Method 12	**Called when Engine Control Tool starts. Searches for connected serial port**	**134 – 172**
Method 13	**Receives data from connected serial port**	**174 – 192**
Method 14	**Takes received data, updates the GUI to display data, and records data in a list variable in the view model**	**194 – 315**
Method 15	**Called when user clicks the Excel Button. This calls methods from the CreateExcel class to generate the final Excel spreadsheet**	**317 – 342**

Method 16	**Called when user changes the entry in Flow Rate Textbox. Confirms that the entry is numerical, and if not, changes the content of the Flow Rate Textbox to 0**	**344 – 362**
Method 17	**Runs on a loop beginning when Engine Control Tool is started. Checks for a temperature or pressure that is over the allowed threshold, and notifies user if so**	**364 – 409**
Method 18	**Checks to see if Log Button has been clicked. If so, set required variables in view model**	**411 – 419**

File 3. CreateExcel.cs

The *CreateExcel.cs* class is responsible for generating two Excel spreadsheets, one in imperial units and one in metric units. This class contains four methods besides the constructor. Three of these methods are called from *MainWindow.xaml.cs* when the user clicks on the Excel Button. This class uses the library *Microsoft.Office.Interop.Excel* to interface with the installed version of Excel.

Method Title	Description	Approximate Lines in Code
Method 1	**Constructor for the CreateExcel class. Gives this class access to the view model in case data must be passed between this class and other classes**	**28 – 34**
Method 2	**Starts new Excel application. Creates two Excel spreadsheets and names them**	**36 – 53**
Method 3	**Populates one of the Excel spreadsheets with data given in metric units**	**55 – 176**
Method 4	**Populates the other Excel spreadsheet with data given in Imperial units**	**178 – 297**
Method 5	**Draws a thick border around Excel cells. Called by Methods 3 and 4**	**299 – 305**

File 4. ViewModel.cs

The final class*, ViewModel.cs*, is a model class. Its only function is to store data so that data may be passed between the XAML view and other classes. This is essential because the only thread that can interact with the GUI elements is the main thread. However, the serial data being read from the Arduino is being received in a separate thread. This thread cannot access the GUI elements, so it instead passes its received data to the view model. The GUI elements are bound to the view model, so they can see the stored data. This is how the gauges and labels in the GUI are updated to show the current data.

The view model also contains lists that are constantly being added to as new data is received. These lists are accessed by the *CreateExcel.cs* class and are used to populate the Excel spreadsheets. The *ViewModel.cs* class is organized by sections.

Section Title	Description	Approximate Lines in Code
Section 1	**Contains definitions that allow the GUI elements to be altered when new data is received.**	**18 – 39**
Section 2	**Contains 3 variables that are changed based on the selections made from the Mode Button or Color Button**	**41 – 67**
Section 3	**Contains 11 variables that store the real-time measurements and mark whether the user would like to log some specific timestamp**	**69 – 165**
Section 4	**Stores all the lists that are used to record data and to populate the Excel spreadsheet**	**167 – 257**

Section 5	**Contains 2 variables that are changed if the warning label needs to be visible**	**259 – 285**

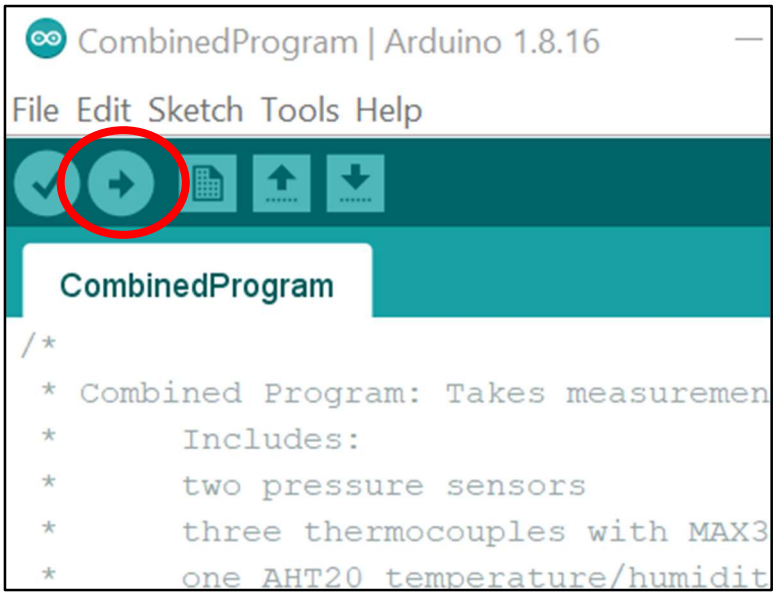
Troubleshooting

The ports “COM3”, “COM4”, and “COM5” are not available on my device.

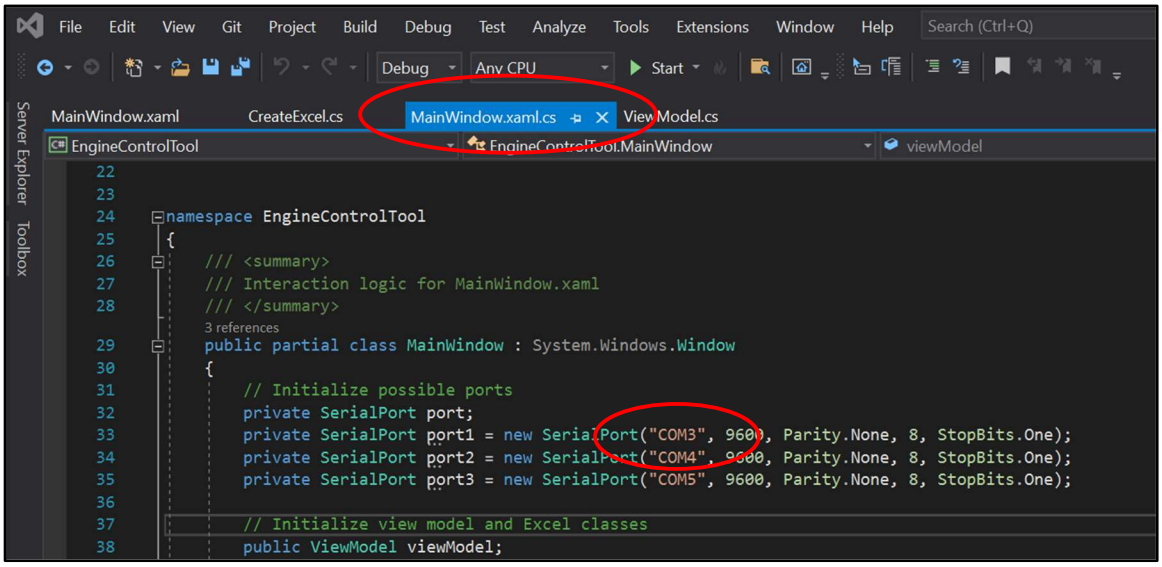
The Arudino installed on the engine is compatible with serial ports “COM3”, “COM4” or “COM5”. If none of these are available on your device, you will have to upload the Arduino code to the Arduino using your available port, and you will need to change the Engine Control Tool source code to read from an alternative port.

Begin by uploading the Arduino code through your chosen port. First, connect the Arduino to your laptop with the provided USB cable. Open the Arduino code (*CombinedProgram.ino*) in the Arudino IDE. On the menu bar, select Tools < Port < COMx. The Arduino IDE will automatically recognize which port you connected to the Arduino with. For example, if you connected to COM6, you would select Tools < Port < COM6.

Next, click on the upload arrow to upload the code to the Arudino.



Now, you must alter the Engine Control Tool source code to recognize data from your selected port. Open the Engine Control Tool source code in Visual Studio following the steps described in the Installation section. Go to the file *MainWindow.xaml.cs*. On line 33, change “COM3” to whichever port you selected in the Arduino program. For example, if you chose “COM6” in the Arduino IDE, change “COM3” to “COM6” on line 33 of this file.



The program may now be run on your device following the steps described in the Installation section.

Occasionally, two of the thermocouple readings are lost.

The MAX31855 thermocouple amplifier breakout boards that are used to take measurements from the thermocouples do not accept readings from grounded thermocouples. So, if any of the thermocouples are touching each other, it will register as an error in the reading. The thermocouple installed directly after the combustion chamber has no metal sheath on it. Thus, when the air in the combustion chamber is too turbulent, it is possible that this thermocouple touches the edge of the combustion chamber. When this happens, since the exhaust thermocouple is mounted to the metal cart, it becomes grounded with the combustion chamber thermocouple, causing an error in the measurement.

This problem will typically fix itself in a matter of seconds, as the combustion chamber thermocouple is blown away from the inside wall of the combustion chamber. As soon as the thermocouples are not grounded, accurate measurements will be recorded again.

The Excel spreadsheet was not fully populated.

If you click inside the Excel spreadsheet before it is finished generating, it will cause an error and stop the spreadsheet generation. Simply keep Engine Control Tool running, click the Excel Button to generate the spreadsheet a second time, and let the spreadsheet fully populate before you begin manipulating and analyzing the data.

Engine Control Tool is not displaying any data.

Restart the Engine Control Tool Application.

If this does not fix your problem, check that the Arduino is connected to the proper port by following the steps described in the first part of this Troubleshooting section.

If the problem persists, open the electrical box, and confirm that no connections have become loose in the Arduino circuit.