

Hypoxic Incubator Control System

A Senior Design Project
Presented to
the Faculty of the Biomedical Engineering Department
California Polytechnic State University, San Luis Obispo

In Partial Fulfillment
Of the Requirements for the Degree
Bachelors of Science in Biomedical Engineering

Prepared by
Derek Tran, Jennifer Rojas, Marina Zellers
March 2023

Table of Contents

1. Executive Summary	4
2. Introduction.....	4
3. Background.....	5
Customer Information	5
Intellectual Property Identification	5
4. Objectives	6
Problem Statement	6
Indications for Use:.....	6
Customer Requirements and Engineering Specifications	6
House of Quality	7
Conjoint Analysis	9
5. Project Management.....	10
Gantt Chart.....	10
Budget Diagram.....	10
6. Concept Generation	11
Morphological Analysis	11
Concept Evaluation.....	12
Conceptual Model	14
7. Detailed Design.....	16
Hardware Selection	16
User Interface Design	18
Schematic Design.....	21
Gas Control System	21
8. Prototype Manufacturing Plans	23
General Prototyping Procedure	23
Bill of Materials.....	23
Resources and Training	24
Manufacturing Process: Touchscreen Stand.....	24
9. Test Plans.....	26
1 Hour and Overnight Time Outside of Range Tests.....	26
Time to Stabilization Test	27
Range of Setpoints Test.....	27

Interface Usability Test.....	28
10. Testing Results.....	29
11. Conclusions.....	31
12. Discussion	32
13. Instructions for Use	33
Operation of Incubator with Touchscreen Procedure	33
Operation of Incubator without Touchscreen Procedure	34
Upload Arduino Code Procedure	34
Output Pressure Calibration Instructions.....	35
14. Works Cited	37
15. Appendix.....	38
Appendix A. Hand calculations for flow rate.....	38
Appendix B. Hand Calculations for CO2 Flow Rate	39
Appendix C. Network Diagram	40
Appendix D. Test Data	41

1. Executive Summary

This paper describes the functionality of the current Hypoxic Incubator, a project started by BMED graduate students Simone Helfrich and Makenzie Jones, with the objective being to redesign the device to address any potential areas of improvement. The hypoxic incubator has been designed to allow for the control of the O₂, CO₂, and N₂ levels such that researchers can grow and study cell culture in hypoxic conditions; however, its efficacy and efficiency are limited by the fact that it often overshoots the desired setpoints for O₂ and CO₂ in addition to being difficult to use due to requiring the operator to understand the basics of coding. These limitations directly translate into the governing customer requirements for our project. Firstly, to address these limitations, a conceptual model was developed to help pinpoint the root cause of overshoot which was determined to be the inability to control the excessive flowrate. Different design concepts were generated to mitigate this issue and were drafted utilizing a morphological analysis. A concept evaluation revealed that the different aspects of our individual design concepts excelled in separate functional categories. As a result, the final design was a Hypoxic Incubator equipped with a new pressure regulator and touchscreen which combined the best aspects of each individual design concept. Detailed test plans were outlined which allowed for validation and verification of our design against customer requirements and engineering specifications. While there were some complications during the test procedure, trends in the test data show that the control system is effective at maintaining the desired gas setpoints necessary to cultivate cells in a hypoxic environment. Specifically, the issue of overshoot that was prevalent in the previous design iteration is now mitigated as CO₂ and O₂ are kept within their acceptable ranges of +/- 0.5% and +/- 0.1% from the setpoint, respectively. The percentage of time spent outside the acceptable range per one-hour incubation time was less than 1.0% in some tests. Additionally, the newly added touchscreen received a usability score of 4.4 out of 5, indicating that it was easy to use with little to no confusion. While the most pertinent issues are addressed, this paper also highlights some future improvements for the incubation chamber such as the implementation of a PID controller to effectively reduce excess gas usage.

2. Introduction

Hypoxia has been linked to a wide range of effects on cellular function ranging from the beneficial growth of regenerative stem cells to the detrimental development of pathological conditions (Brennan, 2014). As such, the Hypoxic Incubator was designed and developed to study the behavior of various cell types in such conditions. However, the main concern is with the Hypoxic Incubator Control System (HICS), which has the tendency to overshoot the desired gas levels and the inability to maintain the desired gas levels. That is, a programmed setpoint of 5.0% CO₂ may result in a gas level of 8.0% CO₂. In more severe cases, this overshoot has been recorded to be upwards of 20.0% CO₂ (Helfrich, 2022). Despite this issue, there have been some attempts to incubate cells; however, since the CO₂ was out of the acceptable range for a large portion of the experiment, no valid results could be obtained. Thus, it is imperative that HICS be redesigned to mitigate this issue and increase the validity of any results. While the Hypoxic Incubator system was designed by Biomedical Engineering graduate students, its use is not limited to the Biomedical Engineering Department. Its value and research opportunities can be very easily translated to the Biology and Biochemistry departments to study various cellular pathways under the conditions of hypoxia.

3. Background

Customer Information

Since this project is a continuation and extension of a previous BMED master's project, a lot of the information we will review will come directly from the Hypoxic Incubator Chamber documentation written by Helfrich and Jones. This is an invaluable resource because not only does it cover all the technical information relevant to the HICS including detailed schematic diagrams, governing software, and operation manual, but also the thought process that resulted in the development of all those technical aspects. Furthermore, they also documented some potential solutions to the issue of overshooting which pertain to the idea of finer control over the flow rate of the input gases. Most importantly, a significant portion of the document reviews the testing of the HICS, including metrics such as "Time Out of Spec" and "Max Overshoot". After our redesign of the HICS, conducting these same tests and comparing our results back to these metrics will provide us with quantifiable results to validate the improved efficiency and efficacy of our redesign.

Intellectual Property Identification

There are several patents revolving around hypoxic incubators that claim functions similar to our HICS. A Cancer cell enrichment system (US20210388306A1) describes a hypoxic cell incubator gas flow system comprising an oxygen sensor, oxygen and nitrogen sources, an air injection pump, and a control system to regulate and maintain oxygen levels below 20%. The control system includes instrumentation and programming by which oxygen levels are measured and oxygen levels are readjusted. Currently, this patent is pending, but if it is approved, our group will need to take precautions not to infringe on this design.

The abandoned patent US20160312169A1 describes a hypoxic incubator with the ability to maintain one or more pre-set temperatures and gas compositions. Chamber gas is constantly circulated to sensors which cause solenoids in the gas system to open and close, correcting the gas levels in the chamber. Besides the aforementioned information, the patent claims did not discuss in detail the control system electronics. As seen in the figure below, the incubator detailed in this patent was similar in function and appearance to the one designed by Helfrich and Jones.

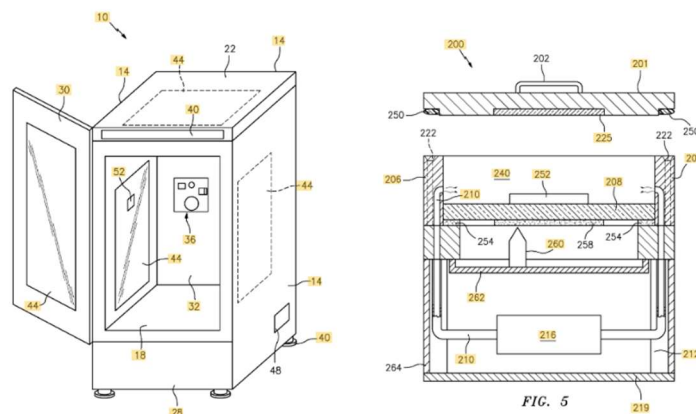


Figure 1. Hypoxic incubator from abandoned patent US20160312169A1.

More generally, our research has also led us to patents pertaining gas control systems. While not applied specifically to incubators, these systems often allow the user to control the gas level to a desired setpoint. This understanding can be translated to our device. For example, the patent USRE41299E1 describes a solenoid valve control system implementing several circuit boards and separate valves joined together; the organization and placement of various components are discussed in detail. The patented multi-gas flow controller method (patent number 20210190575) describes a method for controlling a single mass-flow controller. The system obtains nonlinearity data and sensitivity coefficients from the mass flow and implements this data to control the mass flow rate with precision. The current hypoxic incubator design does not use a mass-flow controller; implementing one would significantly increase the cost of the project. So, it is unlikely that the final design will infringe on this patent. Though our system will not implement the complex instrumentation described in the above patents, we may consider the methods used above to better design our control system.

Finally, although patent infringement is an important factor to consider when revising the hypoxic incubator control system, the consequences for accidentally infringing on an existing or pending patent are limited considering the incubator was not intended to be reproduced or commercialized.

4. Objectives

Problem Statement

The immediate objective of this project is to redesign and refine the Hypoxic Incubator Control System to mitigate any issues of gas level overshoot and stability of gas level over time such that it can successfully incubate cells. With the completion of the former, the scope of this project may be extended to improve the overall user experience with the device.

Indications for Use:

The hypoxic incubator control system is indicated for use in conjunction with the hypoxic cell culture incubator (located in the MPS lab of the Biomedical research department) to control the percentage levels of O₂, CO₂, and N₂ desired to cultivate mammalian cells. This control system is intended for use by students and faculty members of the Cal Poly Biological and Biomedical research departments.

Customer Requirements and Engineering Specifications

As mentioned previously, this project is a continuation of a previous BMED master's project. Therefore, many of the customer requirements for this project can be drawn from the limitations of the current design which include the inability to accurately change gas levels and maintain those gas levels afterward. These customer requirements were then refined and translated into quantifiable engineering specifications.

The table below represents the customer requirements and corresponding engineering specifications for the HICS. The primary function of controlling the gas levels within the chamber can be broken down into smaller requirements which include the ability to adjust gas levels, maintaining those gas levels over time, and reaching the desired gas level within a specified time frame. The values for the passing metrics associated with these engineering specifications were derived directly from the customer requirements that governed the initial development of the

hypoxic incubator. The remainder of the customer requirements such as an easy-to-use interface and inexpensive cost are derived from the user's current inability to easily set gas levels and the project's budget limitations. Measurable engineering specifications were created based on these specific customer requirements to ensure that any design concept will address each of these needs.

Table 1. Customer Requirements & Engineering Specification Table

Customer Requirements	Engineering Specifications	Metric
Gas levels can be controlled and changed	Range of possible O2 levels	0.1 to 10%
Gas levels can be maintained	% Deviation over time falls within acceptable range	0.1% for O2 0.5% for CO2
Gas levels can be easily reached within a timely manner	Time to reach desired gas level	< 5 min
Easy to use interface	Questionnaire ranking	Minimum average of 4
Inexpensive	Final cost falls within budget	< \$750

House of Quality

In some instances, an engineering specification may address multiple customer requirements. As a result, a House of Quality (HOQ) was constructed to determine the relative importance of each of these engineering specifications. To do so, each customer requirement was assigned an importance weighting which can be seen in the yellow section of the HOQ. Then, each engineering specification was ranked according to how much they correlated to a given customer requirement. A value of "9" represents a strong correlation, "6" a medium correlation, "1" a weak correlation, and the absence of a value represents no correlation. The importance of an engineering specification was then tabulated by the summation of each correspondence value multiplied by the importance weight of the customer requirement.

Range of possible O2 levels									
% Deviation over time falls within an acceptable range									
Time to reach gas level									
Number of steps to change gas levels									
Final cost falls within budget	-	-	-	-					
Engineering Requirements	Range of possible O2 levels						Competitor Analysis		
	% Deviation over time falls within an acceptable range						Existing control system	Incuber Incubator	lbidi Gas Incubation System
Customer Specifications		% Range	%	Hrs	#	€			
Gas levels can be controlled and changed	20	9	6				2	5	4
Gas levels can be maintained	15	1	9	1			2	5	5
Gas levels can be reached within a timely manner	10	3	6	9			2	5	4
Easy to use interface	10	1			9		1	5	3
Inexpensive	15					9	3	1	1
Importance		235	315	105	90	135	880		
Importance Weight (%)		27	36	12	10	15	100		
Existing Control System		4.5%-5.5%	1%	3	4	2500			
Incubers		0.5%-20%	0.10%	< 0.5*	3	10000			
lbidi Gas Incubation System		0.5-21%	0.20%	<0.5*	5	10995			
Target		0.1%-10%	1%	1	4	200			
Threshold			2%	3	6	700			

* These values were not included in product specifications.

9 Strong Correlation
 3 Medium Correlation
 1 Small Correlation

Figure 2. HICS House of Quality.

From our HOQ it was determined that the engineering specifications of most importance were “Range of Possible O₂ and CO₂ levels” and “Deviation of gas percentages falls within range”. In accordance with these results, these will be the engineering specifications to which the most resources and attention will be delegated.

Some unnecessary customer requirements were removed from the House of Quality, such as the range of possible CO₂ levels, releasing above a certain pressure, and precision of gas sensors in a specified range. These requirements were already fulfilled by the previous iteration of the device.

Conjoint Analysis

Our initial research revealed several key attributes that may be pertinent to our design concepts. A conjoint survey and analysis were conducted to determine how much the customer values each of these design attributes. The following table represents the attributes and design alternatives for each attribute.

Table 2. Attributes Analyzed in the Conjoint Analysis

Attribute	1	2
Interface	Touch screen interface	Button-based interface
Solenoid Flow Coefficient	4.6 Gal/min	2 Gal/min
Maximum Regulator Output Pressure	60 psi (Smaller range, greater precision)	200 psi (Greater range, less precision)

The survey participants were then asked to rank the following conjoint cards (combinations of each attribute) based on what they believed would most likely contribute to a successful device:

Table 3. Conjoint Cards Provided to Survey Participants

Conjoint Card	Description
(1,1,1)	A hypoxic incubator system with a touch screen interface, a solenoid with flow coefficient of 4.6 Gal/min, and an air regulator with 60 psi.
(1,2,2)	A hypoxic incubator system with a touch screen interface, a solenoid with flow coefficient of 2 Gal/min, and an air regulator with 200 psi
(2,2,1)	A hypoxic incubator with a button-based interface, a solenoid with flow coefficient of 2 Gal/min, and an air regulator with 60 psi.
(2,1,2)	A hypoxic incubator with a button-based interface, a solenoid with flow coefficient of 4.6 Gal/min, and an air regulator with 200 psi.

Table 4. ANOVA Analysis of Conjoint Survey

	Coefficients	Standard Error	t Stat	P-value	Lower 95%	Upper 95%	Lower 95.0%	Upper 95.0%
Intercept	8.88178E-16	0.709459888	1.25191E-15	1	-1.421218937	1.421218937	-1.421218937	1.421218937
X1	0.466666667	0.268150633	1.740315366	0.087297	-0.0705036	1.003836933	-0.0705036	1.003836933
X2	0.466666667	0.268150633	1.740315366	0.087297	-0.0705036	1.003836933	-0.0705036	1.003836933
X3	0.733333333	0.268150633	2.734781289	0.008343	0.196163067	1.2705036	0.196163067	1.2705036

The results of the ANOVA analysis of the conjoint results can be seen above. The only attribute that was significant to our customers, denoted by a P-value of less than 0.05, was the maximum regulator output pressure. Additionally, the coefficient of 0.73 indicates that that the first alternative, the 60-psi regulator, was relatively more important to our customers than the second alternative. Subsequently, this design attribute will be heavily considered during our design conceptualization.

However, while it was not statistically significant, a lower coefficient solenoid has the same effect of decreasing flow rate as a more precise air pressure regulator. This concept may not have been thoroughly explained to the survey participants and consequently, will continue to be considered in the design process.

5. Project Management

Our project management system utilizes two main tools to efficiently complete assigned deliverables while maintaining an appropriate workflow and ensure that our final design of the Hypoxic Incubator aligns with the designated budget.

Gantt Chart

The Gantt chart outlines important milestones of our design process that will dictate the progress of this project and ensure that all major deliverables are completed on time. From the Gantt chart, a network diagram was produced which provided a predicted timeline of this project and highlighted relevant tasks. Additionally, the Gantt chart also allowed us to assign roles and responsibilities throughout the duration of the project. The full network diagram can be seen in Appendix C.

Budget Diagram

The second project management tool that was used is a budget sheet, a living document that was continually updated to estimate and track costs throughout our project. The current budget displayed below reflects the items purchased over the duration of this project.

Table 5. Final Project Budget

Item Description	Purpose	Associated Task	Actual							Notes
			Product Number	Unit	Qty	Cost/Unit	Tax	Shipping	Total Cost	
Gas Tank Regulator	Regulate output pressure of gas that goes into incubator	Regulate the gas pressure	46Z474	1	1	\$364.69	N/A	N/A	\$364.69	Current CO2 regulator has a maximum output pressure of 200 psi, while O2 and N2 have a max pressure of 60 psi (HF)
Gas Regulator Hose Barb	Connection between the gas line tubing and regulator	Connection	3DTN3	1	1	\$22.72	\$32.10	N/A	\$54.82	The current solenoid releases too much gas during the shortest open time (HF)
Arduino - TFT LCD Display	Allows for the user to interface with the HICS without a computer	User Interface	Link	1	1	\$16.99	N/A	N/A	\$16.99	
Jumper Wires	Used for form connections between various components	Electrical Connection	Link	1	1	\$6.49	\$1.70	N/A	\$8.19	Wires to facilitate connection between the Arduino and other components (SP)
N2 Gas Tank	Supplies N2 for the Hypoxic Incubator	Gas supply	N/A	1	2	\$15.00	N/A	N/A	\$30.00	
O2 Gas Tank	Supplies O2 for the Hypoxic Incubator	Gas supply	N/A	1	1	\$15.00	N/A	N/A	\$15.00	
CO2	Supplies CO2 for the Hypoxic Incubator	Gas supply	N/A	1	1	\$15.00	N/A	N/A	\$15.00	
Arduino - UNO Board	Required for use with the LCD Display	User Interface	Rev 3	1	1	\$28.50	N/A	N/A	\$28.50	Previous UNO board stopped connecting to the lab computer.

6. Concept Generation

Morphological Analysis

Three main functions were determined for the HICS: *set desired gas levels*, *display desired gas levels* and *release and control gas flow*. Release and control gas flow can be further broken down into functions such as control flow rate into chamber and translate gas setpoints into the physical opening and closing of solenoid valves. However, the concepts generated to optimize these functions are either displayed below or involve mostly programing which cannot be drawn out.









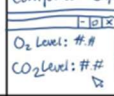

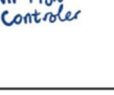

Morphology						
Product: Hypoxic Incubator Ctrl System			Organization Name :			
Function	Concept 1	Concept 2	Concept 3	Concept 4	Concept 5	Concept 6
Set desired Gas levels (by user)	 Touch Screen	 Knob	 Button control	 Mess with code	 Computer Program	
Display Desired and Current Gas Levels	 Screen	 Matrix with code	 Computer code in Terminal	 Computer display		
Release Gas	 Replace Solenoids (Low Flow Coefficient)	 Air Flow Controller	 Replace Regulator			
Team member: Derek Tran		Team member: Jenny Rojas		Prepared by:		
Team member: Marina Zelke		Team member:		Checked by:		Approved by:
The Mechanical Design Process Copyright 2008, McGraw Hill				Designed by Professor David G. Ullman Form # 15.0		

Figure 3. Morphological Chart Based on HIC Subfunctions

Of the various methods to set and display gas levels in the figure above of the morphological chart, the touch screen, computer program, and matrix screen and button combination were the most realistic. Along with replacing the solenoid and regulator, these morphologies were combined to create the morphological combinations below. Apart from being realistic combinations, the pairing of solenoid or regulator replacements with whichever particular method of selecting and displaying gas values was arbitrary. The air flow controller was not included as an option because it would have been an unlikely choice.

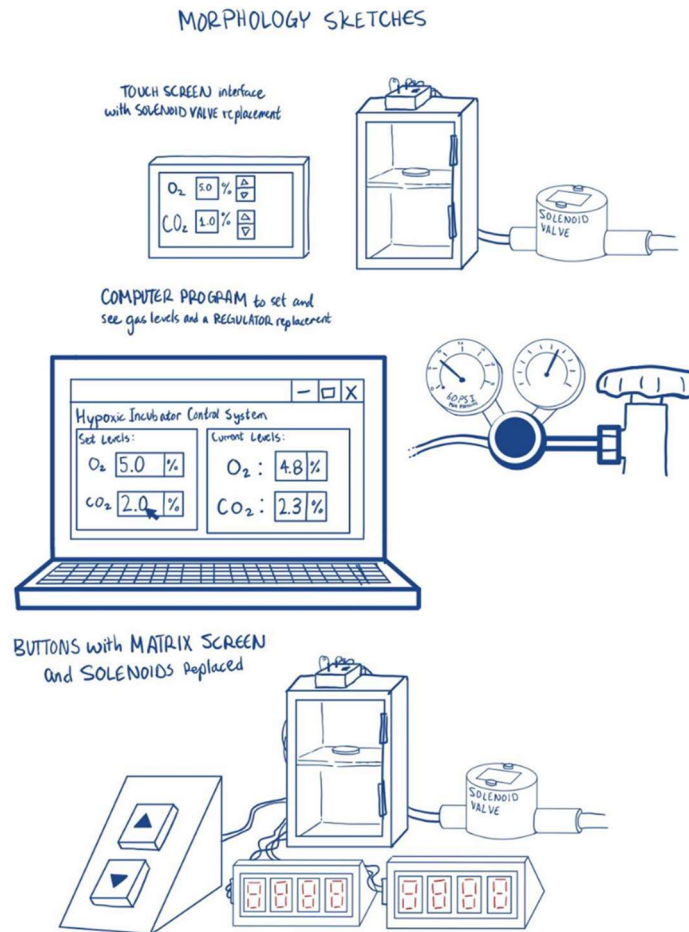


Figure 4. Morphology Sketches

Concept Evaluation

From our morphological concepts, a Pugh matrix was constructed as a tool to compare and evaluate each concept relative to one another. The main elements of the Pugh matrix include (1) the criteria in which we evaluate each design against, (2) the importance of each of those criteria, and (3) all the concepts we which to evaluate. It is important to note that while not the same, each of these criteria is foundationally derived from one or more of the customer requirements.

Within the Pugh matrix, one design concept is selected as a baseline and the rest of the designs are compared against this baseline. A "+1" score indicates that the alternative performs better than the baseline for the given criteria, while a "-1" score indicates a worse performance. In order to ensure the validity of our analysis, the Pugh matrix is iterated according to the number of design concepts with each iteration switching the design concept that is used as the baseline. This process will ensure that the best design defined by the highest weighted total will be the best concept to pursue.

The Pugh matrices for our three morphologies can be seen in the figure below:

Issue: Choose a Hypoxic Incubator Control System.		Touch Screen w/ Solenoid Valve	Computer Program w/ Regulator	Buttons & Matrix w/ Solenoid Valve
Deviation in Gas Percentages Over Time	30	Datum	1	0
Range of Possible Set Points (Discrete Set Points)	25		0	-1
Ability to Adjust Code	10		1	0
Steps to Set Gas Levels	20		-1	-1
Intuitive (User Interface)	15		-1	-1
Total			0	-3
Weighted Total			5	-60

Issue: Choose a Hypoxic Incubator Control System.		Computer Program w/ Regulator	Touch Screen w/ Solenoid Valve	Buttons & Matrix w/ Solenoid Valve
Deviation in Gas Percentages Over Time	30	Datum	-1	-1
Range of Possible Set Points (Discrete Set Points)	25		0	-1
Ability to Adjust Code	10		-1	-1
Steps to Set Gas Levels	20		1	-1
Intuitive (User Interface)	15		1	-1
Total			0	-5
Weighted Total			-5	-100

Issue: Choose a Hypoxic Incubator Control System.		Buttons & Matrix w/ Solenoid Valve	Touch Screen w/ Solenoid Valve	Computer Program w/ Regulator
Deviation in Gas Percentages Over Time	30	Datum	0	1
Range of Possible Set Points (Discrete Set Points)	25		1	1
Ability to Adjust Code	10		0	1
Steps to Set Gas Levels	20		1	1
Intuitive (User Interface)	15		1	1
Total			3	5
Weighted Total			60	100

Figure 5. Pugh Matrices for Each Morphology

From the Pugh Analysis, the best design concept was one that utilizes a computer program to interface between the user and the control system while using an air pressure regulator to control the gas flow rates. The design choice with the next highest total score was one that utilizes a touchscreen with a lower flow coefficient solenoid valve which adds additional value in user-interface. Despite the differences in scores calculated using the Pugh Analysis, both designs provide key aspects that when combined provide a more wholistic and well-rounded concept. As a result, the final design concept integrates these two key features from each of the designs to create a solution that uses a touch screen to interface with the user and a more precise air pressure regulator to offer more fine control over gas flow rate.

An important consideration in this final design is that in the development of our Pugh matrix we assumed that an air pressure regulator would offer more control over gas levels in comparison to the solenoid alternatives. However, the impact on flow rate due to these two different factors will be discussed in the following section.

Conceptual Model

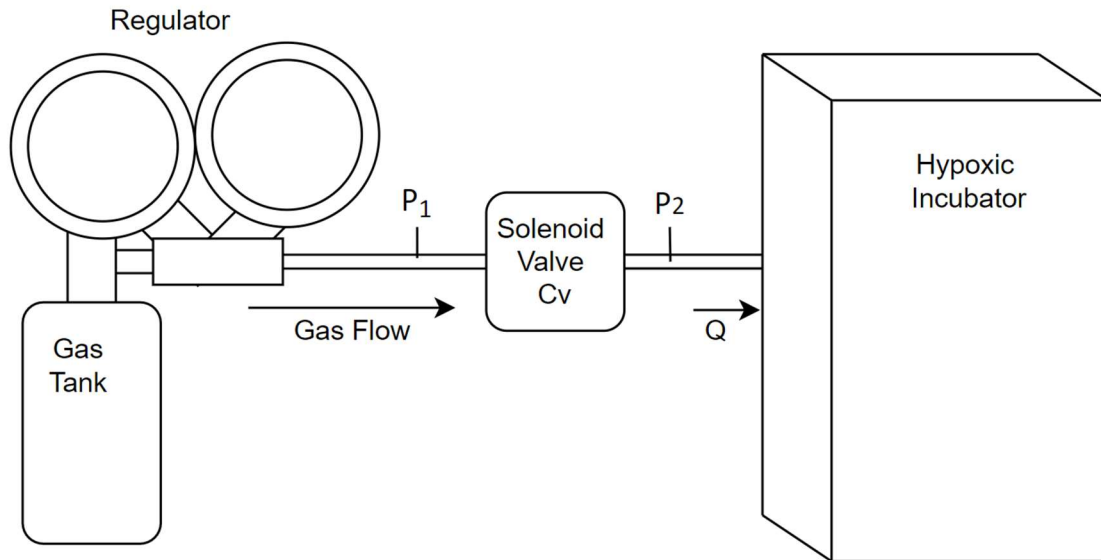


Figure 6. Hypoxic incubator control system components

A mathematical model was developed to analyze the effect on flow rate for each design consideration. This will be used to compare the effects that unique design concepts have on the output flow rate. The above figure depicts the path of airflow from the tank to the incubator. Both the regulator and the solenoid valve impact the flow rate into the incubator: the regulator controls the gas pressure into the solenoid valve, and the solenoid valve controls the final flow rate into the incubator. For each design consideration, either the C_v or the P_1 value will change which reflects a design change in the solenoid or the air pressure regulator, respectively.

The solenoid directly determines the flow of gas into the incubator; therefore, the main equation used in modeling was for flow rate through a solenoid. It is important to note that there are several equations for the flow rate of gas across a solenoid valve [1,2,3,4] and there is no standard across companies to measure flow rates of compressible fluids under varying input pressures and temperatures. However, the general equation below describes most of the flow rate equations we explored, where A is a coefficient that varies from company to company.

$$Q = A C_v \sqrt{\frac{P_2 \times \Delta P}{S G \times T}}$$

Equation 1. General flow rate equation.

A = Coefficient that varies by company

P_1 = Input Pressure

P_2 = Output Pressure

ΔP = $P_1 - P_2$

SG = Specific Gravity of gas

T = Absolute temperature

C_v = Flow coefficient specific to solenoid in imperial units, gpm

Equation 1 uses C_v , the flow coefficient in imperial units. However, the equation is frequently used in metric units. To avoid confusion, K_v represents the flow coefficient in metric units. This conversion is shown below.

$$K_v = 0.862C_v$$

Equation 2. Flow coefficient of a solenoid in metric units, m^3/hr , from imperial units, gpm.

Since coefficient A varies from company to company, there is no way to ensure the accuracy of our calculated flow rate value. However, if we use the same equation throughout the analysis of our design concepts, the percent differences between the flow rates calculated will be valid. Therefore, the percent differences calculated below are consistent regardless of the coefficient A .

The equation that we chose to use for flow rate of gas across a solenoid value is the following:

$$Q = 514k_v \sqrt{\frac{P_2 \times \Delta P}{SG \times T}}$$

Equation 3. Flow rate equation from *aclsolenoidvalves.com*.

Q = Flow rate, m^3/hr

P_1 = Input Pressure, bar

P_2 = Output Pressure, bar

ΔP = $P_1 - P_2$, bar

SG = Specific Gravity of gas, kg/m^3

T = Absolute temperature, K

K_v = Flow coefficient specific to solenoid, m^3/hr

The following values for flow rate were calculated using equation 3 above, assuming a standard temperature of $20^\circ C$ or $293.15 K$. Thorough hand calculations can be seen in appendix A and B. Currently, the output pressures for each gas are as low as possible without being turned off; for the oxygen and nitrogen gas regulators, it is assumed that the pressure is around 1 psi, whereas the output pressure for the CO_2 regulator is assumed to be 3 psi due to the imprecision in setting low output pressures with a larger range pressure regulator. The current solenoids have a flow coefficient of 4.8 gpm while the proposed replacement solenoid had a flow coefficient of 1.8 gpm.

Table 6. Comparison of current O₂ and CO₂ flow rates from Equation 3.

	Flow Rate (m ³ /hr)	Percent Difference from Baseline
O ₂ Current Flow Rate	31.03	Baseline
CO ₂ Current Flow Rate	45.84	47.73%

Table 7. Comparison of CO₂ flow rates due to design changes from Equation 3.

	Flow Rate (m ³ /hr)	Percent Difference from Baseline
CO ₂ Current Flow Rate	45.84	Baseline
CO ₂ Flow Rate with New Regulator	26.46	42.30%
CO ₂ Flow Rate with New Solenoid	17.19	62.50%

According to the values seen in Table 6, flow rate was greater in the problematic CO₂ branch than in the O₂ branch, demonstrating that the issue of gas level overshoot and gas level maintenance is at least partially due to the underlying issue of excessive flow rate. From our calculations, the flow rate through the current O₂ branch is 31.03 m³/hr while the flow rate through the current CO₂ branch is 45.84 m³/hr. This 47.73% increase from the flow rate observed in the O₂ branch supports the idea that higher flow rate results in the issue of gas level overshoot.

As seen Table 7, replacing the air pressure regulator yielded a flow rate of 26.46 m³/hr, a 42.30% decrease from the current system. If the solenoid valve was replaced, the flow rate would be 17.19 m³/hr, a 62.50% decrease from the current system. However, while the solenoid offers better performance in terms of decreasing flow rate at the conditions assumed during our analysis, additional considerations influenced our final decision. That is, with the current N₂ and O₂ branches, the 60 psi regulators allow for appropriate gas control. Therefore it could be argued that by replacing the 200 psi CO₂ regulator with the same 60 psi regulator this issue overshoot would likely be fixed. Additionally, according to the equation above, the flow rate decreases linearly with a decrease in output pressure.

This mathematical model further supported the idea that the issues of gas overshoot are caused by excessive flowrate in the CO₂ branch. Additionally, it helped us quantify the impact of different design ideas that attempt to mitigate that issue of overshoot.

7. Detailed Design

As discussed in the concept evaluation, the final design that was converged upon uses a regulator to facilitate more precise flow control. With respect to user interface, this design will leverage a touchscreen that will be attached to the Arduino that is currently in place.

Hardware Selection

The selected air pressure regulator used in the design is a 30-psig maximum pressure regulator from Grainger, Model KH1000. In comparison with the current pressure regulator, that has a maximum output pressure of 200-psig, this lower range will allow for finer control at lower psi levels. It was hypothesized that a large portion of the excessive flow rate and consequently, gas

overshoot, was due to imprecise control of the output pressure at low levels due to the high 200 psig range. Currently the other hypoxic incubator gas lines function between 0-1 psig outputs with 60 psig maximum regulators and as you can expect, it is extremely difficult to achieve that same 0-1 psig output with a vastly larger range pressure regulator. Conclusively, the switch to a 30-psi regulator enabled the operator to control the output pressure more precisely. The air pressure regulator can be seen in the following figure.



Figure 7. 15 psig Air Pressure Regulator, Model KH1000

The improvements suggested for the gas control were not limited to only interchanging the pressure regulator. Another possible method to regulate gas flow is through the replacement of the current solenoid valve with a lower flow coefficient solenoid, as introduced in the concept generation section of this document.

The solenoid and pressure regulators are independent components that contribute to the output gas flow and by replacing both elements, the lowest and most precise control of gas levels can be achieved. Despite these benefits we determined that the replacement of only the pressure regulator is sufficient to generate an improved CO₂ gas flow rate to a rate that is similar to the functional O₂ branch. If the issue of gas overshoot is still present after testing the final design prototype, the solenoid could be replaced to address any issues or limitations of our final solution. This iterative process would allow for a more conservative approach with regard to our budget.



Figure 8. Adafruit ILI9341 Touchscreen Display

The touchscreen used in this design is the Adafruit ILI9341, depicted in Figure 8. The touchscreen user interface was selected primarily based on its intuitiveness to the operator as well as its ability to relay information.

User Interface Design

Since the Arduino code is based in the C/C++ language, the fundamental programming for this specific touchscreen has already been completed and made available through C/C++ libraries. The code for our touchscreen will heavily rely on the <Adafruit_ILI9341.h> and <Adafruit_GFX.h> libraries that are available online. The first library is necessary as this allows the Arduino to communicate with the ILI9341 touchscreen display. Whereas the second library provides a plethora of functions that allow for the control of graphics on the display. Some of these functions include the ability to change the screen color, create shapes, display text, and even create interactive buttons. The screens mentioned in Table 9 and depicted in Figure 9 was created by leveraging these pre-made functions. All the libraries utilized in our code and a brief summary of their function can be seen in Table 8.

Table 8. Libraries and their Functions

Library:	Function:
SPI.h	Allows for quick communication with one or more peripheral devices over short distances
Adafruit_GFX.h	Provides common syntax and set of graphics functions for all LED and OLED displays and LED matrices.
Adafruit_ILI9341.h	Provides functions to interface with the touchscreen
XPT2046_Touchscreen.h	Provides touch-sensing functionality for the display
Cozir.h	Provides functions to interface with CO2 sensor
DallasTemperature.h	Provides functions to interface with temperature sensor.

The touchscreen was coded to include several functions, these can be seen below in addition to a pictorial representation of each screen:

Table 9. Functionality of Different Screens on the Interface

Screen #	Description
0	This screen provides the operator with the ability to control the output pressure of the gas regulators. The operator can choose to open and close specific solenoids to hone in on proper pressure levels. All solenoids must be closed to return to Page 1.
1	This screen will allow the operator to read the current status of the Hypoxic Incubator Control System and change the O2 setpoint for the control system. It will also include buttons to navigate to other screens.
2	This screen will convey the total time the gas level has deviated from the setpoint after setting at the setpoint for a specific period of time. For example, if the gas level deviates from within .5% of the O2 setpoint this screen will display that data.

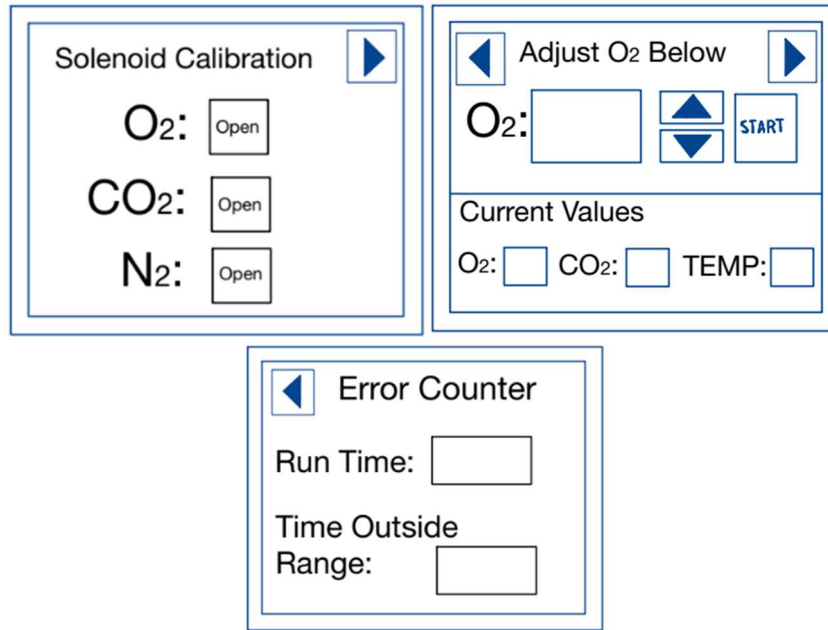
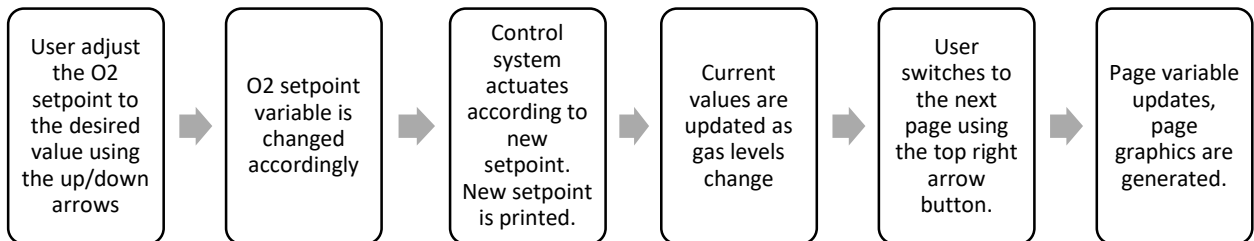


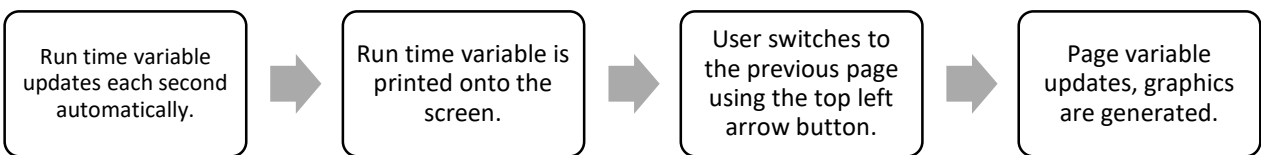
Figure 9. Layout of Different Screens on the Interface

As seen in Figure 9, the user has access to multiple functionalities within each screen. Below is a flowchart that reveals how a user may interact with the interface to change the settings of the incubator.

First Page:



Second Page:



The following code block represents the portion of the code that generates the page graphics for the “Adjust O₂ Below” screen of the touchscreen. This is done through C++ switch statements to switch between different pages when the page variable is updated. The “case 11” block is executed when the pageID variable is equal to the value 11. This calls the function pageOne() which in turn executes all of the draw functions for the different graphics on the page. Then it updates the pageID variable to 12 such that on the subsequent loop of the code it executes the

“case 12” block. The primary function of the “case 12” block is to wait for user to press the buttons on the screen. These buttons are defined by their coordinates on the touchscreen and if touched, execute the subsequent code. For example, if the button to increase the O₂ setpoint is pressed, the O2Setpoint variable is updated.

```
switch (pageID) {
  case 11:
    pageOne();
    pageID = 12;
    break;
  case 12:
    while (!ts.touched());
    delay(100);
    p = ts.getPoint();
    sp = getScreenCoords(p.x, p.y);

    if (sp.x >= 170 && sp.x <= 210) {
      if (sp.y >= tft.height() - 80 && sp.y <= tft.height() - 60) O2Setpoint += 0.1;
      if (sp.y >= tft.height() - 50 && sp.y <= tft.height() - 30) O2Setpoint -= 0.1;
      tft.fillRect(20, tft.height() - 80, 140, 50, ILI9341_WHITE);
      tft.setCursor(45, tft.height() - 80 + 12);
      tft.setTextSize(4);
      tft.setTextColor(ILI9341_BLACK);
      tft.print(O2Setpoint);
    }

    if (sp.x >= tft.width() - 30 && sp.y <= 30) {
      pageID = 21;
    }
    break;
}
```

Schematic Design

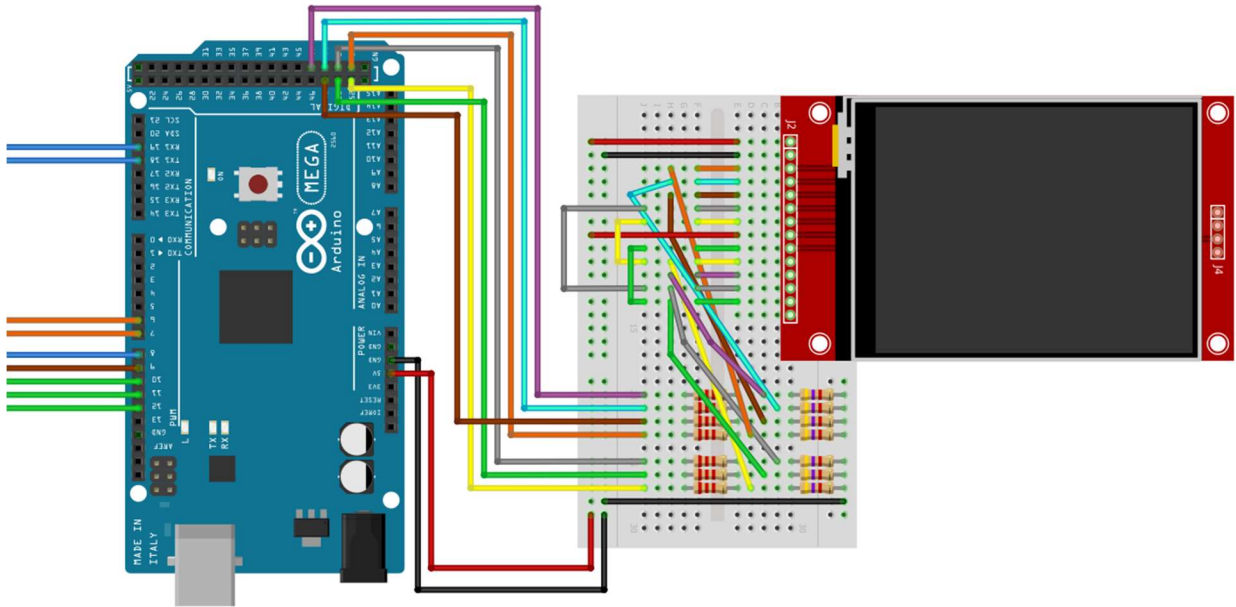


Figure 10. Connections of ILI9341 touchscreen onto Arduino Mega

The figure above represents the schematic diagram required to connect the ILI9341 touchscreen with the Arduino MEGA. It should be noted that the previous design of the control system utilized two Arduino microcontrollers: a UNO that controlled both the relays and sensors for the CO₂ and temperature while the MEGA controlled the relay and sensors for the O₂ and N₂. In our design, all the sensors and relays have been concatenated onto the Arduino MEGA. This reduces the need for serial transfer of data between the boards which can easily convolute the control process. Additionally, by removing the UNO, all the code for control system is now located within one file making the program easier to maintain and update.

The documentation for the ILI9341 touchscreen states that the screen functions with 3.3V logic. Therefore, the 5V logic that the Arduino Mega outputs has the potential to damage the internal touchscreen electronics over time. For this reason, a voltage divider was created using 2.2 kOhm and 4.7 kOhm resistors on a breadboard to reduce the output voltage of the Arduino to within the operating limits of the ILI9341.

One concern with implementing this touchscreen was that the pins required would interfere with the pins that are already used to connect the sensors and solenoids to the Arduino. However, as seen in our schematic diagram, touchscreen connections do not conflict with the wires already present on the system due to ample pin connections. Cost estimation for these parts can be seen in the budget table provided in the Project Management section of this document.

Gas Control System

In this control system, O₂ and CO₂ setpoints are maintained through the actuation of their respective solenoids to upregulate the gas when it is beneath the setpoint. On the other hand,

the solenoid controlling the input of N₂ is solely used to downregulate either gas when their levels are above the setpoint.

The control code to maintain both the O₂ and CO₂ setpoints will reflect a proportional control system. In doing so the open time for each solenoid will be proportional to the error between the reading of gas chamber levels and the desired setpoint. For example, the open time for the CO₂ solenoid will be much longer when the gas level is further away from the setpoint as opposed to the open time when the gas level is closer to the setpoint. The general equation for the open time can be seen in equation 4.

$$\text{Open Time} = K_p * |\text{Gas Level} - \text{Gas Setpoint}|$$

Equation 4. General equation for Open Time

The parameter K_p is proportional gain constant and must be tuned during testing to produce an optimal open time. The proportional control code for O₂ can be seen in the code block below:

```
void controlO2() {
  if (O2Data < O2Setpoint - O2tolerance +.02) {
    if (O2state == false && millis() - O2closetimestamp > closeTime) {
      if (O2Data < prevO2Data * 1.02 && O2Data > prevO2Data * 0.98) {
        digitalWrite(O2Sole, LOW);
        O2opentimestamp = millis();
        O2openTime = computeOpenTime(O2Data, O2Setpoint, O2Kp);
        O2state = true;
      }
    } else if (O2state == true && millis() - O2opentimestamp > O2openTime) {
      digitalWrite(O2Sole, HIGH);
      O2closetimestamp = millis();
      O2state = false;
    }
  } else if (O2Data > O2Setpoint + O2tolerance) {
    digitalWrite(O2Sole, HIGH);
    if (N2state == false && millis() - N2closetimestampO2 > closeTime) {
      if (O2Data > prevO2Data * 0.98 && O2Data < prevO2Data * 1.02) {
        digitalWrite(N2Sole, LOW);
        N2opentimestampO2 = millis();
        N2openTimeO2 = computeOpenTime(O2Data, O2Setpoint, N2KpO2);
        N2state = true;
      }
    } else if (N2state == true && millis() - N2opentimestampO2 > N2openTimeO2) {
      digitalWrite(N2Sole, HIGH);
      N2closetimestampO2 = millis();
      N2state = false;
    }
  } else {
    digitalWrite(O2Sole, HIGH);
  }
}
```

There are two main if-statements within this control function. The first if-statement checks whether the current O₂ reading is above the setpoint, and the second if-statement checks if the reading is above the setpoint. As mentioned previously, if O₂ is beneath the desired setpoint then the code will actuate the O₂ solenoids. Nested within these statements are another pair of if-statements that check if the solenoid has been closed or open for the set amount of time. Once the solenoids pass the defined open or close time they switch to the opposite state. The control code for the CO₂ branch is the same with some minor changes such as variable names. The implementation of this control function can be seen in Appendix E.

8. Prototype Manufacturing Plans

Since this project is a revision of the current control system, many of the prototyping efforts relate to replacing problematic components, testing the code that controls solenoid actuation, and developing a user interface. Consequently, the prototyping process that will be outlined below will reflect this as opposed to more traditional product prototyping.

General Prototyping Procedure

As discussed briefly in the detailed design portion, the prototyping for this control system consists of an iterative process whereby the group replaced a component (starting with the air pressure regulator), collected data in accordance with the defined test plans, and analyzed that data to determine whether the overshoot issue had been resolved. If the issue was still present, the design was iteratively changed to replace additional components or to adjust the control code. The prototyping of the critical design adhered to the following process instructions to limit any potential error in data:

1. Stop gas flow into the hypoxic incubator by closing the air pressure regulator.
2. Disconnect the current gas control component.
 - a. Uncouple any gas line connected to the component.
3. Connect the desired gas control component or edit the control code.
4. Run the HICS tests defined in the Test Plan section.
5. Record the data.

Regarding the prototyping of the screens depicted previously, the code for each screen was deconstructed into base components. For example, to develop the first screen shown in Figure 9 several smaller components must be handled: the creation of rectangles, displaying text, and implementing navigation buttons. All these components can be created by predefined functions found within the Adafruit_GFX library. After learning how to use these functions and their required parameters, each of the screens were created through combinations of each component. Deconstructing each of these screens into their base components allowed for a better understanding of the code as well as being much easier to troubleshoot.

Bill of Materials

The complete list of materials required for the redesigned control system can be seen in the Bill of Materials, Table 10. The regulator was attached to the CO₂ tank as shown in Figure 11, whereas the touchscreen, jumper wires, and resistors were attached to the breadboard in accordance with the schematic diagram detailed in Figure 10. In addition, the laptop connected to the hypoxic incubator was used to receive the data, a continuous stream of chamber gas percentages. This

data helped evaluate the efficacy of our design changes in accordance with the test plans defined in the subsequent section.

Table 10. Bill of Materials

Item #	Part #	Qty	Name	Material	Source
1	4Z474	1	Gas Tank Regulator	Brass, Chrome Plated Brass	Grainger
2	3DTN3	1	Gas Regulator Hose Barb	Stainless Steel	Grainger
3	ED-DP_L20	1	Jumper Wires	Copper	Amazon
4	3-01-1433	1	Arduino - TFT LCD Display	FR4	Amazon
5	CF18JT2K20	7	2.2 kOhm Resistor	Carbon Film	Digikey
6	CF18JT4K70	7	4.7 kOhm Resistor	Carbon Film	Digikey



Figure 11. Air pressure regulator assembled onto the CO2 gas tank

Resources and Training

To facilitate our prototyping process, access to the hypoxic incubator and consequently access to the MPS lab was required. All group members performed a series of training regarding proper lab protocol. Within this training several topics were covered including but not limited to biosafety cabinet training, general biosafety operating procedure, and proper response to biohazards. This training helped mitigate any potential lab hazard risk while the group prototyped the proposed design changes. Access to the machine shops was not required.

Manufacturing Process: Touchscreen Stand

In continuation with assembling various parts for the hypoxic incubator, different tools and materials were used to build the design for our touchscreen display as seen in Figure 13. The measurements for the touchscreen can be seen in Figure 12. This prototype stand, Figure 14, was developed to help highlight several design considerations that will be pertinent within the final design for the stand. Namely, the optimal viewing angle and clearance areas required to access

the ports on the touchscreen. An electric screwdriver was used to insert the screws along with a drill saw that allowed for the cutting of the material.

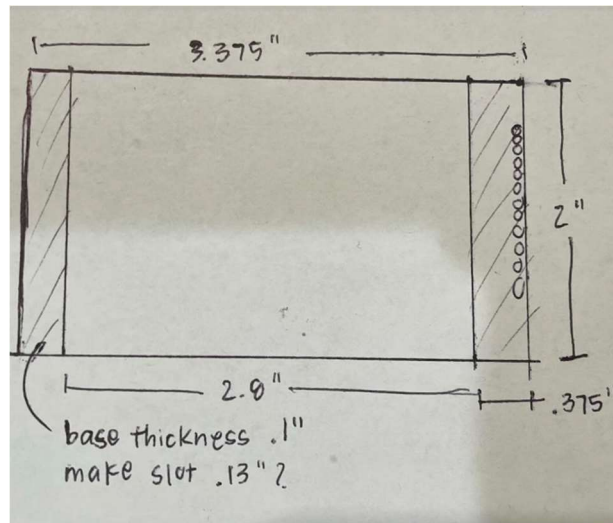


Figure 12. Measurements of touchscreen display



Figure 13. Tools used to build touch screen display



Figure 14. Touchscreen display stand

9. Test Plans

The two crucial engineering specifications deduced from our customer requirement analysis and House of Quality were range of possible set points and deviation over time, where a higher range of set points and low deviation over time is desirable. To test these specifications, the hypoxic incubator was run at different O₂ setpoints for different durations while chamber gas levels were recorded. Furthermore, the usability of the interface was quantified using a questionnaire and rating scale. The tests below build upon the tests documented in the original hypoxic incubator report.

1 Hour and Overnight Time Outside of Range Test Protocol

Both O₂ and CO₂ gas levels must fall within their specified tolerances for at least 98% of the total run time. Time started once the setpoint was reached as defined in the setpoint test above and ended at either 1 hour or at least 12 hours for overnight testing. At least one team member was present to start and end the test but did not need to be present while data was being collected. Initially, the 1-hour test was to be performed for O₂ values of 0.2%, 2%, 5% and 10%, while the overnight test was to be performed for 0.2% and 10%. However, the lowest achievable value for O₂ is 1% due to the limitations of diffusion of room air into the chamber. Limitations are further discussed in the discussion section of this report. Both the stabilization test and the overshoot tests were performed during the same incubator run. To determine the percentage of time outside of the range, data was analyzed in Excel.

This test followed the subsequent protocol:

1. Open the incubator door and allow the chamber to reach ambient levels of O₂ and CO₂

- a. These values are about 20.9% and 0.4% for O₂ and CO₂, respectively
2. Set the incubator to the desired testing setpoint
3. Simultaneously press the start button to start the runtime tracker and close the incubator door
4. Allow the incubator to run of the specified run time
5. After the run time has elapsed, record the total error time
6. Record the percentage of time outside of range
 - a. Percent time outside of range = Error time / Total Run Time

One sample t-test analysis was used to demonstrate that the average O₂ and CO₂ levels for each run fell within the bounds. One-sample t-tests were performed with the following hypotheses:

H₀ = The average percent of time outside range is equal to upper or lower bound

H_A = The average percent of time outside range is less than or greater than the bound

The goal of this one-sample t-test is to show that the average percentage of time outside of the acceptable range falls within our acceptable target range of 2%.

Time to Stabilization Test

The O₂ and CO₂ gas levels must reach their setpoint range within 5 minutes of closing the chamber. To test this, one team member will open the chamber to allow natural air inside. The timer was started after O₂ setpoint was set upon closure of the chamber, and O₂ and CO₂ values were detected by the existing sensors as percentages and stored in an excel sheet through the existing program on the Arduinos at a rate of 1 point every 3 seconds. This was consistent with the previous report. The gas levels were considered stabilized when they fell within their specified setpoint tolerances. This test was run while testing overshoot as time to reach setpoint could easily be measured when starting up the incubator. Thus, this test was also run at the same setpoints as the test above.

This test followed the subsequent protocol:

1. Open the incubator door and allow the chamber to reach ambient levels of O₂ and CO₂
 - a. These values are about 20.9% and 0.4% for O₂ and CO₂, respectively
2. Set the incubator to the desired testing setpoint
3. Simultaneously press the start button to start the runtime tracker and close the incubator door
 - a. Ensure that regulator output pressures are within 1psi
4. Monitor the gas levels
5. When the gas levels reach the acceptable range, record the total time
 - a. Note: These times may be different for O₂ and CO₂

Range of Setpoints Test

To test the range of possible setpoints, the overshoot and stabilization tests must pass for all O₂ set points between 0.1% and 10%. If a test does not pass, the test must be run again until it passes to pass the range of setpoints test.

Interface Usability Test

To assess interface usability, 5 members of the MPS lab were asked to operate the hypoxic incubator after being given a brief description of the operation process and then rate the interface on a 1 to 5 scale shown below:

Rate the usability of the hypoxic incubator control system interface on a scale of 1 to 5.

- 1 = Non-operable
- 2 = Operation is very confusing
- 3 = Operation is slightly confusing
- 4 = Easy to operate, little confusion
- 5 = Easy to operate, no confusion whatsoever

The above tests required full access to the MPS lab, and all instrumentation needed was present within the existing hypoxic incubator control system, including gas sensors and a computer to record data. Interface usability testing required coordination with frequent MPS lab users as subjects. Below is a summary table of the planned tests. Not all tests were run due to limitations discussed later.

Table 11. HICS Tests, testing criteria and requirements for testing.

Test Name	Number of Tests (Minimum)	Testing Metric	Pass/Fail Criteria	Requirements for Testing
Range of Setpoints	1	Pass/Fail of tests below	All tested O ₂ values between and including 0.1% to 10% pass <i>Time to Stabilization</i> and <i>Overshoot</i> tests	MS Lab access, Several hours in lab
Time to Stabilization Test	18	Time to stabilization, t	t < 5 minutes	MPS Lab access, 1+ Hours in lab
1 Hour Deviation from Setpoint	12	Time outside of range	O ₂ within 0.1% and CO ₂ Within 0.5%* for at least 98% of runtime	MPS Lab access, 1+ Hours in lab
Overnight Deviation from Setpoint	6	Time outside of range	O ₂ within 0.1% and CO ₂ Within 0.5%* for at least 98% of runtime	2 consecutive days of lab access, 2+ hours in lab
Interface Usability	5	Questionnaire rating score, 1-5	Average score ≥ 4	MPS Lab access, 1+ Hours in lab, 5 regular MPS lab users as test subjects

*The percentage values here represent the percentage of gas in the hypoxic incubator, not a percent of the setpoint.

10. Testing Results

Our team encountered several difficulties that limited our testing time; however, testing proved invaluable to detect and fix issues with the code and operating procedures. Due to time limitations, we were not able to perform the overnight tests we initially planned. One overnight test was captured for 10% O₂.

O₂ Setpoint values below 1% required constant flow of N₂ into the chamber with no solenoid close time. This is due to the large difference in O₂ partial pressures between chamber air and room air. Leaving the N₂ tank open proved costly and impractical, so the lowest setpoint of 0.2% to be tested was abandoned. Three tests for each of the remaining O₂ setpoints were run, and time to setpoint and deviation from setpoint are displayed in Tables 12, 13 and 14. Graphs displaying the entirety of each run are included in appendix D.

Table 12. 2% O₂ time to setpoint and deviation test results.

Test	Time to Setpoint (min)	Pass/Fail	Percent of Runtime Spent Outside of Range (%)	Pass/Fail
Test 1	5.56 ^[1]	Fail	3.5 ^[2]	Fail
Test 2	6.05 ^[1]	Fail	3.7 ^[2]	Fail
Test 3	2.3	Pass	0.04	Pass
Average	4.63		2.41	

^[1] Time to setpoint was high because of gas tank adjustment during the first portion of testing.

^[2] Percent outside of runtime is out of specification due to a bug in code during test.

Table 13. 5% O₂ time to setpoint and deviation test results.

Test	Time to Setpoint (min)	Pass/Fail	Percent of Runtime Spent Outside of Range (%)	Pass/Fail
Test 1	3.25	Pass	3.1 ^[2]	Fail
Test 2	6.5 ^[1]	Fail	5 ^[2]	Fail
Test 3	3.8	Pass	0.01	Pass
Average	4.52		2.70	

^[1] Time to setpoint was high because of gas tank adjustment during the first portion of testing.

^[2] Percent outside of runtime is out of specification due to a bug in code during test.

Table 14. 10% O₂ time to setpoint and deviation test results.

Test	Time to Setpoint (min)	Pass/Fail	Percent of Runtime Spent Outside of Range (%)	Pass/Fail
Test 1	2.4	Pass	2.3 ^[1]	Fail
Test 2	1.9	Pass	0.8	Pass
Test 3 (12Hr)	2.5	Pass	1.0	Pass
Average	2.27		1.37	

^[1] Percent outside of runtime is out of specification due to a bug in code during test.

Several issues in our system were solved while we performed the tests above. A bug in the CO₂ loop code was detected and eliminated half-way through testing; this bug contributed to the failed deviation from setpoint tests. The effects of this bug can be seen in the plots for each test in appendix D as the spikes in CO₂ cause O₂ to fall out of the range. Additionally, the regulators were adjusted during the beginning of each run that failed time to setpoint. At this point in our testing, the screen did not work, so we could not use our regulator calibration screen to calibrate before running the tests. We also discovered that output pressure must be tuned precisely to achieve our time to setpoint limit of 5 minutes. However, after each change was implemented, the final test for all three setpoints passed both times to setpoint and deviation from range tests.

Tables 15, 16, and 17 show average and standard deviation gas values during the test after the incubator has reached the setpoint. All averages were within the bounds set. These values were used in the subsequent statistical analysis.

Table 15. Average gas values during 2% O₂ test.

Test	Average CO ₂	Standard Deviation	Average O ₂	Standard Deviation
Test 1	4.947	0.248	2.045	0.062
Test 2	4.926	0.280	2.047	0.0631
Test 3	4.875	0.0746	2.004	0.0427

Table 16. Average gas values during 5% O₂ test.

Test	Average CO ₂	Standard Deviation	Average O ₂	Standard Deviation
Test 1	4.836	0.0849	5.082	0.00461
Test 2	5.023	0.210	5.0922	0.0591
Test 3	4.914	0.252	5.091	0.0584

Table 17. Average gas values during 10% O₂ test.

Test	Average CO ₂	Standard Deviation	Average O ₂	Standard Deviation
Test 1	4.895	0.074	9.990	0.089
Test 2	4.981	0.179	10.002	0.066
Test 3	4.982	0.179	10.019	0.470

One-sample t-tests were performed on O₂ and CO₂ values for each run to determine whether the mean O₂ and CO₂ percentages of each test run fell within desired ranges. Data was compared to a hypothesized mean chosen based on the acceptable range value closest to the average in question; for example, the lower bound of 4.5 was closest to the average CO₂ percentage of 4.947 for the duration of Test 1 under 2% O₂. P-values and the hypothetical means used in each comparison are shown in Tables 18, 19 and 20 below. P-values less than .05 indicate that there is a significant difference between the average gas percentage in question and the hypothetical value listed. P-values greater than .05 indicate that there is not enough evidence to support the alternative hypothesis that the mean gas percentage is different than the hypothesized mean. At

10% O₂, tests 2 and 3 both showed no statistically significant difference from the setpoint; therefore, the setpoint was used instead of a lower or upper bound.

Table 18. T-test statistics for tests at 2% O₂.

Test Number	Hypothetical Mean	p-Value for CO ₂	Hypothetical Mean	p-Value for O ₂
Test 1	4.5	<.01	2.1	<.01
Test 2	4.5	<.01	2.1	<.01
Test 3	4.5	<.01	2.1	<.01

Table 19. T-test statistics for tests at 5% O₂.

Test Number	Hypothetical Mean	p-Value for CO ₂	Hypothetical Mean	p-Value for O ₂
Test 1	4.5	<.01	5.1	<.01
Test 2	5.5	<.01	5.1	<.01
Test 3	4.5	<.01	5.1	<.01

Table 20. T-test statistics for tests at 10% O₂.

Test Number	Hypothetical Mean	p-Value for CO ₂	Hypothetical Mean	p-Value for O ₂
Test 1	4.5	<.01	9.9	<.01
Test 2	4.5	<.01	10	0.385
Test 3	4.5	<.01	10	0.383

The interface usability test returned to an average value of 4.4 among the 5 subjects that used the control system touchscreen. Feedback given shown in Table 21 was considered when finalizing the interface design; for example, Subject 1 suggested that background colors change when a run has started.

Table 21. Interface Usability Test Results.

Subject	Score	Feedback
1	4	Add colors to indicate whether the system is running or not, add pause button
2	4	Screen turned white while operating, had to restart
3	4	Bugs when opening screen
4	5	N/A
5	5	N/A

11. Conclusions

The results show that the redesigned control system for the hypoxic incubator can maintain the hypoxic conditions required to cultivate mammalian cells.

Testing has shown the average time to reach setpoints 2%, 5%, and 10% to be 2.41, 2.7, and 1.3 minutes, respectively. The deviation from range for the setpoints 2%, 5%, and 10% was 4.63, 4.52, and 2.27 minutes, respectively. Each of these averages fall within the desired range of +/- 0.1%. One-sample t-tests showed that all gas percentage means were within the specified ranges. Not all tests passed the time to setpoint or deviation from range tests; however, a test from each setpoint passed under the current conditions of the control system. This is due to the fact that the time to setpoint and deviation from range is highly sensitive to the output pressure of the CO₂ regulator. Essentially, there was a "sweet spot" near 0 psi that would allow for quick time to reach setpoints but also small deviations from the setpoint after it has been reached. Any lower would result in slower time to reach setpoints and any higher would result in more deviation from the setpoint. On the other hand, it should be noted that the tests that failed show generally consistent O₂ and CO₂ values aside from the CO₂ spikes caused by a bug in the program. Only one 12-hour test was successfully recorded – this test at 10% passed both former criteria.

Our range of setpoints test failed since our hypoxic incubator was unable to reach and maintain setpoint of 0.2%. During testing, the lowest setpoint that hypoxic incubator could sustain was 0.5%. The implications of this failed test will be discussed in detail below.

Lastly, the average score from our interface usability test was 4.4, indicating that our system was easy to use with little to no confusion. Previously, usage of the system would require the operator to have basic knowledge of coding in the sense that any changes to the desired setpoint would be made through manipulating variables within the Arduino code. In contrast, the touchscreen allows the operator to manipulate the setpoint and view chamber conditions without ever having to open any code.

12. Discussion

The issue of gas overshoot seen in the previous design – specifically that of CO₂ – has been mitigated. The incubator has the capacity to accept a user-defined O₂ setpoint and maintains that value within 0.1% under the right conditions. CO₂ stays within 0.5% of its constant setpoint of 5%. This is a vast improvement in comparison to the previous design in which CO₂ endured large fluctuations well outside the acceptable range.

Not only does the addition of a touchscreen enhance the usability of the system but it also adds several functions that were not present prior. For example, the operator can now manually open the solenoids such that it is easier to accurately adjust the output pressure of the gas tanks. During testing, the operator can also view the total runtime as well as the total time spent outside of the range. While the latter metric should not be significant as shown in our testing, it may be valuable information for the operator to have when cultivating cells. Perhaps the biggest advantage of touchscreen is that it allows the hypoxic incubator to function as a standalone device. If the operator does not need to record the gas values throughout the test, then the hypoxic incubator can be run without an accompanying computer.

While the main concerns of gas overshoot and usability have been addressed, the redesigned control system still has some limitations. As mentioned previously, the control system was unable to reach 0.2% O₂ within the chamber, failing to pass our engineering specification of being able to control the setpoint within 0.1-10%. Even with the N₂ solenoid constantly open to downregulate

the O₂ levels within the chamber, the lowest achievable O₂ setpoint was 0.5%. Lower setpoints could most likely be achieved by increasing the output pressure from the N₂ tank, but the goal of the system was to be able to achieve all setpoints without having to adjust the output pressure. It was believed that having to adjust the output pressure to achieve different setpoints would make the operation process much more convoluted for operators. More importantly, the ability to reach the 0.2% setpoint is not a pressing concern as the typical usage for the incubator will be around 5% O₂. Secondly, there is an issue with the touchscreen occasionally turning off during testing. While the cause of this issue is not necessarily known, it has been hypothesized that there may be due to how the O₂ solenoid interacts with the rest of the system. On multiple occasions the touchscreen has turned white when the O₂ solenoid opens to release gas, thus leading us to believe that the solenoid may be drawing too much current. This inhibits any visual information from being displayed on the touchscreen; however, the values are still displayed on the connected computer. Finally, to create a more efficient gas control system it may be advantageous to implement a full PID controller for the actuation of the solenoids. This will have the direct effect of reducing the gas usage and consequently the cost associated with using the incubator. In the future, subsequent redesigns of this control system may aim to address these limitations.

13. Instructions for Use

Operation of Incubator with Touchscreen Procedure

Summary:

These are instructions used for setting up the hypoxic incubator with the touchscreen display and loading samples into the incubator.

Instructions:

1. Open the gas tank by rotating the knob counterclockwise until it is fully open. Then rotate the knob clockwise for a quarter turn.
 - a. Repeat for all gas tanks.
2. Plug the power cable into the barrel connector of the Arduino MEGA. The touchscreen should turn on.
3. Turn on the 12V power supply connected to the solenoids.
4. If the correct code is not already uploaded, follow the **Upload Arduino Code Procedure**. This step is only necessary when switching between using the touchscreen and laptop to control the setpoint.
 - a. Correct code for this procedure: MEGA_TS.ino
5. Calibrate the touchscreen according to the instructions on the screen.
6. If the air pressure regulators have been reset, calibrate them according to the **Output Pressure Calibration Instructions**
7. Set the desired O₂ setpoint by using the up and down arrows.
8. Open the incubator door and load desired samples onto the shelves of the incubator.
9. Close the incubator door.
10. Press the green start button to begin the incubator process.
11. After the desired incubation time has been desired, press the red stop button to end the incubator process.
12. Open the incubator door and remove the samples.

13. Close the gas tanks by turning the knobs clockwise until closed. Regulators do not have to be closed.

Operation of Incubator without Touchscreen Procedure

Summary:

These are the instructions used for setting up the hypoxic incubator with a laptop and loading samples into the incubator

Instructions:

1. Open the gas tank by rotating the knob counterclockwise until it is fully open. Then rotate the knob clockwise for a quarter turn.
 - a. Repeat for all gas tanks.
2. Plug the USB cable into the Arduino MEGA and the lab laptop. The touchscreen should turn on.
3. Turn on the 12V power supply connected to the solenoids.
4. If the correct code is not already uploaded, follow the **Upload Arduino Code Procedure**. This step is only necessary when switching between using the touchscreen and laptop to control the setpoint.
 - a. Correct code for this procedure: MEGA_NoTS.ino
5. Calibrate the touchscreen according to the instructions on the screen.
6. If the air pressure regulators have been reset, calibrate them according to the **Output Pressure Calibration Procedure**
7. Set the desired O₂ setpoint by using the up and down arrows.
8. Open the incubator door and load desired samples onto the shelves of the incubator.
9. Close the incubator door.
10. Press the green start button to begin the incubator process.
11. After the desired incubation time has been desired, press the red stop button to end the incubator process.
12. Open the incubator door and remove the samples.
13. Close the gas tanks by turning the knobs clockwise until closed. Regulators do not have to be closed.

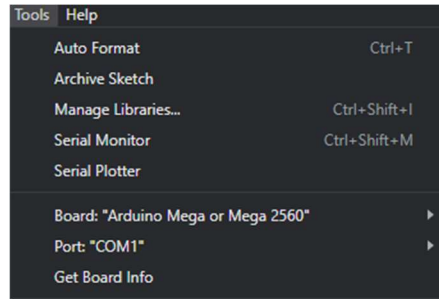
Upload Arduino Code Procedure

Summary:

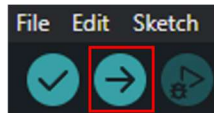
These are the instructions used for uploading the code to the hypoxic incubator.

Instructions:

1. Plug the USB cable into the Arduino MEGA and the lab laptop.
2. Navigate to the "Hypoxic Incubator Code" folder located on the desktop of the lab laptop.
3. Open the desired code in a file
4. Ensure that the correct "Board" and "Port" are selected underneath the "Tools" tab in the top left.



5. Upload the code to the Arduino by pressing the button in red shown below.



6. Once the code is finished uploading, close the Arduino IDE.

Output Pressure Calibration Instructions

Summary:

These are the instructions for calibrating the output pressure of the gas regulators after the regulators have been reset (fully closed).

Instructions:

1. Navigate to the regulator calibration screen by pressing the "Left" arrow located in the top left of the touchscreen.
2. To calibrate a specific gas regulator, open the respective solenoid by pressing the button on the touchscreen.
 - a. i.e., to calibrate the O₂ regulator, press the "Open" button next to O₂ on the touchscreen
3. Slowly turn the air pressure regulator knob clockwise to increase the output pressure.
 - a. The operator should aim for the following regulator pressures to ensure that the incubator functions properly:
 - b. O₂: Note that the output pressure for O₂ is extremely low. Rule of thumb for calibrating O₂ pressure is to increase it such that the dial hand is no longer resting on the pin; a faint hissing sound should be heard while the O₂ solenoid is open for calibration.



c. N₂:



d. CO₂:



14. References

- Atkin, B. (2005). Solenoid valve control system (U.S. Patent No. USRE41299E1). U.S. Patent and Trademark Office. <https://patents.google.com/patent/USRE41299E1/en>
- Burgess, P. (n.d.). Adafruit GFX Graphics Library. Adafruit Learning System. Retrieved March 14, 2023, from <https://learn.adafruit.com/adafruit-gfx-graphics-library/overview>
- Calculate flow coefficient KV of solenoid valve: InstrumentationTools. Inst Tools. (2019, April 19). Retrieved March 14, 2023, from <https://instrumentationtools.com/calculate-flow-coefficient-kv-of-solenoid-valve/>
- EXPLORIR®-WV 20% CO2 Sensor. CO2 Meter. (n.d.). Retrieved March 14, 2023, from <https://www.co2meter.com/products/cozir-wrv-20-percent-co2-sensor>
- Flow calculation. ACL. (2017, February 10). Retrieved March 14, 2023, from <https://www.aclsolenoidvalves.com/CatalogoAcl/flow-calculation/>
- Helfrich, S., Jones, M. (2022). (rep.). Hypoxic Incubation Chamber. San Luis Obispo, California: California Polytechnic San Luis Obispo.
- Lim, S. (2016). Cancer cell enrichment system (U.S. Patent No. 20180100134A1). U.S. Patent and Trademark Office. <https://patents.google.com/patent/US20180100134A1/en>
- Smirnov, A. (2019). Multi-gas mass flow controller and method (U.S. Patent No. US11041749B1). U.S. Patent and Trademark Office. <https://patents.google.com/patent/US11041749B1/en>

15. Appendix

Appendix A. Hand calculations for O₂ Flow Rate

$$Q_n = 514 K_v \sqrt{\frac{P_2 \cdot \Delta P}{SG \cdot T}}$$

Assumptions:

$$\Delta P < \frac{P_1}{2}$$

$$SG = 1.1044 \text{ for } O_2$$

$$T = 293.15 \text{ K}$$

$$P_2 = 14.7 \text{ psi (1.0 bar)}$$

$$\Delta P = 1 \text{ psi (0.0689 bar)}$$

$$C_v = 4.8 \frac{\text{gal}}{\text{min}}$$

$$K_v = 4.8 \frac{\text{gal}}{\text{min}} \cdot 0.862 \frac{\text{m}^3}{\text{gal} \cdot \text{min}}$$

$$K_v = 4.138 \frac{\text{m}^3}{\text{hr}}$$

$$Q_n = 514 \left(4.138 \frac{\text{m}^3}{\text{hr}} \right) \sqrt{\frac{1 \text{ bar} \cdot 0.0689 \text{ bar}}{1.1044 \frac{\text{kg}}{\text{m}^3} \cdot 293.15 \text{ K}}}$$

$$Q_n = 31.03 \frac{\text{m}^3}{\text{hr}}$$

Appendix B. Hand Calculations for CO2 Flow Rate

For CO₂:

Assumptions

$\Delta P = 3$ psig (0.2068 bar) With new Regulator: $\Delta P = 1$ psig (0.0689 bar)

$P_2 = 14.7$ psi (1.0 bar), or 0 psig

$C_v = 4.8 \frac{m^3}{hr}$

With Alternative Solenoid: $C_v = 1.8$

$SG = 1.5189 \frac{m^3}{m^3}$ for CO₂

$T = 293.15$ K (20°C)

current solenoid: $K_v = 4.138 \frac{m^3}{hr}$

alternate solenoid: $K_v = 1.552 \frac{m^3}{hr}$

Current Q

$$Q_n = 514 \left(4.138 \frac{m^3}{hr} \right) \sqrt{\frac{1 \text{ bar} \cdot 0.2068 \text{ bar}}{1.5189 \frac{m^3}{m^3} \cdot 293.15 \text{ K}}}$$

$$Q_n = 45.84 \frac{m^3}{hr}$$

New Solenoid

$$Q_n = 514 \left(1.552 \frac{m^3}{hr} \right) \sqrt{\frac{1 \text{ bar} \cdot 0.2068 \text{ bar}}{1.5189 \frac{m^3}{m^3} \cdot 293.15 \text{ K}}}$$

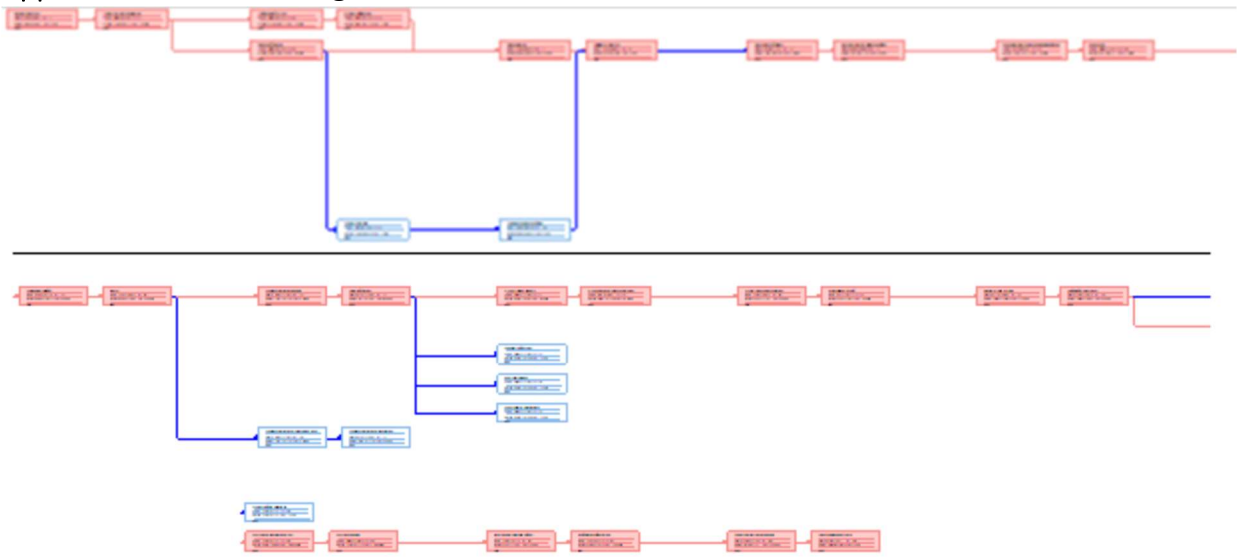
$$Q_n = 17.19 \frac{m^3}{hr} \quad 62.5 \% \text{ decrease in Flow Rate}$$

New Regulator

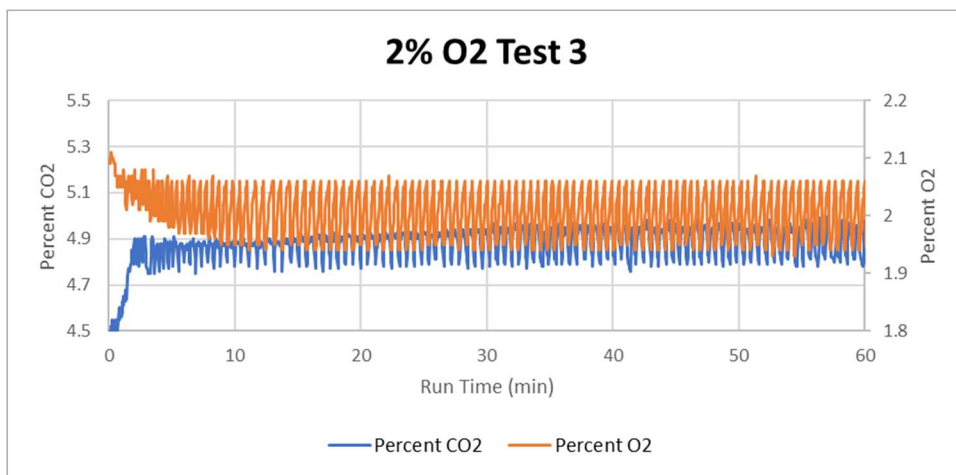
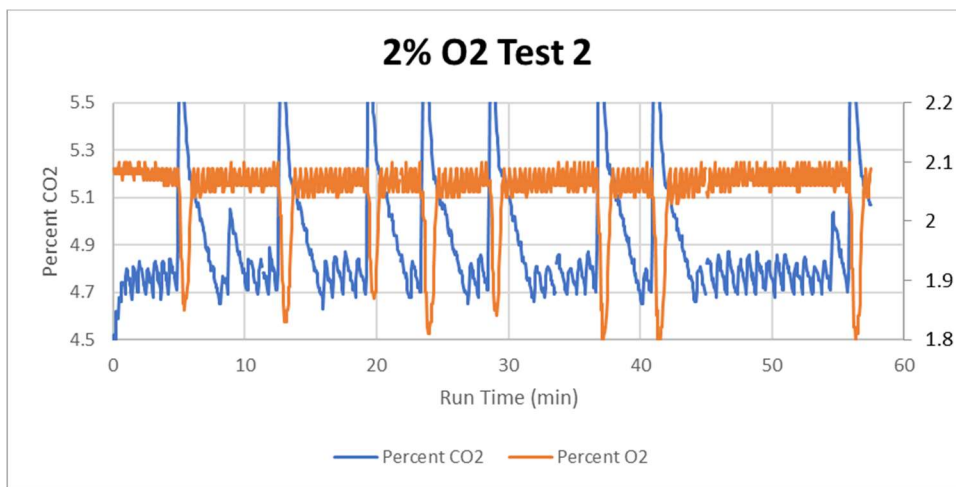
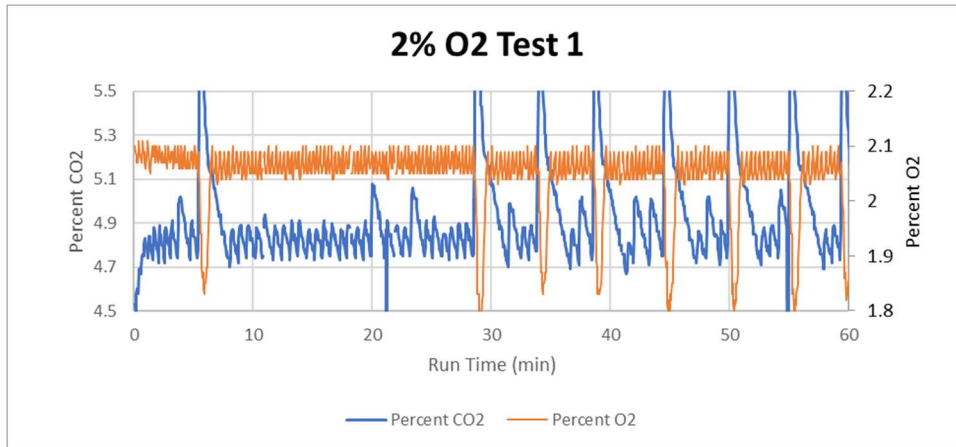
$$Q_n = 514 \left(4.138 \frac{m^3}{hr} \right) \sqrt{\frac{1 \text{ bar} \cdot 0.0689 \text{ bar}}{1.5189 \frac{m^3}{m^3} \cdot 293.15 \text{ K}}}$$

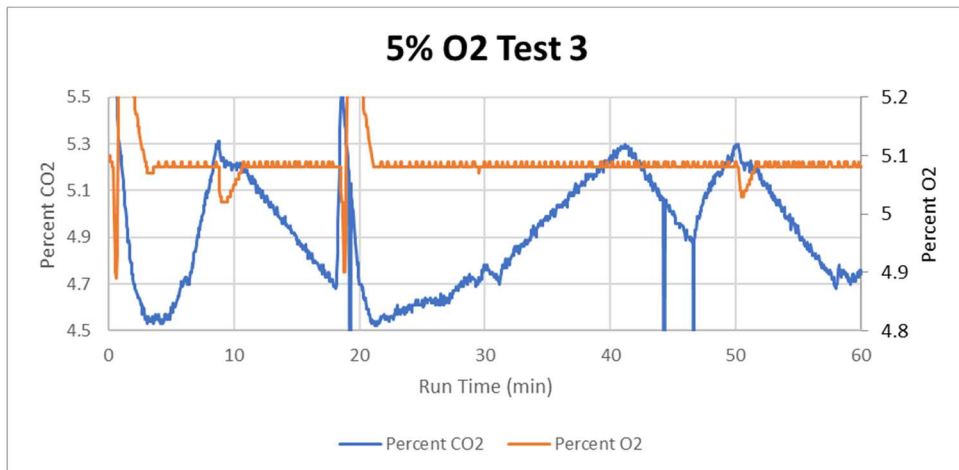
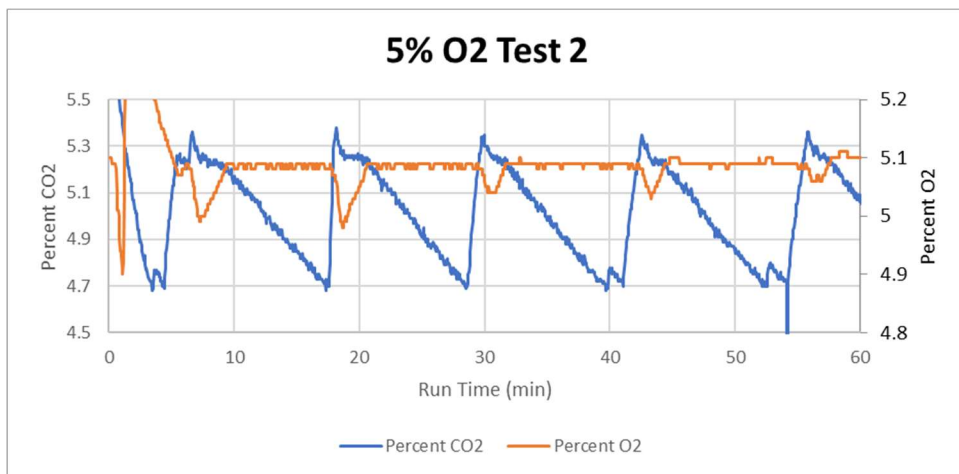
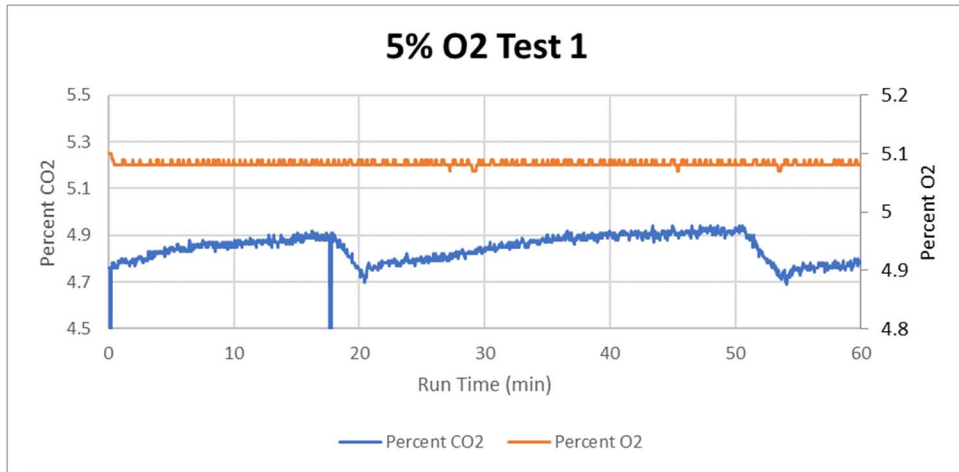
$$Q_n = 26.46 \frac{m^3}{hr} \quad 42.3 \% \text{ decrease in Flow Rate}$$

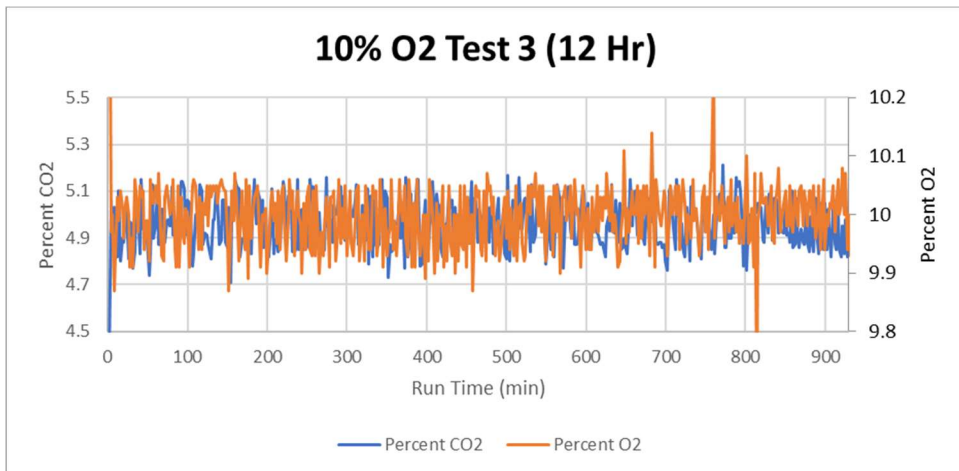
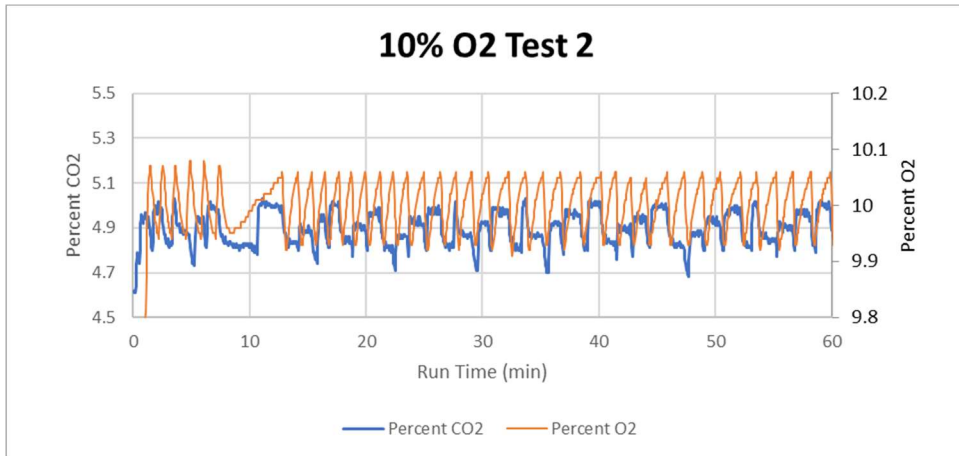
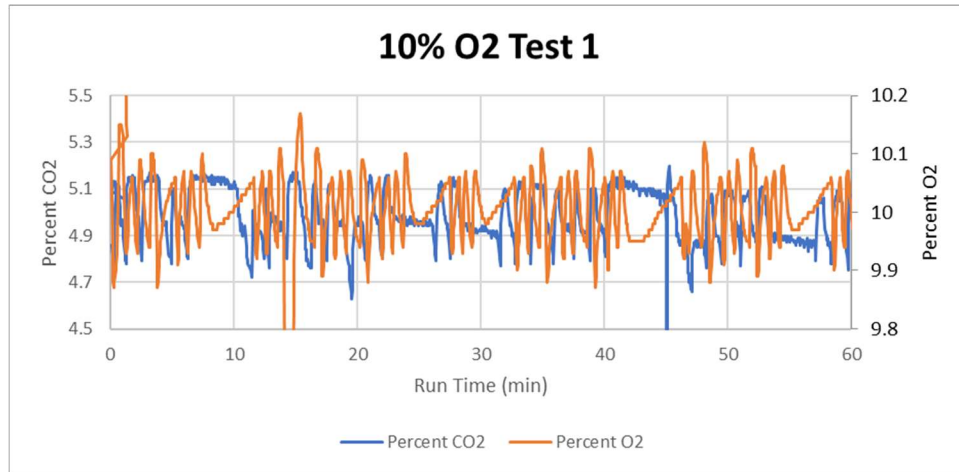
Appendix C. Network Diagram



Appendix D. Test Data







Appendix E. Code for Arduino MEGA

```

/*****
Code for the Arduino MEGA
*****/

// Arduino Libraries
#include "SPI.h"
#include "Adafruit_GFX.h"
#include "Adafruit_ILI9341.h"
#include "SoftwareSerial.h"
#include "OneWire.h"
#include "DallasTemperature.h"
#include "cozir.h"

/* Pin Connections */
// Serial connections
// O2 Sensor: 18/19
// CO2 Sensor: 11/12
#define HeatPad 6
#define ONE_WIRE_BUS 7 // Temperature sensor
#define O2Sole 8
#define N2Sole 9
#define CO2Sole 10
#define TFT_CS 53
#define TFT_DC 48
#define TFT_MOSI 51
#define TFT_CLK 52
#define TFT_RST 47
#define TFT_MISO 50
#define TS_CS 45
#define ROTATION 1
#define O2Sole 8
#define N2Sole 9

OneWire oneWire(ONE_WIRE_BUS); // Setup a oneWire instance to communicate with ANY OneWire devices
DallasTemperature sensors(&oneWire); // Pass our oneWire reference to Dallas Temperature.
SoftwareSerial nss(11,12); // Rx, Tx - Pin 5,7 on sensor
COZIR czr(nss);
Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC, TFT_RST);
XPT2046_Touchscreen ts(TS_CS);

/* Variables for O2 sensor */
String inString=""; // hold the string
String subString=""; // O2 substring
float O2Data = 1; // variable to store string -> float (Needs value of 1 for prevO2Data first calculation)
bool dataReceived = false;

/* Variables for CO2 sensor */
float SingleCO2 = 0;
float holdCO2 = 0.00;
float multiplier = 0.001; // 10/10000 (Hardware multiplier/ppm conversion).
float CO2Data;

/* Variables for Temp sensor */
float T1, T2, Temp;

```

```

unsigned long dataGrabtimestamp = 0UL;

/* Gas Controller Variables */
float O2Setpoint = 10;
float O2tolerance = 0.05;
float CO2Setpoint = 5;
float CO2tolerance = 0.2;
float TSetpoint = 37.5;
bool CO2state = false;
bool O2state = false;
bool N2state = false;
long O2Kp = 2000;
long CO2Kp = 2000;
long N2KpO2 = -2671 * log(O2Setpoint) + 7750;
long N2KpCO2 = 1500;
int HeatOnTime = 1000;
unsigned long closeTime = 10000UL;
unsigned long O2openTime = 500UL;
unsigned long N2openTime = 2500UL;
unsigned long N2openTimeO2 = 2500UL;
unsigned long CO2openTime = 3000UL;
unsigned long O2opentimestamp = 0UL;
unsigned long N2opentimestampO2 = 0UL;
unsigned long N2opentimestampCO2 = 0UL;
unsigned long CO2opentimestamp = 0UL;
unsigned long O2closetimestamp = 0UL;
unsigned long N2closetimestampO2 = 0UL;
unsigned long N2closetimestampCO2 = 0UL;
unsigned long CO2closetimestamp = 0UL;
float printTime = 3001;
long startUpTime = 180000;
float prevO2Data = 0.0;
unsigned long O2Datatimestamp = 0;

/* Touchscreen Calibration variables */
float xCalM, yCalM;
float xCalC, yCalC;

class ScreenPoint {
public:
    int16_t x;
    int16_t y;
    ScreenPoint(int16_t xIn, int16_t yIn) {
        x = xIn;
        y = yIn;
    }
};

ScreenPoint getScreenCoords(int16_t x, int16_t y) {
    int16_t xCoord = round((x * xCalM) + xCalC);
    int16_t yCoord = round((y * yCalM) + yCalC);
    if (xCoord < 0) xCoord = 0;
    if (xCoord >= tft.width()) xCoord = tft.width() - 1;
    if (yCoord < 0) yCoord = 0;
    if (yCoord >= tft.height()) yCoord = tft.height() - 1;
    return (ScreenPoint(xCoord, yCoord));
}

```

```
void calibrateTouchScreen() {
  TS_Point p;
  int16_t x1, y1, x2, y2;

  tft.fillScreen(ILI9341_BLACK);
  tft.setCursor(45, 90);
  tft.setTextSize(2);
  tft.setTextColor(ILI9341_CYAN);
  tft.print("Calibrate screen by");
  tft.setCursor(75, 130);
  tft.print("tapping crosses");
  while (ts.touched()); // Waits if the user is already touching
  tft.drawFastHLine(10, 20, 20, ILI9341_RED);
  tft.drawFastVLine(20, 10, 20, ILI9341_RED);
  while (!ts.touched()); // Waits until the user touches
  delay(50);
  p = ts.getPoint();
  x1 = p.x;
  y1 = p.y;
  tft.drawFastHLine(10, 20, 20, ILI9341_BLACK);
  tft.drawFastVLine(20, 10, 20, ILI9341_BLACK);
  delay(500);
  while (ts.touched());
  tft.drawFastHLine(tft.width() - 30, tft.height() - 20, 20, ILI9341_RED);
  tft.drawFastVLine(tft.width() - 20, tft.height() - 30, 20, ILI9341_RED);
  while (!ts.touched());
  delay(50);
  p = ts.getPoint();
  x2 = p.x;
  y2 = p.y;
  tft.drawFastHLine(tft.width() - 30, tft.height() - 20, 20, ILI9341_BLACK);
  tft.drawFastVLine(tft.width() - 20, tft.height() - 30, 20, ILI9341_BLACK);

  int16_t xDist = tft.width() - 40;
  int16_t yDist = tft.height() - 40;

  // translate in form pos = m x val + c
  // x
  xCalM = (float)xDist / (float)(x2 - x1);
  xCalC = 20.0 - ((float)x1 * xCalM);
  // y
  yCalM = (float)yDist / (float)(y2 - y1);
  yCalC = 20.0 - ((float)y1 * yCalM);
}

void setup() {
  // Sets baudrate for serial (and digital serial) communication
  Serial.begin(9600);
  Serial1.begin(9600);

  // Initializes solenoid and sets to closed (HIGH)
  pinMode(CO2Sole, OUTPUT);
  digitalWrite(CO2Sole, HIGH);
  pinMode(O2Sole, OUTPUT);
  digitalWrite(O2Sole, HIGH);
  pinMode(N2Sole, OUTPUT);
  digitalWrite(N2Sole, HIGH);
  pinMode(HeatPad, OUTPUT);
```

```
digitalWrite(HeatPad, HIGH);

// Starts sensors
czr.SetOperatingMode(CZR_POLLING);
sensors.begin();

// Initializes the touchscreen graphics
pinMode(TS_CS, OUTPUT);
pinMode(TFT_CS, OUTPUT);
digitalWrite(TS_CS, HIGH);
digitalWrite(TFT_CS, HIGH);

// Starts the touchscreen graphics
tft.begin();
tft.setRotation(ROTATION);

// Starts the touch-sensing ability
ts.begin();
ts.setRotation(ROTATION);
calibrateTouchScreen();
}

/* Touchscreen function variables */
int pageID = 11;
int page1Ver;
TS_Point p;
ScreenPoint sp = getScreenCoords(0, 0);

/* Runtime and error variables */
unsigned long startTime;
unsigned long reachTime;
unsigned long reachTimeInSeconds;
unsigned long errorStartTime;
unsigned long errorEndTime;
unsigned long errorCycleTime;
unsigned long errorTotalTime;
unsigned long totalTimeInSeconds;
int t_h, t_m, t_s, e_h, e_m, e_s, r_h, r_m, r_s;
int temp_s = 0;
int temp_es = 0;
int temp_rs = 0;
bool started = false;
bool prevstarted = started;
bool reached = false;
bool oldreached = false;
bool inRange = false;
bool oldinRange;

unsigned long computeOpenTime(float gasData, float gasSetpoint, long Kp) {
    double error = abs(gasData - gasSetpoint);
    long opentime = Kp * error;
    return opentime;
}

void controlO2() {
    if (O2Data < O2Setpoint - O2tolerance + .02) {
        if (O2state == false && millis() - O2closetimestamp > closeTime) { // opens O2 solenoid if it has been closed for
            "closetime"
```

```

    if (O2Data < prevO2Data * 1.02 && O2Data > prevO2Data * 0.98) { // skips O2 solenoid open if O2 rises more
than 3%
        digitalWrite(O2Sole, LOW);
        O2opentimestamp = millis();
        O2openTime = computeOpenTime(O2Data, O2Setpoint, O2Kp);
        O2state = true;
    }
    } else if (O2state == true && millis() - O2opentimestamp > O2openTime) { // closes O2 solenoid if has been open
for "opentime"
        digitalWrite(O2Sole, HIGH);
        O2closetimestamp = millis();
        O2state = false;
    }
    } else if (O2Data > O2Setpoint + O2tolerance) {
        digitalWrite(O2Sole, HIGH); // If O2 is greater than setpoint, O2 should close immediately
        if (N2state == false && millis() - N2closetimestampO2 > closeTime) { // opens N2 solenoid if it has been closed
for "closetime"
            if (O2Data > prevO2Data * 0.98 && O2Data < prevO2Data * 1.02) {
                digitalWrite(N2Sole, LOW);
                N2opentimestampO2 = millis();
                N2openTimeO2 = computeOpenTime(O2Data, O2Setpoint, N2KpO2);
                N2state = true;
            }
            } else if (N2state == true && millis() - N2opentimestampO2 > N2openTimeO2) { // closes N2 solenoid if has been
open for "opentime"
                digitalWrite(N2Sole, HIGH);
                N2closetimestampO2 = millis();
                N2state = false;
            }
        } else {
            digitalWrite(O2Sole, HIGH);
        }
    }
}

void controlCO2() {
    if (CO2Data > CO2Setpoint + CO2tolerance) {
        digitalWrite(CO2Sole, HIGH);
        if (N2state == false && millis() - N2closetimestampCO2 > closeTime) { // opens N2 solenoid if it has been closed
for "closetime"
            digitalWrite(N2Sole, LOW);
            N2opentimestampCO2 = millis();
            N2openTime = computeOpenTime(CO2Data, CO2Setpoint, N2KpCO2);
            N2state = true;
        } else if (N2state == true && millis() - N2opentimestampCO2 > N2openTime) { // closes N2 solenoid if has been
open for "opentime"
            digitalWrite(N2Sole, HIGH);
            N2closetimestampCO2 = millis();
            N2state = false;
        }
    } else if (CO2Data < CO2Setpoint - CO2tolerance) {
        if (CO2state == false && millis() - CO2closetimestamp > closeTime) { // opens CO2 solenoid if it has been closed
for "closetime"
            digitalWrite(CO2Sole, LOW);
            CO2opentimestamp = millis();
            CO2openTime = computeOpenTime(CO2Data, CO2Setpoint, CO2Kp);
            CO2state = true;
        } else if (CO2state == true && millis() - CO2opentimestamp > CO2openTime) { // closes CO2 solenoid if has been
open for "opentime"
    }
}

```



```
    digitalWrite(CO2Sole, HIGH);
    CO2closetimestamp = millis();
    CO2state = false;
  }
} else {
  digitalWrite(CO2Sole, HIGH);
}
}

void controlTemp() {
  if (Temp < TSetpoint) {
    digitalWrite(HeatPad, LOW);
  } else if (Temp > TSetpoint) {
    digitalWrite(HeatPad, HIGH);
  }
}

void grabO2() {
  if (millis() - O2Datatimestamp > 1000){
    prevO2Data = O2Data;
    O2Datatimestamp = millis();
  }

  int inChar = Serial1.read();
  if (isDigit(inChar))inString += (char)inChar;
  subString = inString.substring(13,17);
  if (inChar == '\n'){
    O2Data = subString.toFloat();
    O2Data = O2Data / 100;
    dataReceived = true;
    inString = "";
    subString = "";
  }
}

void grabCO2() {
  CO2Data = czr.CO2() * multiplier;
  if (CO2Data <= 0.6) {
    grabCO2();
  }
}

void pageZero() {
  tft.fillScreen(ILI9341_WHITE);
  tft.fillRect(0, 40, tft.width(), 2, ILI9341_BLACK);
  tft.setTextSize(2);
  tft.setTextColor(ILI9341_BLACK);
  tft.setCursor(30, 14);
  tft.print("Solenoid Calibration");
  tft.fillRect(tft.width() - 36, 3, 34, 34, ILI9341_BLACK);
  tft.fillRect(tft.width() - 34, 5, 30, 30, ILI9341_WHITE);
  tft.fillTriangle(tft.width() - 27, 10, tft.width() - 27, 28, tft.width() - 10, 20, ILI9341_BLACK);

  if (started == false) {
    tft.setTextSize(3);
    tft.setCursor(80, 70);
    tft.print("O2:");
    tft.fillRect(131, 54, 79, 54, ILI9341_BLACK);
  }
}
```

```
if (O2state == false) {
  tft.setTextSize(2);
  tft.setCursor(145,72);
  tft.fillRect(133, 56, 75, 50, 0x03A0);
  tft.print("Open");
} else if (O2state == true) {
  tft.setTextSize(2);
  tft.setCursor(140, 72);
  tft.fillRect(133, 56, 75, 50, 0xF800);
  tft.print("Close");
}

tft.setTextSize(3);
tft.setCursor(62, 129);
tft.print("CO2:");
tft.fillRect(131, 113, 79, 54, ILI9341_BLACK);
if (CO2state == false) {
  tft.setTextSize(2);
  tft.setCursor(145,131);
  tft.fillRect(133, 115, 75, 50, 0x03A0);
  tft.print("Open");
} else if (CO2state == true) {
  tft.setTextSize(2);
  tft.setCursor(140, 131);
  tft.fillRect(133, 115, 75, 50, 0xF800);
  tft.print("Close");
}

tft.setTextSize(3);
tft.setCursor(80, 188);
tft.print("N2:");
tft.fillRect(131, 172, 79, 54, ILI9341_BLACK);
if (N2state == false) {
  tft.setTextSize(2);
  tft.setCursor(145,190);
  tft.fillRect(133, 174, 75, 50, 0x03A0);
  tft.print("Open");
} else if (N2state == true) {
  tft.setTextSize(2);
  tft.setCursor(140, 190);
  tft.fillRect(133, 174, 75, 50, 0xF800);
  tft.print("Close");
}
} else if (started == true) {
  tft.setCursor(15, 90);
  tft.setTextSize(2);
  tft.print("Incubator must be paused");
  tft.setCursor(70, 130);
  tft.print("for calibration.");
}
}

void pageOne() {
  // Determine background color
  if (started == false) {
    tft.fillScreen(0xFDB8);
  } else {
    tft.fillScreen(0x9772);
  }
}
```

```
}

// Draw the header
tft.fillRect(0, 40, tft.width(), 2, ILI9341_BLACK);
tft.setTextSize(2);
tft.setTextColor(ILI9341_BLACK);
tft.setCursor(64, 14);
tft.print("Control Setpoint");
tft.fillRect(4, 3, 34, 34, ILI9341_BLACK);
tft.fillRect(6, 5, 30, 30, ILI9341_WHITE);
tft.fillTriangle(27, 10, 27, 28, 12, 20, ILI9341_BLACK);
tft.fillRect(tft.width() - 36, 3, 34, 34, ILI9341_BLACK);
tft.fillRect(tft.width() - 34, 5, 30, 30, ILI9341_WHITE);
tft.fillTriangle(tft.width() - 27, 10, tft.width() - 27, 28, tft.width() - 10, 20, ILI9341_BLACK);

// Draw the control section:
tft.setCursor(7,70);
tft.setTextSize(3);
tft.print("O2:");
tft.fillRect(61, 54, 79, 54, ILI9341_BLACK);
tft.fillRect(63, 56, 75, 50, ILI9341_WHITE);
tft.setCursor(66, 69);
tft.print(O2Setpoint);
if (pageID == 12) {
  tft.fillRect(198, 54, 54, 54, ILI9341_BLACK);
  tft.fillRect(200, 56, 50, 50, 0x03A0);
  tft.fillTriangle(208, 63, 208, 99, 244, 81, ILI9341_BLACK);

  tft.fillRect(147, 54, 44, 26, ILI9341_BLACK); //top rec
  tft.fillRect(149, 56, 40, 22, ILI9341_WHITE);
  tft.fillTriangle(159, 75, 169, 57, 179, 75, ILI9341_BLACK);
  tft.fillRect(147, 82, 44, 26, ILI9341_BLACK); //bottom rec
  tft.fillRect(149, 84, 40, 22, ILI9341_WHITE);
  tft.fillTriangle(159, 86, 169, 104, 179, 86, ILI9341_BLACK);
} else {
  tft.fillRoundRect(258, 54, 54, 54, 4, ILI9341_BLACK);
  tft.fillRoundRect(260, 56, 50, 50, 4, 0xF800); //PALE VIOLET RED IS 0xDB92, PAPAYAWHIP 0xFF7A, PERU 0xCC27;
  tft.fillRoundRect(268, 64, 34, 34, 8, ILI9341_BLACK);
}

// Draw the "Current Value" Section
tft.setTextSize(2);
tft.setCursor(10, 130);
tft.print("Current Values");
tft.fillRect(0, 120, tft.width(), 2, ILI9341_BLACK);

tft.setCursor(22, 170);
tft.print("O2:");
tft.fillRect(56, 160, 69, 34, ILI9341_BLACK);
tft.fillRect(58, 162, 65, 30, ILI9341_WHITE);
tft.setCursor(132, 170);
tft.print("%");

tft.setCursor(10, 210);
tft.print("CO2:");
tft.fillRect(56, 200, 69, 34, ILI9341_BLACK);
tft.fillRect(58, 202, 65, 30, ILI9341_WHITE);
```

```
tft.setCursor(132, 210);
tft.print("%");

tft.setCursor(160, 170);
tft.print("TEMP:");
tft.fillRect(219, 160, 69, 34, ILI9341_BLACK);
tft.fillRect(221, 162, 65, 30, ILI9341_WHITE);
tft.setCursor(295, 170);
tft.print("C");
}

void printO2Setpoint() {
tft.setTextSize(3);
tft.fillRect(63, 56, 75, 50, ILI9341_WHITE);
tft.setCursor(66, 69);
tft.print(O2Setpoint);
}

void printO2() {
tft.fillRect(58, 162, 65, 30, ILI9341_WHITE);
tft.setTextSize(2);
tft.setCursor(62, 170);
tft.print(O2Data);
dataReceived = false;
}

void printCO2() {
tft.fillRect(58, 202, 65, 30, ILI9341_WHITE);
tft.setTextSize(2);
tft.setCursor(66, 210);
tft.print(CO2Data);
}

void printTemp() {
tft.fillRect(221, 162, 65, 30, ILI9341_WHITE);
tft.setTextSize(2);
tft.setCursor(224, 170);
tft.print(Temp);
}

void pageTwo() {
// Draw the header
tft.fillRect(0, 40, tft.width(), 2, ILI9341_BLACK);
tft.setTextSize(2);
tft.setTextColor(ILI9341_BLACK);
tft.setCursor(80, 14);
tft.print("Error Tracker");
tft.fillRect(4, 3, 34, 34, ILI9341_BLACK);
tft.fillRect(6, 5, 30, 30, ILI9341_WHITE);
tft.fillTriangle(27, 10, 27, 28, 12, 20, ILI9341_BLACK);

// Draw error range section
tft.setCursor(10, 70);
tft.print("Run Time:");
tft.fillRect(120, 60, 44, 34, ILI9341_BLACK);
tft.fillRect(122, 62, 40, 30, ILI9341_WHITE);
tft.setCursor(167, 70);
```

```

tft.print("h");
tft.fillRect(182, 60, 32, 34, ILI9341_BLACK);
tft.fillRect(184, 62, 28, 30, ILI9341_WHITE);
tft.setCursor(217,70);
tft.print("m");
tft.fillRect(231, 60, 34, 34, ILI9341_BLACK);
tft.fillRect(233, 62, 30, 30, ILI9341_WHITE);
tft.setCursor(269,70);
tft.print("s");

tft.setCursor(10,110);
tft.print("Time to Setpoint:");
tft.fillRect(182, 130, 32, 34, ILI9341_BLACK);
tft.fillRect(184, 132, 28, 30, ILI9341_WHITE);
tft.setCursor(217,140);
tft.print("m");
tft.fillRect(231, 130, 34, 34, ILI9341_BLACK);
tft.fillRect(233, 132, 30, 30, ILI9341_WHITE);
tft.setCursor(269,140);
tft.print("s");

tft.setCursor(10, 170);
tft.print("Time Outside Range:");
tft.fillRect(120, 192, 44, 34, ILI9341_BLACK);
tft.fillRect(122, 194, 40, 30, ILI9341_WHITE);
tft.setCursor(167, 202);
tft.print("h");
tft.fillRect(182, 192, 32, 34, ILI9341_BLACK);
tft.fillRect(184, 194, 28, 30, ILI9341_WHITE);
tft.setCursor(217, 202);
tft.print("m");
tft.fillRect(231, 192, 34, 34, ILI9341_BLACK);
tft.fillRect(233, 194, 30, 30, ILI9341_WHITE);
tft.setCursor(269, 202);
tft.print("s");
}

void grabTemp() {
  sensors.requestTemperatures();
  T1 = sensors.getTempCByIndex(0);
  T2 = sensors.getTempCByIndex(1);
  Temp = (T1 + T2)/2;
}

bool checkInRange() {
  if ((O2Data > O2Setpoint - 0.1 && O2Data < O2Setpoint + 0.1) && (CO2Data > CO2Setpoint - 0.5 && CO2Data <
CO2Setpoint + 0.5)) {
    return true;
  } else {
    return false;
  }
}

void calculateReachTime() {
  reachTimeinSeconds = (reachTime - startTime) / 1000;
  r_h = reachTimeinSeconds / 3600;
  r_m = (reachTimeinSeconds - (3600 * r_h)) / 60;
  r_s = (reachTimeinSeconds - (3600 * r_h) - (r_m * 60));
}

```

```
}

void calculateRunTime() {
    totalTimeInSeconds = ((millis() - reachTime) / 1000);
    t_h = totalTimeInSeconds / 3600;
    t_m = (totalTimeInSeconds - (3600 * t_h)) / 60;
    t_s = (totalTimeInSeconds - (3600 * t_h) - (t_m * 60));
}

void calculateErrorTime() {
    totalTimeInSeconds = (errorTotalTime + errorCycleTime) / 1000;
    e_h = totalTimeInSeconds / 3600;
    e_m = (totalTimeInSeconds - (3600 * e_h)) / 60;
    e_s = (totalTimeInSeconds - (3600 * e_h) - (e_m * 60));
}

void printReachTime() {
    if (r_s != temp_rs && reached == false) {
        tft.fillRect(184, 132, 28, 30, ILI9341_WHITE);
        tft.setCursor(187, 139);
        tft.print(r_m);
        tft.fillRect(233, 132, 30, 30, ILI9341_WHITE);
        tft.setCursor(236, 139);
        tft.print(r_s);
        temp_rs = r_s;
    } else {
        tft.fillRect(184, 132, 28, 30, ILI9341_WHITE);
        tft.setCursor(187, 139);
        tft.print(r_m);
        tft.fillRect(233, 132, 30, 30, ILI9341_WHITE);
        tft.setCursor(236, 139);
        tft.print(r_s);
    }
}

void printRunTime() {
    if (t_s != temp_s) {
        tft.fillRect(122, 62, 40, 30, ILI9341_WHITE);
        tft.setCursor(125, 69);
        tft.print(t_h);
        tft.fillRect(184, 62, 28, 30, ILI9341_WHITE);
        tft.setCursor(187, 69);
        tft.print(t_m);
        tft.fillRect(233, 62, 30, 30, ILI9341_WHITE);
        tft.setCursor(236, 69);
        tft.print(t_s);
        temp_s = t_s;
    }
}

void printErrorTime() {
    if (e_s != temp_es) {
        tft.fillRect(122, 194, 40, 30, ILI9341_WHITE);
        tft.setCursor(125, 201);
        tft.print(e_h);
        tft.fillRect(184, 194, 28, 30, ILI9341_WHITE);
        tft.setCursor(187, 201);
        tft.print(e_m);
    }
}
```

```

tft.fillRect(233, 194, 30, 30, ILI9341_WHITE);
tft.setCursor(236, 201);
tft.print(e_s);
temp_es = e_s;
}
}

void loop() {

if (O2Setpoint >=5) {
closeTime = 20000UL;
}

/*-----RECEIVE/SEND DATA-----*/
// Grabs data from sensors
if (Serial1.available()) {
grabO2();
}

if (millis() - dataGrabtimestamp > 50) {
grabCO2();
grabTemp();
dataGrabtimestamp = millis();
}

/*-----*/
if (started == false && millis() >= startUpTime) {
started = true;
} else if (started == false) {
digitalWrite(O2Sole, HIGH);
digitalWrite(N2Sole, HIGH);
digitalWrite(CO2Sole, HIGH);
}
/*-----CONTROL SYS / ERROR DETECTOR-----*/
controlTemp();
if (started == true) {
controlO2();
controlCO2();
if (O2Data < O2Setpoint + O2tolerance && CO2Data < CO2Setpoint + CO2tolerance){
digitalWrite(N2Sole, HIGH);
}

if (reached == false) {
reached = checkInRange();
reachTime = millis();
calculateReachTime();
} else if (reached == true) { // Assigning time from start to in range value
if (oldreached != reached) {
reachTime = millis();
calculateReachTime();
oldreached = reached;
oldinRange = true;
}
}

inRange = checkInRange();
if (oldinRange != inRange && inRange == false) { // This if-statement checks if it switched out range
errorStartTime = millis();
oldinRange = inRange;
}
}

```

```

} else if (oldinRange != inRange && inRange == true) { // This if-statement checks if it switched in range
    errorEndTime = millis();
    errorCycleTime = errorEndTime - errorStartTime;
    errorTotalTime += errorCycleTime;
    errorCycleTime = errorEndTime = errorStartTime = 0;
    oldinRange = inRange;
}

if (inRange == false) {
    errorEndTime = millis();
    errorCycleTime = errorEndTime - errorStartTime;
}
}
}
/*-----*/
/*-----GENERATE PAGE GRAPHICS/CONTROLS-----*/
switch (pageID) {
case 31: // Generate Page 0
    pageID = 32;
    pageZero();
    break;

case 32:
    if (ts.touched()) {
        delay(100);
        p = ts.getPoint();
        sp = getScreenCoords(p.x, p.y);

        if (started == false) {
            if (sp.x >= tft.width() - 30 && sp.y <= 30 && N2state == false && O2state == false && CO2state == false) { //
Page 2 Select Button
                pageID = page1Ver;
                pageOne();
                break;
            } else if ((sp.x >= 133 && sp.x <= 208) && (sp.y >= 56 && sp.y <= 106)) {
                if (O2state == false) {
                    O2state = true;
                    digitalWrite(O2Sole, LOW);
                    pageZero();
                    break;
                } else if (O2state = true) {
                    O2state = false;
                    digitalWrite(O2Sole, HIGH);
                    pageZero();
                    break;
                }
            }
            } else if (((sp.x >= 133 && sp.x <= 208) && (sp.y >= 115 && sp.y <= 165))) {
                if (CO2state == false) {
                    CO2state = true;
                    sendData.CO2state = CO2state;
                    myTransfer.sendDatum(sendData);
                    pageZero();
                    break;
                } else if (CO2state = true) {
                    CO2state = false;
                    sendData.CO2state = CO2state;
                    myTransfer.sendDatum(sendData);
                    pageZero();
                }
            }
        }
    }
}

```



```

        break;
    }
} else if (((sp.x >= 133 && sp.x <= 208) && (sp.y >= 174 && sp.y <= 224))) {
    if (N2state == false) {
        N2state = true;
        digitalWrite(N2Sole, LOW);
        pageZero();
        break;
    } else if (N2state = true) {
        N2state = false;
        digitalWrite(N2Sole, HIGH);
        pageZero();
        break;
    }
}
} else if (started == true) {
    if (sp.x >= tft.width() - 30 && sp.y <= 30) {
        pageID = page1Ver;
        pageOne();
        break;
    }
}
}
break;

case 11: // Generate Page 1
    pageID = 12;
    pageOne();
    break;

case 12: // Page 1 - (Start Button) - Wait for input
    page1Ver = 12;
    if (dataReceived == true) {
        printO2();
        printCO2();
        printTemp();
    }
    if (ts.touched()) {
        delay(100);
        p = ts.getPoint();
        sp = getScreenCoords(p.x, p.y);

        if (sp.x >= tft.width() - 30 && sp.y <= 30) { // Page 2 Select Button
            pageID = 21;
            break;
        } else if (sp.x <= 30 && sp.y <= 30) { // Page 0 Select Button
            pageID = 31;
            break;
        } else if ((sp.x >= 149 && sp.x <= 189) && (sp.y >= 56 && sp.y <= 78)) { // O2 Adjust Up Button
            if (O2Setpoint < 9.9) {
                O2Setpoint += .1;
                printO2Setpoint();
                break;
            }
        } else if ((sp.x >= 149 && sp.x <= 189) && (sp.y >= 84 && sp.y <= 106)) { // O2 Adjust Down Button
            if (O2Setpoint > 0.11) {
                O2Setpoint -= .1;
                printO2Setpoint();
            }
        }
    }
}

```

```

        break;
    }
} else if ((sp.x >= 200 && sp.x <= 250) && (sp.y >= 56 && sp.y <= 106)) { // Start button
    startTime = millis();
    started = true;
    reached = false;
    errorTotalTime = 0;
    pageID = 13;
    pageOne();
}
}
break;

case 13: // Page 1 - (Stop Button) - Wait for Input
    page1Ver = 13;
    if (dataReceived == true) {
        printO2();
        printCO2();
        printTemp();
    }

    if (ts.touched()) {
        delay(100);
        p = ts.getPoint();
        sp = getScreenCoords(p.x, p.y);

        if (sp.x >= tft.width() - 30 && sp.y <= 30) { // Page 2 Select Button; if pressed then moves on to page two
            pageID = 21;
            break;
        } else if (sp.x <= 30 && sp.y <= 30) { // Page 0 Select Button
            pageID = 31;
            break;
        } else if ((sp.x >= 260 && sp.x <= 310) && (sp.y >= 56 && sp.y <= 106)) { // Stop Button
            // STOP timer
            started = false;
            reached = false;
            oldreached = false;
            inRange = false;
            errorCycleTime = 0;
            errorTotalTime = 0;
            pageID = 12;
            pageOne();
        }
    }
}
break;

case 21: // Generate Page 2
    pageTwo();
    pageID = 22;
    break;

case 22:
    if (ts.touched()) {
        p = ts.getPoint();
        sp = getScreenCoords(p.x, p.y);
        if (sp.x <= 30 && sp.y <= 30) {
            pageID = page1Ver;
            pageOne();
        }
    }
}

```

```
        break;
    }
}
if (started == true && reached == true) {
    calculateRunTime();
    printRunTime();
    calculateErrorTime();
    printErrorTime();
    printReachTime();
} else if (started == true && reached == false) {
    printReachTime();
}
default:
    Serial.print("Default");
    break;
}
/*-----*/
if (millis() - printTime >= 3000) {
    Serial.print("CO2,");
    Serial.print(CO2Data);
    Serial.print(",");
    Serial.print("Temp,");
    Serial.print(Temp);
    Serial.print(",");
    Serial.print("O2,");
    Serial.print(O2Data);
    Serial.print(",");
    Serial.print(N2KpO2);
    Serial.print(",");
    Serial.println(N2openTimeO2);
    printTime = millis();
}
}
```