Automatic HIFLOW Oxygen Weaning Device

Spring 2017

Final Report 5/3/2017

CSEE 4911, Spring 2017

Computer Systems Engineering

College of Engineering

The University of Georgia

Automatic HiFlow Oxygen Weaning Device

Team Members:
Sarah Dowling, Ryan Foster, Kate Garrett,
Justin Joseph, Melanie Kemp, Luis Perea

Faculty Advisor:
Dr. Ramana Pidaparti

Sponsors:
Children's Healthcare of Atlanta
Dr. Pradip Kamat
Dr. Jocelyn Grunwell

University of Georgia
- CSEE 4911 -
May 3rd, 2017

Table of Contents

Semester 1

Abstract

The purpose of this project is to design an interface for the HIFLOW nasal cannula system. This interface should use negative feedback to slowly decrease or increase the amount of oxygen delivered to the patient based on the patient's current and previous FiO2 and SpO2. An alarm should be used when the patient's levels of FiO2 are low. The purpose of this project is to relieve bedside staff from having to interpret the oxygen level data and manually having to adjust the oxygen levels, therefore allowing for staff to resume other duties.

Project Title: Automatic Control of Inspired Oxygen for Non-Invasive Ventilation in the Pediatric Intensive Care Unit

Project Background: Oxygen therapy is frequently required for critically ill children in the pediatric intensive care unit (PICU) via endotracheal intubation and mechanical ventilation (invasive), non-invasive positive pressure ventilation (NIPPV) and, more commonly, through HIFLO nasal cannula systems. Exposure to high oxygen concentrations (hyperoxia) can result in oxidative damage to already damaged lungs. Infants born prematurely are especially sensitive to oxidative damage to their developing eyes, brain and lungs upon prolonged exposure to hyperoxia during necessary respiratory support. Respiratory therapists and nurses have many competing responsibilities and at times care for multiple patients in the PICU limiting their ability to quickly wean oxygen. The patient's oxygenation status is continuously monitored using pulse oximetry, arterial blood gas, or both. Pulse oximetry is non-invasive, but fluctuates depending on the patient's respiratory status and perfusion. Oxygen flow meters/blenders require bedside staff to manually adjust oxygen delivered.

Project Objective: The goal of this project is to develop an auto-weaning device using a patient feedback loop that interfaces with and adjusts the oxygen concentration according to the child's oxygen needs to minimize both hypoxia and hyperoxia for use with HIFLOW systems.

Design Requirements:
The device will:

1. Use data from the patient pulse oximeter to determine the FiO2 required to maintain the target SpO2 in a continuous feedback loop mechanism. A lower limit SpO2 of 93% may be initially set, but the target may be lowered to 88-90% to prevent lung injury from hyperoxia in a sicker child. The difference between the target SpO2 and the measured SpO2 will be used to adjust the FiO2. Trended data (captured on the patient's bedside monitor) or duration outside the SpO2 parameters can help guide the algorithm for FiO2 adjustment.

2. If the patient's SpO2 is above the target for > 30 min, the FiO2 will automatically decrease by 10% to a FiO2 of 50%. At an FiO2 of 50%, an alarm will trigger requiring that a medical provider check on the patient and actively allow the device to continue weaning the FiO2.
3. A built-in safety system will be designed to maintain the target SpO2 or automatically increase the FiO2 if the patient's SpO2 is undetectable, decreases to < 88%, or drops under the set target SpO2.

Deliverable: Negative feedback control system for automatically weaning oxygen delivered through a HIFLO nasal cannula system to a critically ill pediatric patient using pulse oximetry.

# of Students Needed and Their Backgrounds: 3-4 students
Majors: Biomedical Engineering, Electrical Engineering, and Computer Systems Engineering

Company Name: Children's Healthcare of Atlanta
Company Contact Name: Pradip Kamat, MD, MBA and Jocelyn Grunwell, MD, PhD
Company Contact E-Mail: Pradip.Kamat@choa.org; jgrunwe@emory.edu
Company Contact Phone: 404-277-8010 (PK cell); 404-307-4697 (JG cell)

Problem Statement

Oxygen flow through a HIFLOW nasal cannula system is a noninvasive method that provides oxygen to a patient through a nasal tube. Compared to endotracheal systems, there is reduced risk of patient discomfort in a HIFLOW oxygen system. However, current HIFLOW oxygen systems require manual flow rate calibration by a bedside nurse technician, and overexposure to oxygen can result in fatal consequences such as hypoxia. An automated oxygen system that is capable of reading a patient's oxygen levels and reduce the oxygen flow to rate through a negative feedback system until the patient oxygen levels are stable is needed because it reduces the risk of miscalculated oxygen concentration and relieves the nurse to pursue other duties. There is currently no device in the United States that is capable of reading these oxygen sensors, having the ability to adjust oxygen flow rate, and alert a physician when oxygen levels are critical. We will design a device capable of these functions and be compatible with current HIFLOW systems, as per the design requirements of the client. So far, we have implemented a breadboard circuit and basic logic. The current circuit includes a pulse oximeter and servo motor.

Background

Oxygen therapy is frequently required for critically ill children in the pediatric intensive care unit (PICU) via endotracheal intubation and mechanical ventilation (invasive), non-invasive positive pressure ventilation (NIPPV) and, more commonly, through HIFLO nasal cannula systems. Exposure to high oxygen concentrations (hyperoxia) can result in oxidative damage to already damaged lungs. Infants born prematurely are especially sensitive to oxidative damage to their developing eyes, brain and lungs upon prolonged exposure to hyperoxia during necessary respiratory support. Respiratory therapists and nurses have many competing responsibilities and at times care for multiple patients in the PICU limiting their ability to quickly wean oxygen. The patient's oxygenation status is continuously monitored using pulse oximetry, arterial blood gas, or both. Pulse oximetry is non-invasive, but fluctuates depending on the patient's respiratory status and perfusion. Oxygen flow meters/blenders require bedside staff to manually adjust oxygen delivered.

On September 16, 2016, our group visited the Children's Healthcare of Atlanta to meet our sponsors, Dr. Kamat and Dr. Grunwell. They gave us a demonstration of the machine we are to improve for in our project, a combination of the Hudson RCI Neptune humidifier, an oxygen flowmeter, a nasal cannula, and the main component, the CareFusion BIRD Air-Oxygen blender. The oxygen blender's purpose is to mix together pure oxygen with normal air. The amount of oxygen supplied to a patient is known as fiO2, or fraction of inspired oxygen.

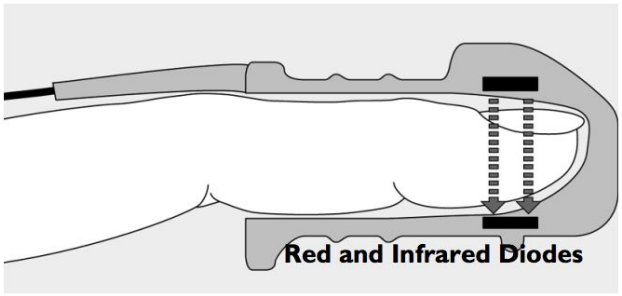
The CareFusion BIRD High-Flow oxygen meter (BDF11214) is the main component of the system. The purpose of this device is to blend a mixture of pure oxygen gas with humidified air to be delivered to the patient, earning the name, "blender". It uses a dial that indicates the the percentage of pure oxygen gas that will be mixed with air, ranging from 0% oxygen to 100% oxygen.

The humidifier is also an important component for adding moisture to the mixture, therefore increasing the comfort of the patient. This is achieved because delivering oxygen without adding humidity will dry out the mucous membranes of the nose and throat in the patient.

Another component of the blender system is the oxygen flow meter. The purpose of this component is to visualize and control the flow of oxygen from the blender to the nasal cannula. The ball in the meter is pushed upwards by the current of air, so the more air flowing through the tube, the more the ball will be pushed upwards. The knob to the right controls the valve through the flow meter, allowing the flow of oxygen to be increased or decreased.

The complete machine can be seen in the following image. In theory, the device is simple. Blend oxygen with air and humidify it, and deliver the mixture to the patient. Simple. The problem comes with the device not being an automated process. Physicians and nurses are required to continually monitor the patient's SpO2 and FiO2 levels throughout the process of the patient's oxygen therapy.

SaO2 stands for arterial oxygen saturation, or "Sat's", for short. It is an estimate of a person's oxygen-saturated hemoglobin blood compared to the total amount of hemoglobin available in the blood. A device called a pulse oximeter is used to measure the SaO2, and this measurement is referred to as SpO2. Pulse Oximetry is computed by a device that usually attaches to a fingertip, and sends a light signal from an LED through the finger and measures the amount of light that is received on the other side of the finger. The amount of oxygen in the blood is able to be estimated because oxygenated-hemoglobin absorbs a different amount of light at different wavelengths of light compared to unoxygenated-hemoglobin ("Understanding Pulse Oximetry"). An example of a pulse oximeter can be seen below. Pulse oximetry is performed by flashing an LED through a fingertip and measuring the intensity of the received light through a photodiode.



In a healthy person, the SpO2 should be above 94%, which means that 94% of the blood is oxygenated and is being used to distribute oxygen throughout the circulatory system. Less than 90% SpO2 indicates mild hypoxemia, a condition where tissues are not receiving sufficient oxygen. 70% or less SpO2 will lead to moderate hypoxemia and impaired brain function, and 55% or less SpO2 will lead severe hypoxemia and loss of consciousness, tissue damage, and death (Bottrell). Alternatively, hyperoxia is the condition where blood is oversaturated with oxygen. This is indicated by a prolonged SpO2 level greater than 100%, or overexposure to oxygen at increased partial pressures. This condition can just as well be a fatal condition to the patient because of the risk of development of oxygen toxicity (Kim et. al).

Another important factor will be monitoring FiO2, or fraction of inspired oxygen. This is a measurement of the amount of oxygen present in a space being measured. For example, oxygen gas makes up about 21% of the air at normal atmosphere conditions, which corresponds to 0.21 FiO2. An FiO2 value of 1.00 indicates 100% of a gas solution is oxygen. Where SpO2 is useful in measuring the amount of oxygen present inside a person's bloodstream, FiO2 is useful in measuring the amount of oxygen being applied to the patient. Where an SpO2 value greater than 100% can indicate hyperoxemia, an FiO2 value greater than 0.5 can potentially also cause hyperoxemia in at-risk patients (Bitterman). In short, the oxygen blender is responsible for controlling the FiO2 levels, which should in turn affect the SpO2 levels of the patient by maintaining, increasing, or decreasing the amount of oxygenated blood in the patient. Continuously monitoring these two values will be critical in the function of the automatic oxygen blender device.

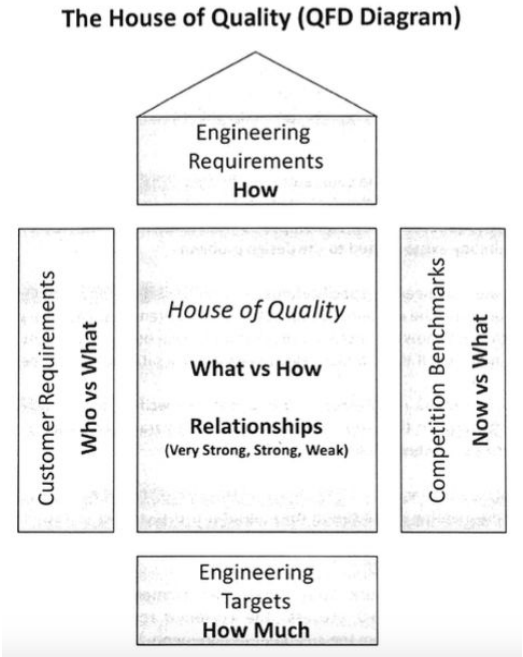
Technical Requirements

Through a combination of client and engineering specifications, we compiled a list of the targets we wish the device to meet. These requirements serve as guidelines and parameters when designing the device, as they cater to the features and constraints we need to implement. They different specifications are outlined below.

Stakeholder Requirements

The stakeholders are Children's Healthcare of Atlanta along with their patients who have trouble breathing on their own without the help of an oxygen delivery device. The stakeholder requirements were therefore given to us by the our hospital-staff mentor who requires that the product should have the certain key requirements outlined below. These requirements are necessary to automate the oxygen blending task in a safe and patient-first manner.

Specifications development	
Stakeholder Requirements	● Automate oxygen blending. ● Use non-invasive methods for SpO2 monitoring. ● Be cost-effective. ● Able to be sanitized. ● Compatible with ordinary electrical systems. ● Be simple to use. ● Should not require technical maintenance to operate. ● Be compatible or replace existing oxygen blender systems. ● Alert a medical staff member when SpO2 levels or FiO2 levels are unsafe.



These requirements would be useful to gain a basic understanding of the task and how to proceed with the project using design methodology.

Engineering Specifications

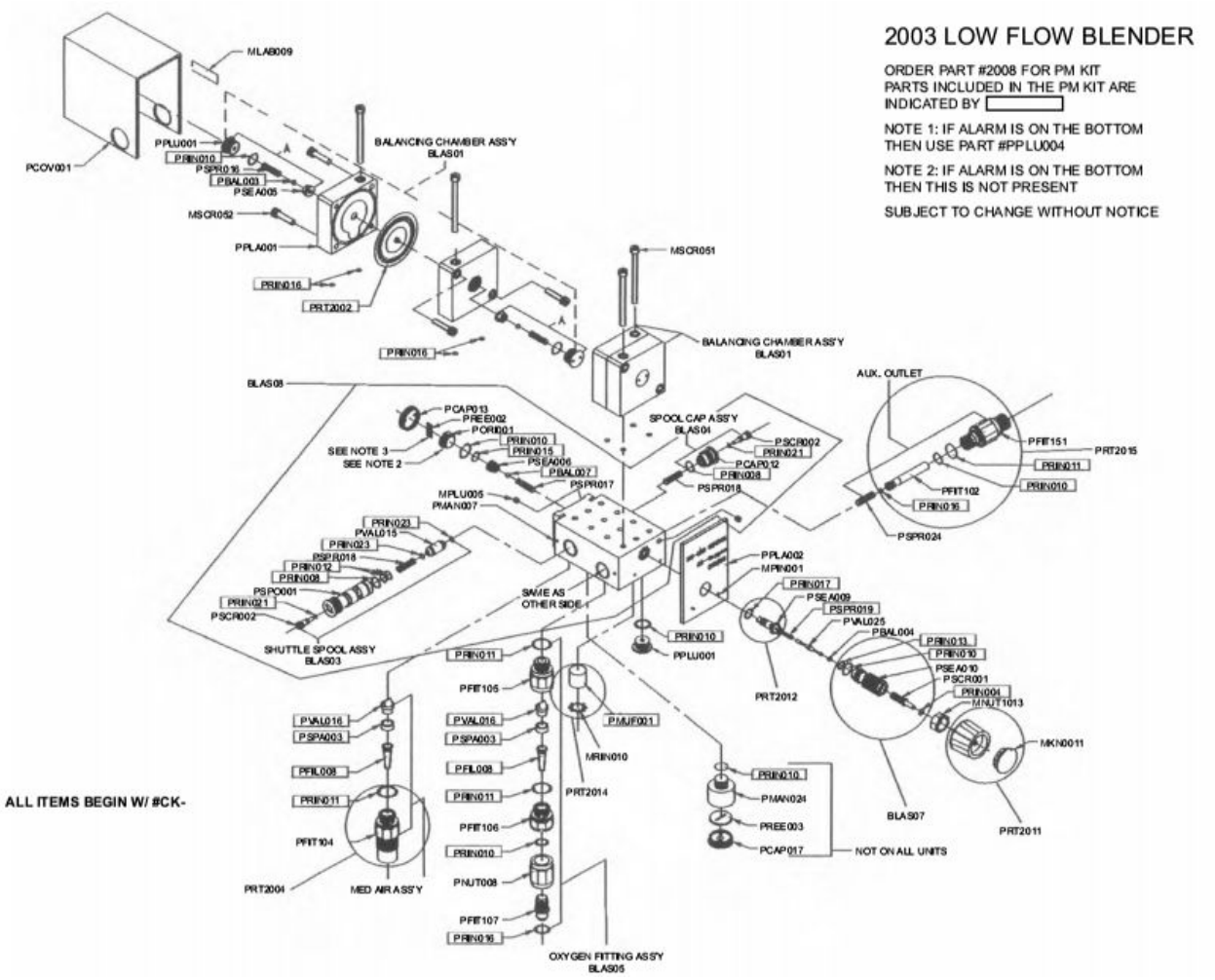
The engineering specifications outlined below are compiled from specific targets given by our mentor and engineering targets we desire that the final product would achieve.

Specifications development	
Engineering Requirements	● Automate oxygen blending using a negative feedback mechanism. ● Use pulse oximeter for noninvasive patient SpO2 monitoring. ● Difference between the target SpO2 and measured SpO2 will be used to adjust the FiO2. ● If SpO2 is above target for more than 30 mins, the FiO2 will decrease by 10%. ● At an FiO2 of 50%, an alarm will trigger requiring a medical provider actively allow the device to continue weaning the FiO2. ● A built-in safety system will be designed to maintain the target SpO2 or automatically increase the FiO2 under certain conditions. ● Be compatible or replace existing oxygen blender systems. ● Meet Clause 201.12.1 of ISO 80601-2-61:2011 for pulse oximeter testing purposes. ● Comply with section 201(h) of the Federal Food Drug & Cosmetic (FD&C) Act for medical device categorization. ● The pulse oximeter should should comply with ISO 80601-2-61:2011 Medical Electrical Equipment — Part 2-61: Particular requirements for basic safety and essential performance of pulse oximeter equipment ● Device should be enclosed and function while sterilized. ● Run through a standard power outlet.

Design Concepts

Internal Reverse-Engineering Modification plan

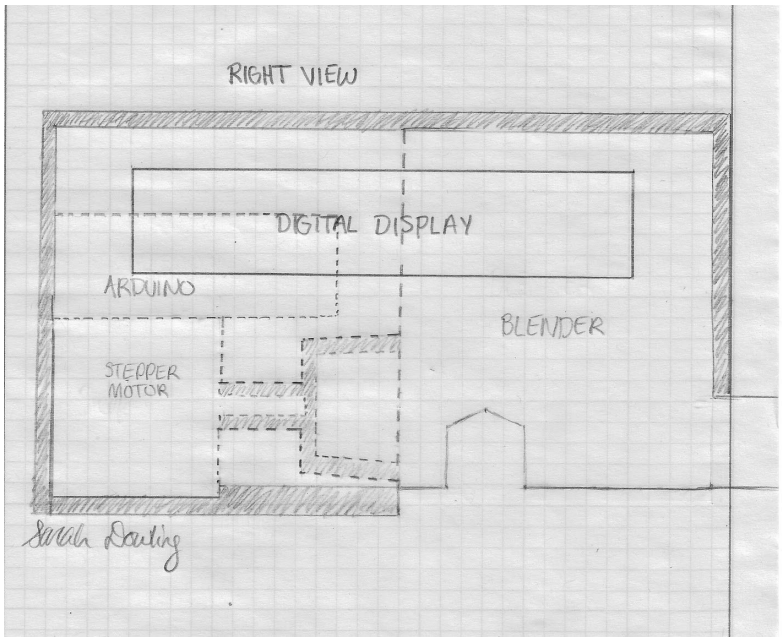
Initially, the group thought that modifying the valve on the oxygen blender that adjusts the ratio between the air and oxygen would be the best course of action. We thought this because internal modification would help the product retain it's FDA and ISO certificates. The problem with this would be that there would be increased liability with our group because we would have to modify the internal valve structure to automatically change the oxygen blending percentage. Aside from this, it would also be difficult to automatically control the butterfly valve in the oxygen blender through a microcontroller and reliably program and test the mechanism. Because of these obstacles, we decided that internal reverse-engineering would not be an effective choice for our approach, as seen from a generic oxygen blender schematic (“Bird Sentry Blender”).

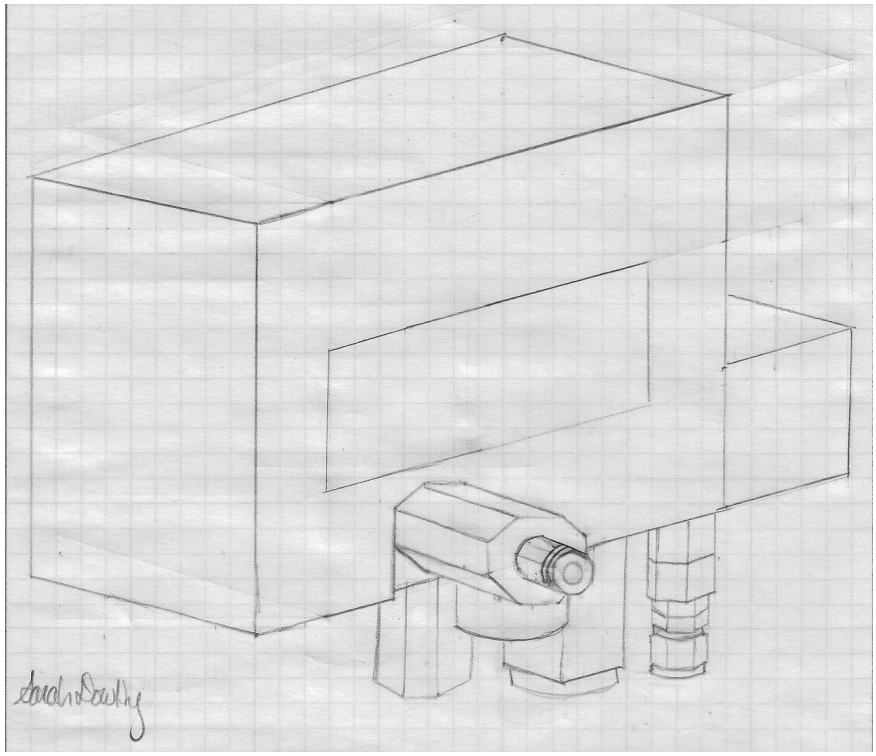


External-Attachment Modification plan

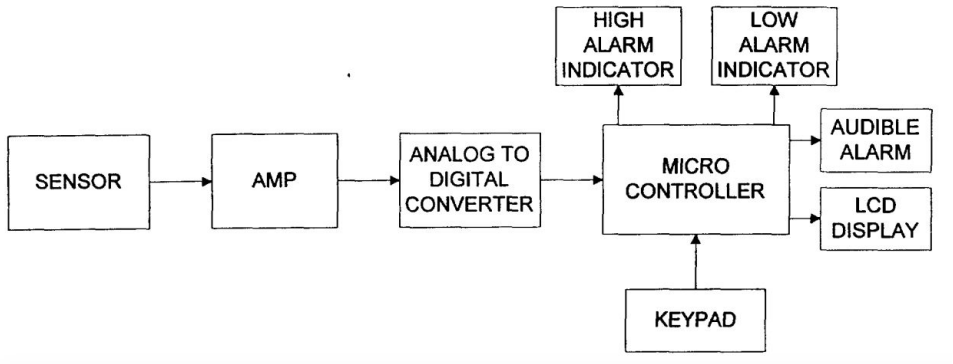
Our second approach would be to use an external control circuit to physically rotate the knob on the oxygen blender to adjust the FiO2 automatically. We decided that this approach would be more cost effective, time effective, and be backwards compatible with the general pre-existing oxygen blender system. This would also reduce the complexity of the control circuit. This approach brings the issue of how to control the knob on the oxygen blender. For most of the time we decided on using a servo to control knob would be adequate. The servo would be controlled from a microcontroller, which would be a simple process as most servos can be programmed to rotate to a specified angle and hold that position until further notice. This is how our current implementation functions.

Thirdly, we wish to improve on the concept by switching the servo with a stepper motor. A servo would be adequate for the design but a servo requires constant power to hold it's position. Not only that but servos are also noisy in their operation and may be prone to failure. The biggest factor however is the fact that most servos can turn at most 180°, and the knob on the oxygen blender has a rotation arc of about 240°, which is insufficient to be fully compatible with the servo. There are servos that offer full 360° rotation, but they lose their tracking in that the microcontroller could no longer call servo.write(60°) and expect the servo to turn to 60°, as full-rotation servos lack reference tracking mechanisms when they switch gears to fully turn around. As an alternative, a stepper motor would be useful in the proposed product's rotation mechanism as stepper motors are typically used in precision tracking operations (Bill). The downside to using a stepper motor is that they lack a tracking mechanism and only count the number of steps from their initial movement. For this compensation, a tracking mechanism will have to be improvised. Sketches for our proposed project can be seen below.





Using the touch screen control, respiratory therapists will have the authority to set the SpO2 lower limit, typically 93%, and the FiO2 starting value at a level deemed necessary, depending on the patient's need at the start of the treatment. The use of touch screens to implement data is meant to make our product easily accessible and adjustable. The product will then begin to reduce the FiO2 levels by 10% every hour if the patient's SpO2 levels exceed the target set at the beginning of treatment, while alarming nurse technicians at 60%, 40%, and 25% FiO2 for permission to continue weaning. It was very clear that our mentors' goals were not to take the technicians out of the equation, but instead to make this specific task simpler. In cases where the patient's SpO2 levels do not meet the target value, the alarm will sound for the respiratory therapist to manually adjust the FiO2 level and monitor the patient. The device should follow a flow similar to the diagram outlined below.



For concept generation we the stakeholder specifications were analyzed for importance by using a Quality Function Deployment (QFD) diagram shown below.

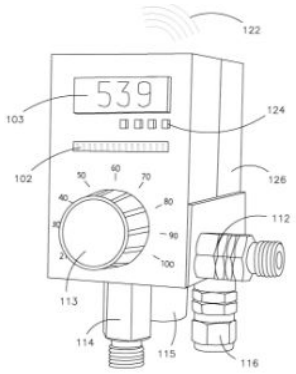
				Column #	1	2	3	4	5	6	7	8
				Direction of Improvement: Minimize (▼), Maximize (▲), or Target (x)	x	x	x	▲	▼	x		
Row #	Max Relationship Value in Row	Relative Weight	Weight / Importance	Demanded Quality (a.k.a. "Customer Requirements" or "Whats")	Quality Characteristics (a.k.a. "Functional Requirements" or "Hows") Feedback loop	display monitor	graph	audible alarm	small size/weight	plug-in w/cordless capability	remote control	sealed container
1	3			lower limit SpO2 of 93% initially	O	O						
2	3			target SpO2 may be lowered to 88-90% to prevent hyperoxia		O						
3				difference between target SpO2 and measured SpO2 will be used to adjust FiO2								
4	9			SpO2 above target for >30 min, the FiO2 will automatically decrease by 10% to a FiO2 of 50%	⊖							
5	9			if FiO2>50, alarm triggered	⊖			⊖			O	
6	9			checked by provider at FiO2>50 to actively allow device to continue weaning the fiO2		⊖						
7	9			automatically change FiO2 if patient's SpO2 is undetectable, decreases to <88% or drops under set target SpO2	⊖							
8	9			mobile					⊖	⊖	O	
9	9			sanitizable								⊖
10	9			trended data			⊖					

Our sponsors can be seen in the image above.

Benchmarking

Competitive benchmarking was carried out to evaluate automatic oxygen weaning devices that are currently in the market. The most advanced oxygen delivery system we found was the FreeO2 by Oxy'nov shown below. It is an automated oxygen blender created by a French company that enables automatic titration of oxygen flow based on patient response and saturation level set by the respiratory therapist, nurse, or doctor. The FreeO2 provides continuous monitoring and automatic weaning, but it is not available in the United States because it is not currently FDA certified. We will aim to produce a product similar to the Free O2, but be more cost-effective and backwards-compatible with current oxygen blenders.

The other device benchmarked was the Auto-Controlled Air-Oxygen Blender by Richard Caso shown below. It is a simplistic, efficient design that consists of oxygen input, gas input, combined gas output, and electronic mixing valve (knob or CPU-controlled) which mixes medical air with hospital-grade oxygen (21%-100%) (Caso). While the design has been patented, no final product has ever been created. Instead of trying to invent a new oxygen blender, we will use a simple motor and sleeve which will be placed over existing oxygen blenders to automatically wean the patient off of HIFLO oxygen.

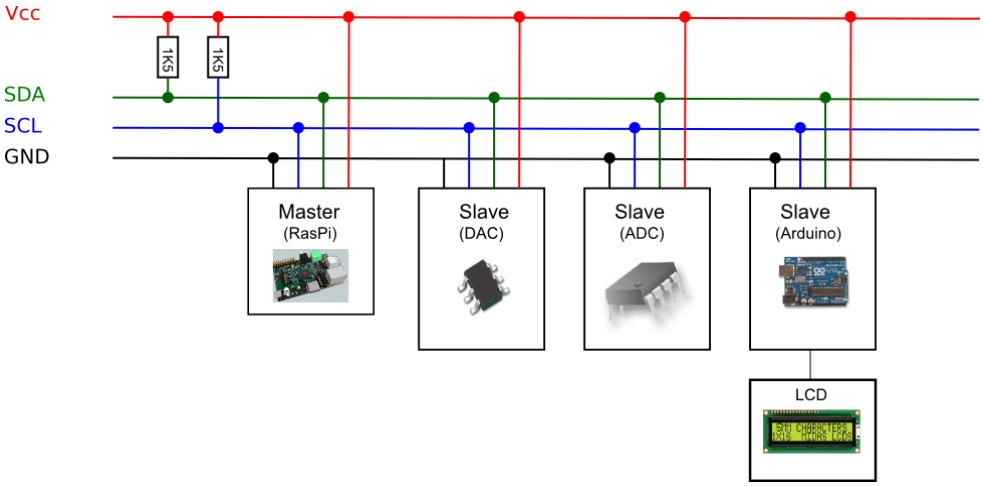


New Knowledge Developments

This project combines biomedical concepts with electrical and computer engineering applications, so all of our strengths should have been useful in the development of this project. We had to research pulse oximetry in order to gain an understanding of how pulse oximetry works and how it is implemented in a sensor. For our sensor, we had to learn how to read it's values from I2C communication and how these values correspond to oxygen levels.

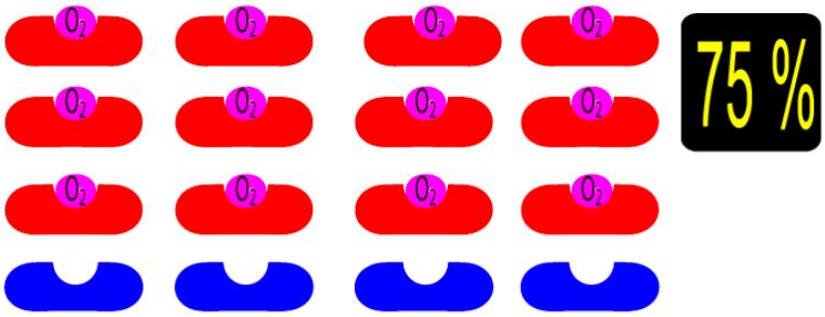
I2C

I2C (Inter-integrated Circuit) communication is a protocol that can be used to deliver digital data to multiple devices from a single source. I2C features a SDA and a SCL bus line. SDA functions as the data line while SCL functions as the clock line to keep the slave devices in sync with the clock of the master device. I2C works by sending a start, stop, and acknowledge condition with each transmission of data. The advantage of I2C communication over serial and SPL communication is that I2C only requires the two aforementioned wires, and supports up to 1008 slave devices (Hord). To establish an I2C connection, the devices must be in communication with each other's device registers. In a sensor application, the device must wake up the sensor, clear it's registers, instruct the sensor to write its data to its registers, and the microcontroller must read the data into its own register for processing. An example of an I2C setup can be seen below.



Pulse Oximetry

We also had to learn about oxygen and oxygen concentration in the blood. The diagram below demonstrates what blood-oxygenation is (“How pulse oximeters work explained simply”).



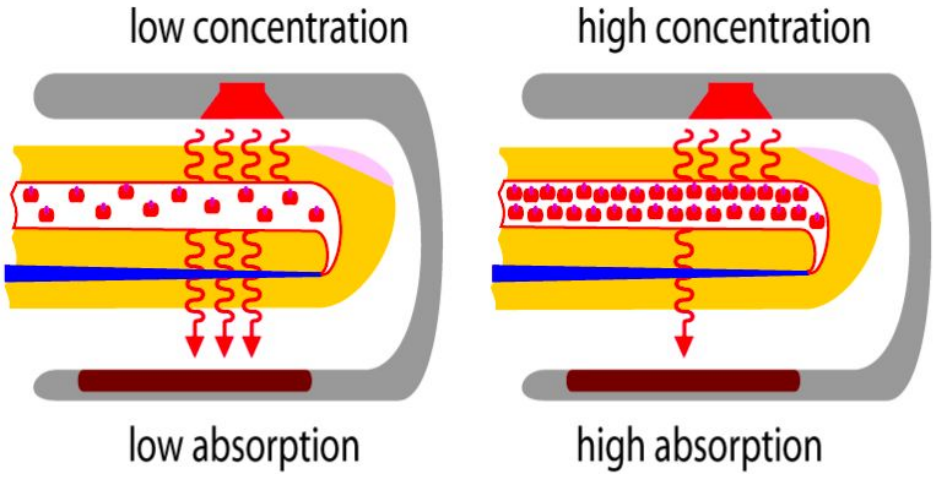
The red cells represent hemoglobin that has been oxygenated, while the blue cells represent hemoglobin that has been deoxygenated. In the figure, 75% of the cells contain oxygen, which would imply a SpO2 of 75%, as it is a measure of oxygenated-hemoglobin to total hemoglobin, given by the equation below.

$$SpO_2 = \frac{HbO_2}{HbO_2 + Hb}$$

SpO_2 : *Oxygen saturation of arterial blood*

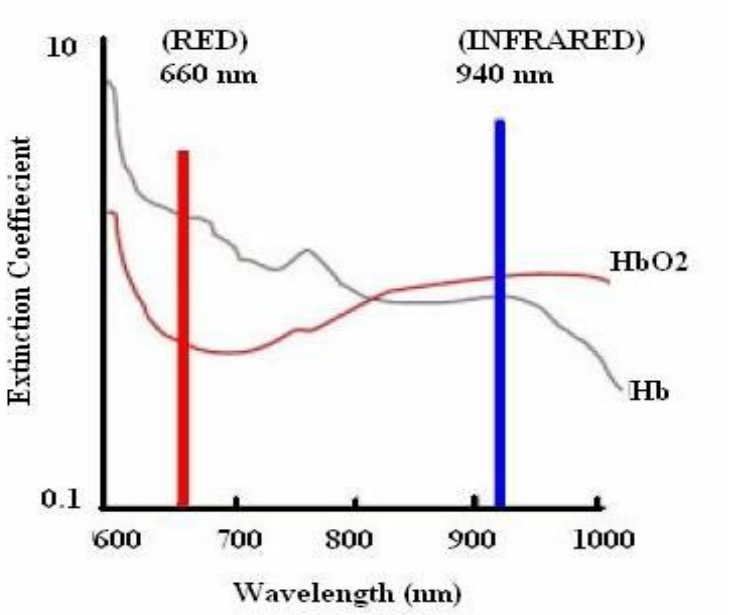
HbO_2 : *Molar concentration of oxygenated hemoglobin*

Hb : *Molar concentration of deoxygenated hemoglobin*



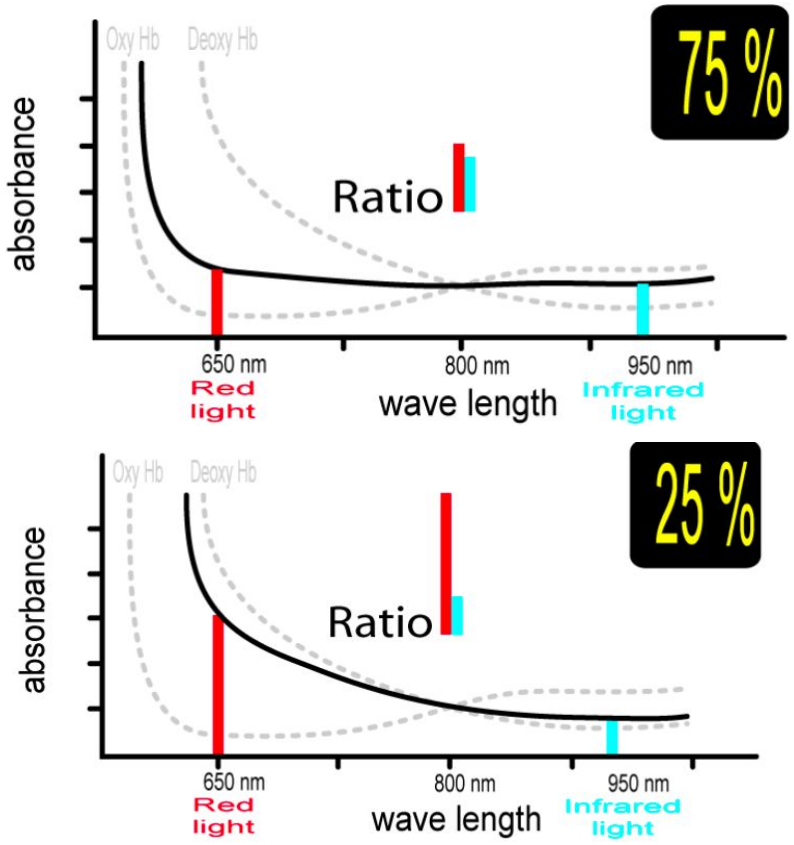
An led and photodiode are used to measure the absorbance (extinction rate) of light through the hemoglobin (“How pulse oximeters work explained simply”).

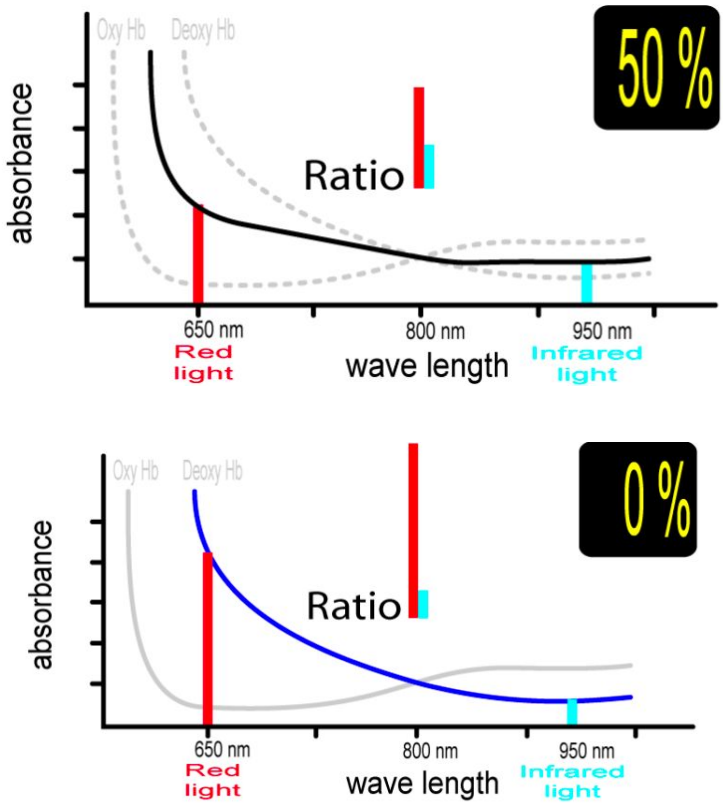
Since oxygenated hemoglobin absorbs light at a different rate than non-oxygenated hemoglobin, this percentage can be calculated, as the figure below shows (Iddir et. al).



Using these different percentages of absorption, pulse oximetry works by calculating the percent of oxygenated to non-oxygenated hemoglobin.

Pulse oximeters cannot calculate the ratio of oxygenated and deoxygenated hemoglobin directly as that would require counting the individual blood cells and their oxygen status. Instead, pulse oximeters use the ratio of absorption from oxygenated and deoxygenated hemoglobin at different wavelengths. This is done by using a red LED, an infrared LED, and a photodiode. An example of this ratio relationship can be seen below (“How pulse oximeters work explained simply”).


Oxy Hb
Deoxy Hb
75 %
absorbance
Ratio
650 nm
Red
light
800 nm
wave length
950 nm
Infrared
light
Oxy Hb
Deoxy Hb
25 %
absorbance
Ratio
650 nm
Red
light
800 nm
wave length
950 nm
Infrared
light



The absorbance is an estimation of the concentration of oxygenated and deoxygenated hemoglobin. The photodiode is responsible for measuring the amount of light from each respective LED, refered to as the reflectance, and converting this into a current. The currents produced from the photodiode by each LED is used as a ratio to approximate the absorbance and therefore the SpO2, using the equation:

$$R = \frac{I(\lambda_R)}{I(\lambda_{IR})}$$

This ratio, R, is related to the absorbance of the hemoglobin and therefore the SpO2 as well.

To understand absorbance, the Beer-Lambert law relates absorbance to the properties of the material through which the light is traveling (Clark).

Light traveling through a medium will be absorbed.

$$I = I_0 e^{-\varepsilon l c}$$

I : *intensity of light*

I_0 : *starting intensity of light*

ε : *absorptivity of light at wavelength* λ

c : *molar concentration of medium*

l : *light path length*

This equation can be manipulated to account for the transmittance, T, the ratio of transmitted light to starting intensity.

$$T = \frac{I}{I_0} = e^{-\varepsilon lc}$$

T : *transmittance of light through medium*

The beer-lambert law arises from taking the natural logarithm of both sides of the equation to result in the absorbance of light.

$$A = -ln(T) = \varepsilon lc$$
$$A = -ln(\frac{I}{I_0}) = \varepsilon lc$$

A : *absorbance of light through medium*

The above equation is the Beer-Lambert law, which is useful for calculating the concentration of a medium at a specified absorbance using a spectrometer. However, in the human body, arteries pulsate and the skin is composed of different absorptivities at different points, so modifications are necessary to account for these variables (Kennedy). There should be two measurements to be taken at the minimum and maximum points of a pulse.

Allow α to be the function of ε, absorptivity at wavelength λ and the medium concentration, c.

$$\alpha = \varepsilon(\lambda)c$$

Let the minimum intensity from the baseline of a pulse, and the maximum intensity from the pulsation be defined as:

$$I_{min} = I_0 e^{-\alpha_{min}(\lambda)l}$$
$$I_{max} = I_{min} e^{-\alpha_{max}(\lambda)\Delta l} = I_0 e^{-\alpha_{min}(\lambda)l - \alpha_{max}(\lambda)\Delta l}$$

The change in transmittance is therefore the ratio of maximum pulse intensity to minimum pulse intensity:

$$\Delta T = \frac{I_{max}}{I_{min}} = \frac{I_0 e^{-\alpha_{min}(\lambda)l - \alpha_{max}(\lambda)\Delta l}}{I_0 e^{-\alpha_{min}(\lambda)l}} = e^{-\alpha_{max}(\lambda)\Delta l}$$

The absorbance is therefore:

$$\Delta A = -ln(\Delta T) = -ln(e^{-\alpha_{max}(\lambda)\Delta l}) = \alpha_{max}(\lambda)\ \Delta l$$

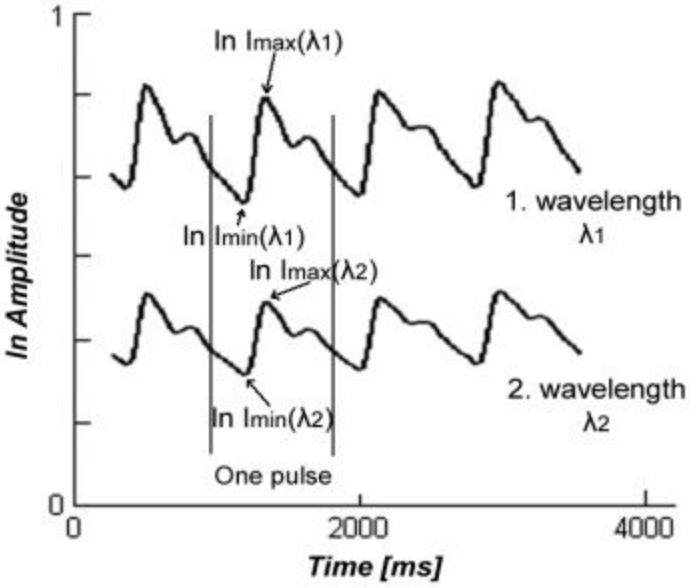
The term Δl may be dropped for simplification if the light will go through approximately the same distance at each measurement. For pulse oximetry, light in the red and infrared wavelengths are used.

$$\Delta A(\lambda_R) = \alpha_{max}(\lambda_R)$$
$$\Delta A(\lambda_{IR}) = \alpha_{max}(\lambda_{IR})$$

Allow the ratio R to be the ratio of red to infrared LED photodiode current:

$$R = \frac{\Delta A(\lambda_R)}{\Delta A(\lambda_{IR})} = \frac{\alpha_{max}(\lambda_R)}{\alpha_{max}(\lambda_{IR})}$$

The transmittance can be visualized from the graph below, as the difference between the minimum and maximum light intensities from the pulsation (Stuban and Masatsugu).



$$\Delta T(\lambda_R) = \frac{I_{max}(\lambda_R)}{I_{min}(\lambda_R)} \quad \text{and} \quad \Delta T(\lambda_{IR}) = \frac{I_{max}(\lambda_{IR})}{I_{min}(\lambda_{IR})}$$

Solving for the absorbances using the Beer-Lambert Law results in:

$$\Delta A(\lambda_R) = ln(\Delta T(\lambda_R)) = ln\left(\frac{I_{max}(\lambda_R)}{I_{min}(\lambda_R)}\right)$$
$$\Delta A(\lambda_{IR}) = ln(\Delta T(\lambda_{IR})) = ln\left(\frac{I_{max}(\lambda_{IR})}{I_{min}(\lambda_{IR})}\right)$$

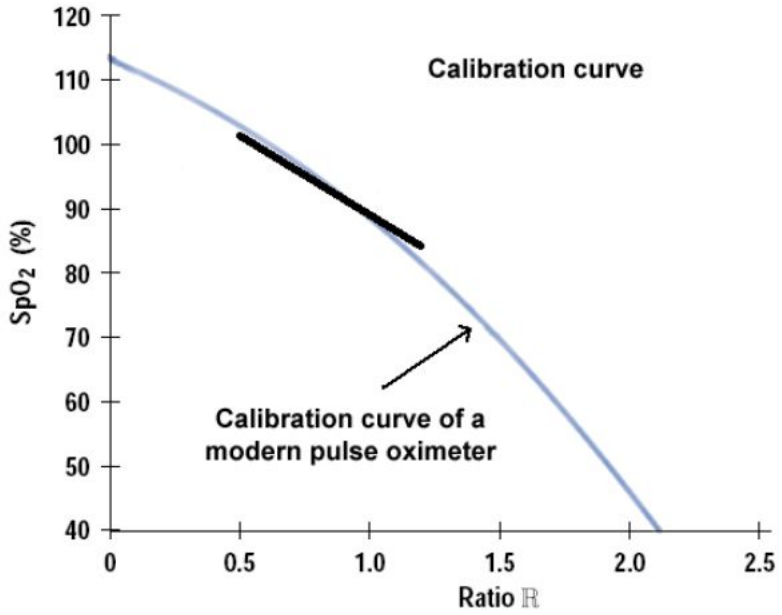
Using the ratio equation of red to infrared LED photodiode current:

$$R = \frac{\Delta A(\lambda_R)}{\Delta A(\lambda_{IR})} = \frac{ln\left(\frac{I_{max}(\lambda_R)}{I_{min}(\lambda_R)}\right)}{ln\left(\frac{I_{max}(\lambda_{IR})}{I_{min}(\lambda_{IR})}\right)}$$

For simplicity, this ratio can be estimated as:

$$R = \frac{ln(I_R(\lambda_R))}{ln(I_{IR}(\lambda_{IR}))}$$

This ratio is used to estimate SpO2 through an empirical calibration curve, such as, % SpO2 = 110 – 25 × R, which should be tested against healthy patients and known SpO2 levels from using invasive measures (Oak and Aroul).



Design for X

With the project goal in mind, we developed our initial idea from various specifications we hope to meet. For example, we want our product to be safe, cost-effective, and and medically beneficial so our design should reflect these intentions.

Design for Safety

Since this product will be used in a medical environment, the product must be able to be sterilized. To do this, we propose enclosing the hardware in a sealed case while making the rotation mechanism assembly removable from the air-oxygen blender for simple cleaning, sterilizing, and maintaining. The air-oxygen blenders must be airtight so that a correct ratio of air to oxygen is delivered. The enclosure should also protect the user from any physical contact with the moveable components of the product. This enclosure should also be electrically isolating, made of fire-resistant material and not contain any small chokeable hazard pieces.

Design for Cost

Since hospitals already have inventory of non-automated air-oxygen blenders, it would be most cost effective to create a device that works with the current units rather than create an automated blender. The blenders are at least $1000 per unit, so replacing all of the units in one hospital would be very expensive. Designing a blender would also be very expensive because we would have to acquire one for that price in order to modify it, then certify that it operates correctly under automated conditions, and require that every old oxygen blender be replaced with the automated unit. With this in mind, we produced a cost proposal, as seen below.

Item(s)	Vendor / Weblink	Unit Price ($)	QTY.	Total Cost ($)
Servo Motor	https://www.adafruit.com/products/1404?gclid=CjwKEAjw-abABRDquOTJi8qdojwSJABt1S1OMmRWbnTNAVQFsj_p_Rhq77AdQ1JkNbgrlSb3F9j_mRoCo3Dw_wcB	$14.95	1	$14.95
ArduinoUNO	https://www.amazon.com/Arduino-Uno-R3-Microcontroller-A000066/dp/B008GRTSV6/ref=sr_1_3?s=pc&ie=UTF8&qid=1477095772&sr=1-3&keywords=arduino+uno	$23.99	1	$23.99
Pulse Oximeter	http://www.aliexpress.com/store/product/Heart-Rate-Click-MAX30100-Sensor-for-Arduino/2340056_32699390906.html	$3.71	1	$3.71
Alarm	https://www.amazon.com/Arduino-Compatible-Active-Buzzer-Module/dp	$2.995	2	$5.99

	/B019GUDNRQ/ref=sr_1_3?s=pc&ie=UTF8&qid=1477097214&sr=1-3&keywords=arduino+alarm			
Digital Displays	https://www.amazon.com/Kuman-Shield-Display-Arduino-MEGA2560/dp/B01C466H1S/ref=sr_1_1?s=electronics&ie=UTF8&qid=1477098068&sr=1-1&keywords=arduino+display+buttons	$9.90	2	$19.80

Total Requested:
$68.44 (before tax and shipping costs)

Cost justification:
Engineering Capstone project required for graduation

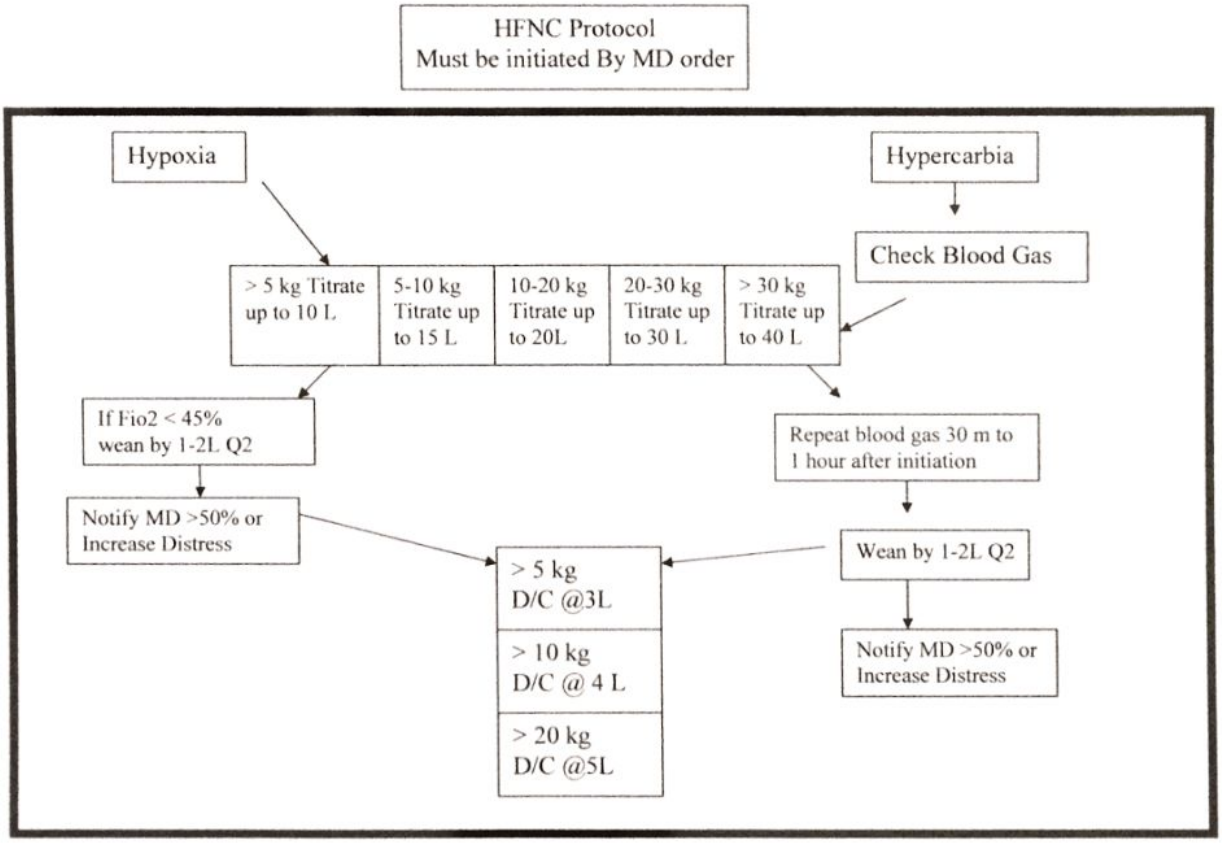
Physical Space Requirements:
None

Design for Medical Applications

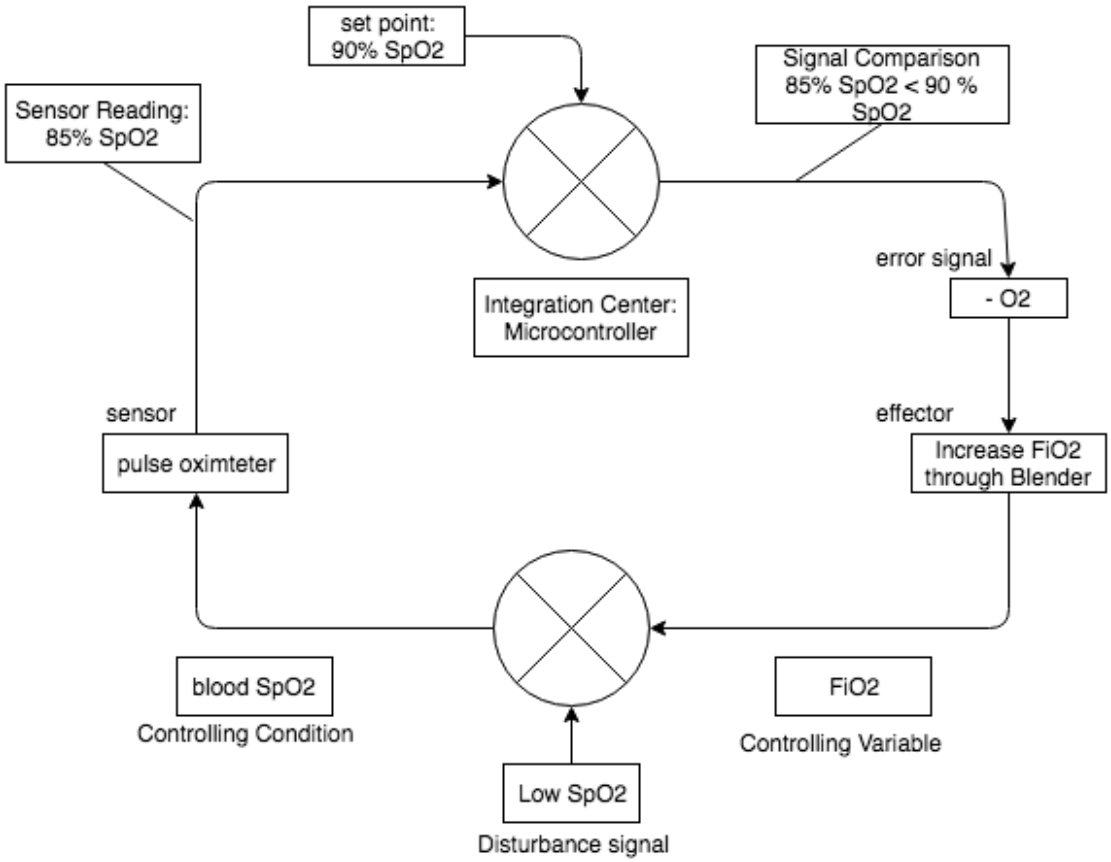
We want our product to be able to meet the guidelines for FDA and ISO certification. More specifically we would want the product to meet Clause 201.12.1 of ISO 80601-2-61:2011 for pulse oximeter testing purposes. This guideline requires extensive testing of the pulse oximeter to be safely used on patients in real world applications. From this same certification, the pulse oximeter should should comply with ISO 80601-2-61:2011 Medical Electrical Equipment — Part 2-61: “Particular requirements for basic safety and essential performance of pulse oximeter equipment”. We also want the product to comply with section 201(h) of the Federal Food Drug & Cosmetic (FD&C) Act for medical device categorization. This certification serves as the establishment of the product to be able to be used in a medical setting, which requires testing for approval. In order to meet these guidelines we will have to test on real patients through different conditions such as verifying their SpO_2 levels, testing with different skin pigments, different blood pH’s and different temperatures. We could implement a medical-grade pulse oximeter to reduce the risk of liability and ensure that our product would function better than with an untested pulse oximeter.

Evaluation

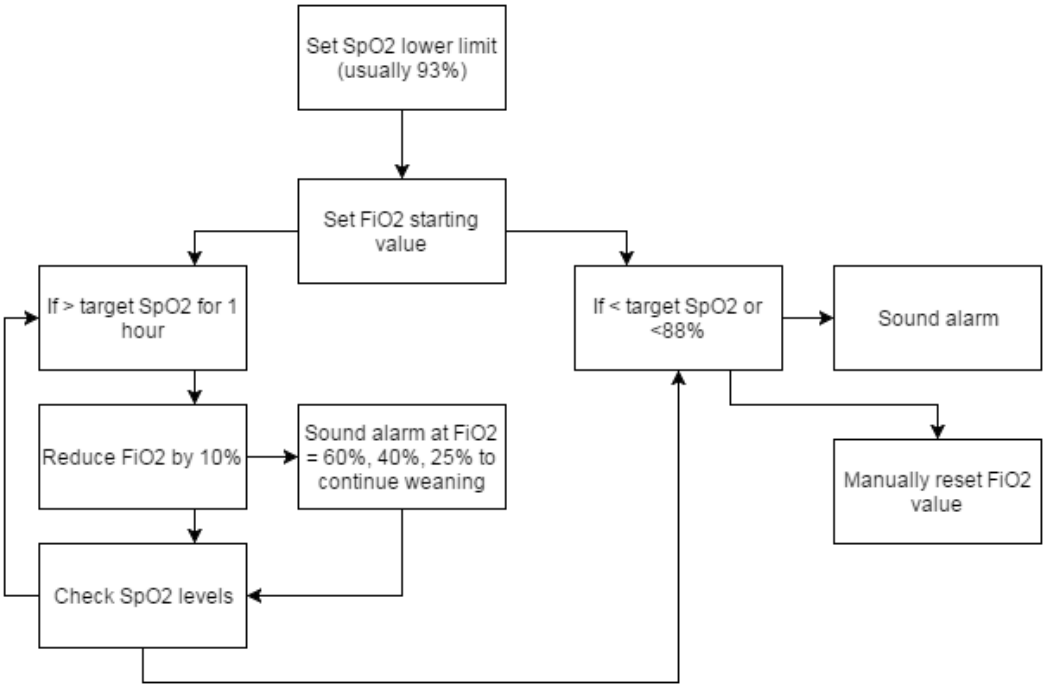
The following diagram is the protocol that the Children's healthcare of Atlanta follows when using their oxygen-blending systems.



It is important to note that this protocol requires manual FiO_2 adjustment, and more notably that the weaning of the FiO_2 is at the nurse's discretion as they must see how the patient reacts to a lower FiO_2 level before continuing to decrease or increase the supplied oxygen flow. With this in mind, a negative feedback loop could be implemented to better describe this process, as seen in the diagram below.



With this in mind, a flowchart of the process logic was created to better understand how to proceed with the project.

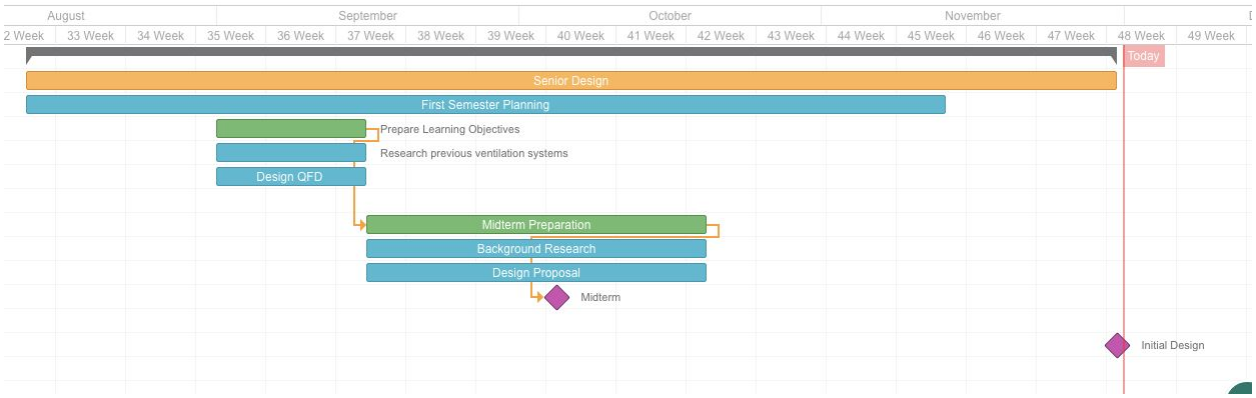


baseLogic

```
int SpO2; //read this from pulse-ox
int target;
int FiO2 = 90;
int count;
int alarmStatus = 0;
Servo serv1;

void setup()
{
  Serial.begin(115200);
  serv1.write(90);
}


void loop()
{
  if(count >= 3600) //SpO2 greater than target for an hour
  {
    FiO2 = FiO2 - 10;
    count = 0;

    if (FiO2 == 60 || FiO2 == 40 || FiO2 == 25)
    {
      alarmStatus = 1;
    }
  }

  else if (SpO2  > target)
  {
    count++;
    Serial.println(count);
    delay(1000);
  }
  else //SpO2 less than target
  {
    //manually set FiO2;
    count = 0;
    alarmStatus = 1;
  }
}
```

The initial logic can be seen in the figure above.

We produced a Gannt chart to follow through the first semester of this project.



At first, we looked for pulse oximeter sensors that are compatible with Arduinos. We stumbled upon the MAX30100 sensor by Maxim Integrated, which features integrated LEDs, Photo Sensor, and High-Performance Analog Front-End. This sensor uses I2C to communicate with the microcontroller.

It's I2C timing diagram can be seen below.

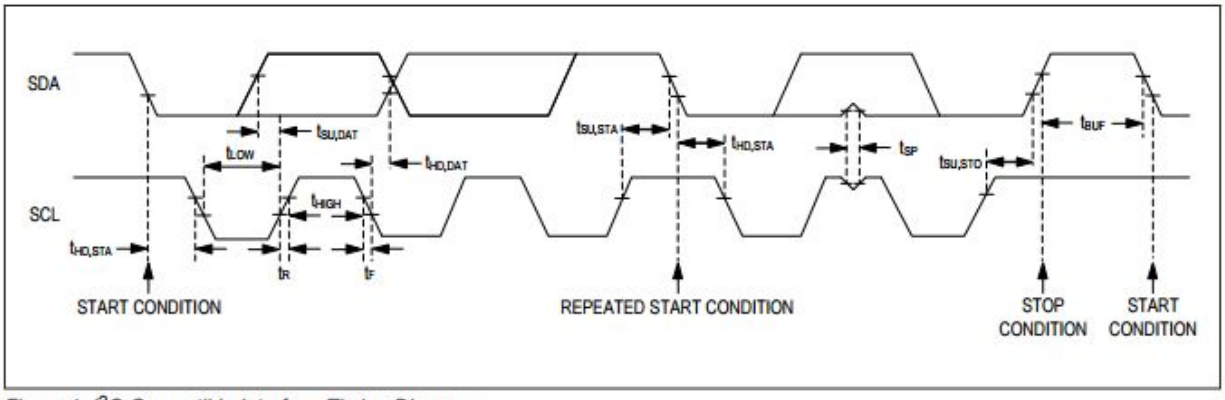
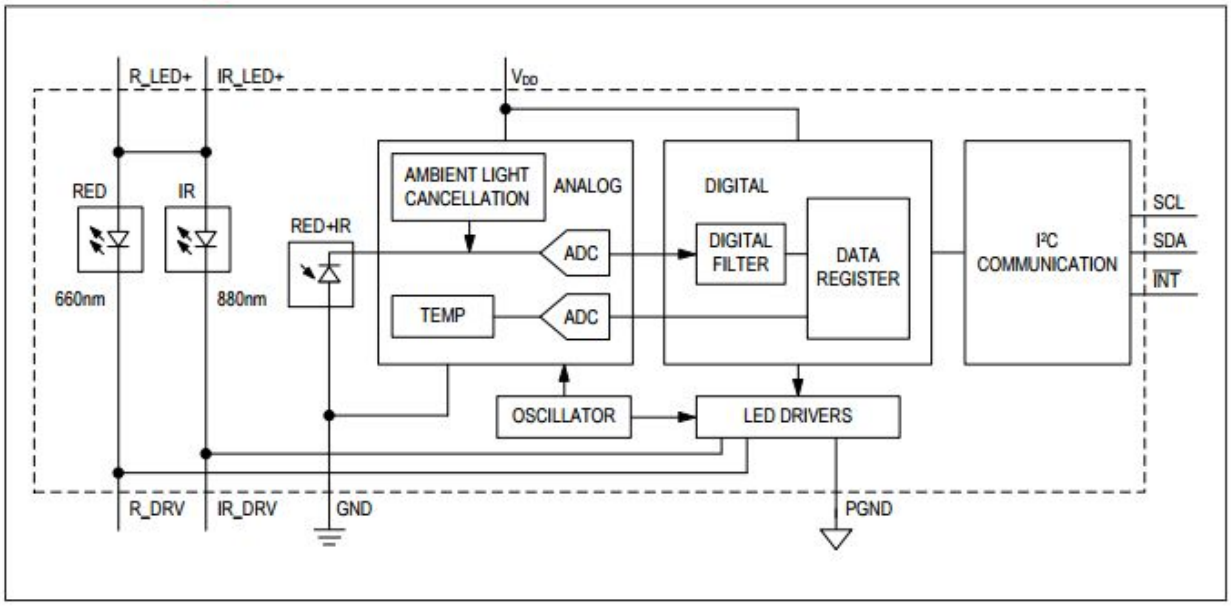
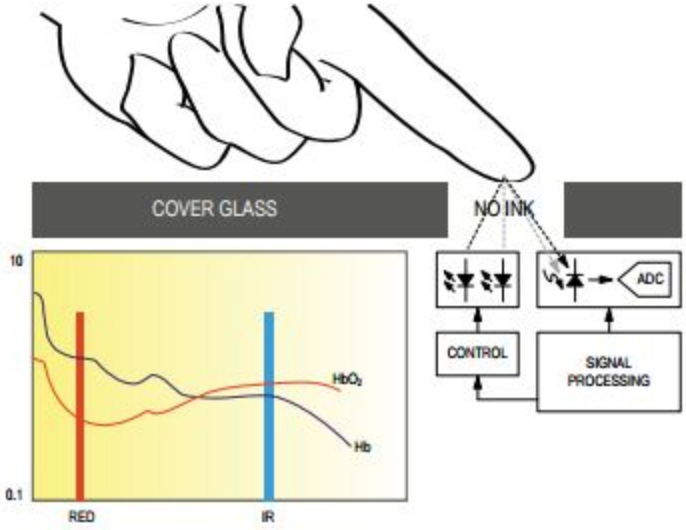


Figure 1. I²C-Compatible Interface Timing Diagram

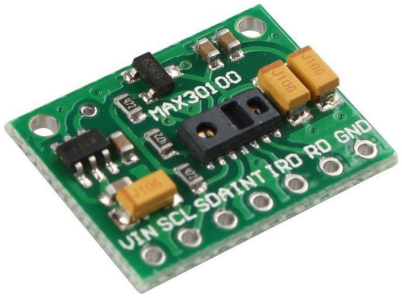
The following schematic shows the internal components that make up the MAX30100 sensor.

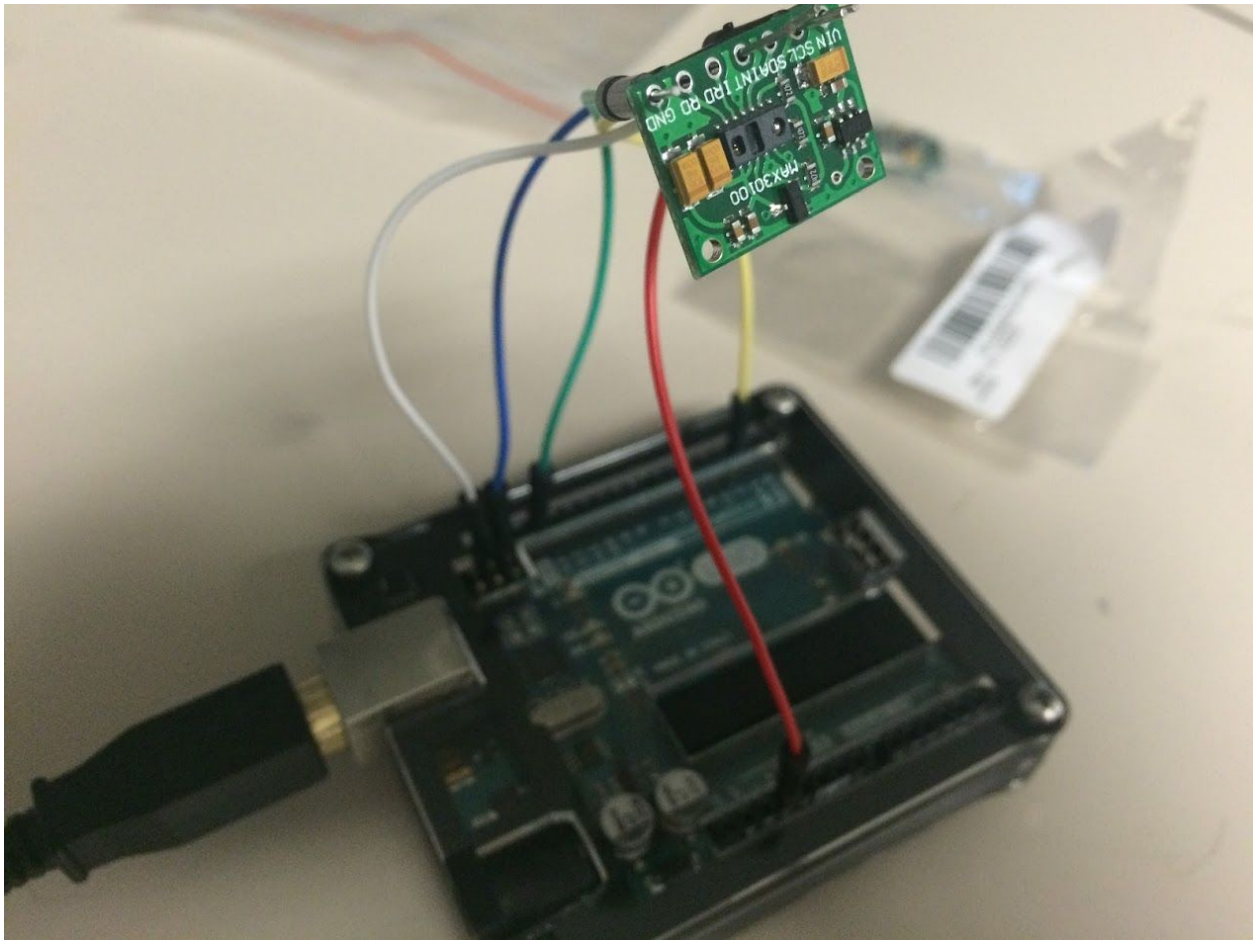


The following diagram shows how the sensor is to be used, which indicates it uses reflective pulse oximetry rather than transmissive. The difference between the two is that reflective pulse oximetry has the LED and the photodiode on the same side while transmissive pulse oximetry has the LED and photodiode on opposite sides of the finger, but both behave in the same manner.

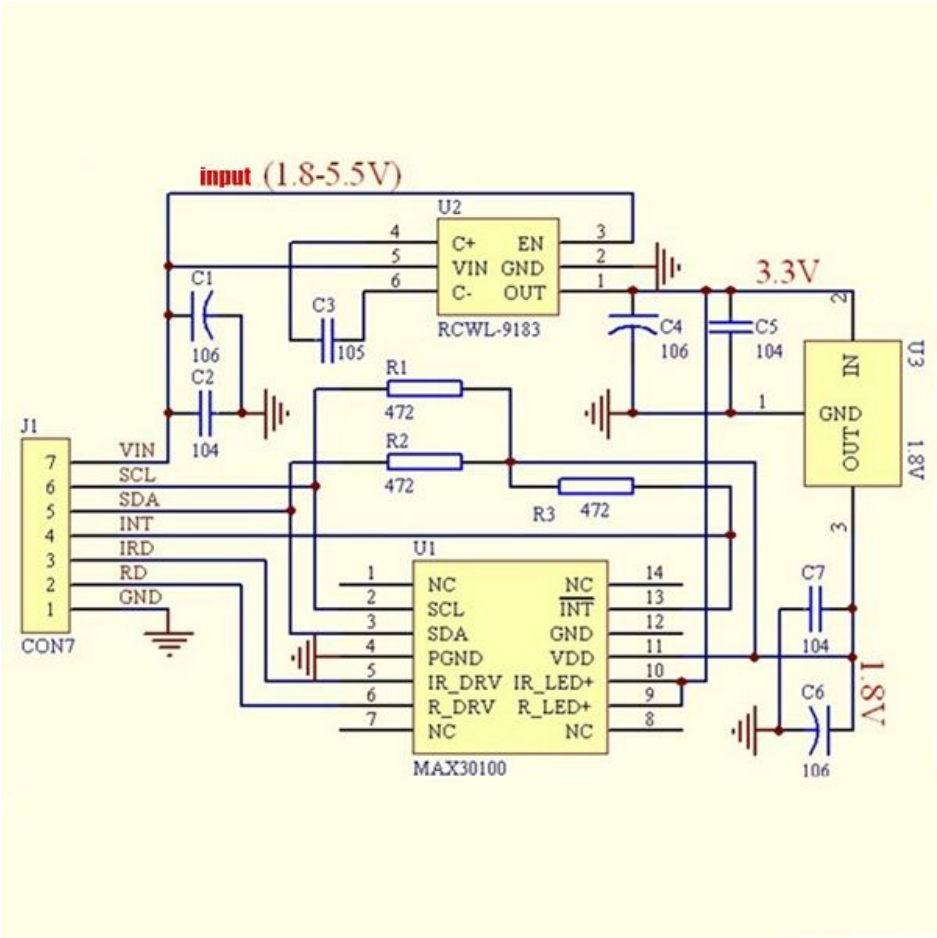


We initially purchased the RCWL MAX30100 breakout board, which can be seen below. Our testing can be seen below.

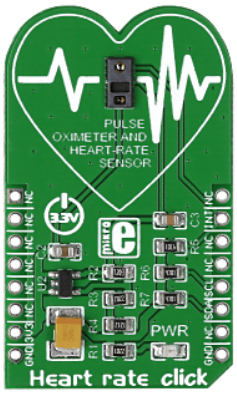


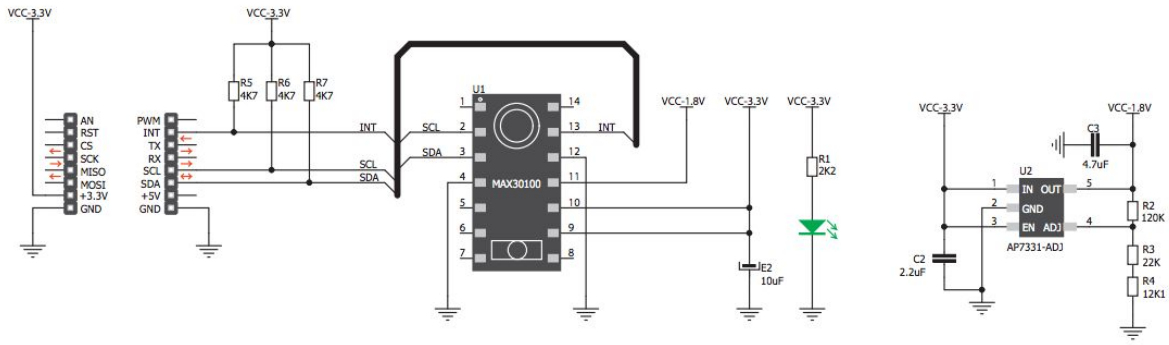


A system schematic for what we based our layout can be seen below.

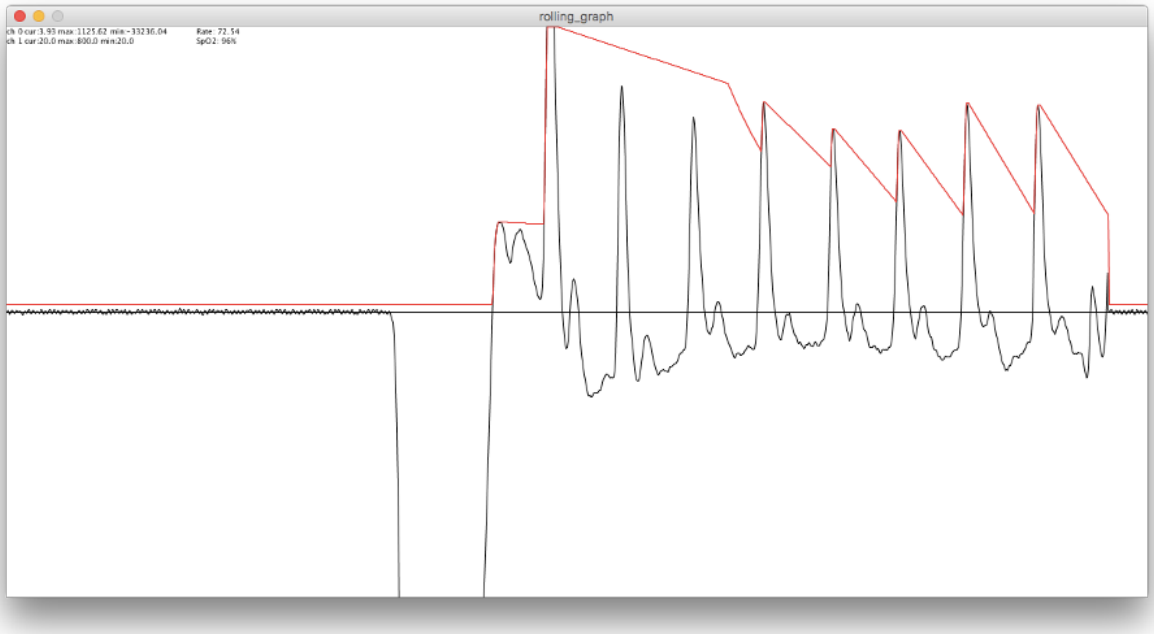


However, we could not get this breakout board to function correctly, and decided that we would need to pursue a different pulse oximeter sensor, and we stumbled upon the “Heart Rate Click” by MikroElektronika (“Heart Rate Click, 2010). Again, it also uses the MAX30100 sensor so it should have the same 1600 µs pulse width and 16-bit ADC resolution as the RCWL breakout board.

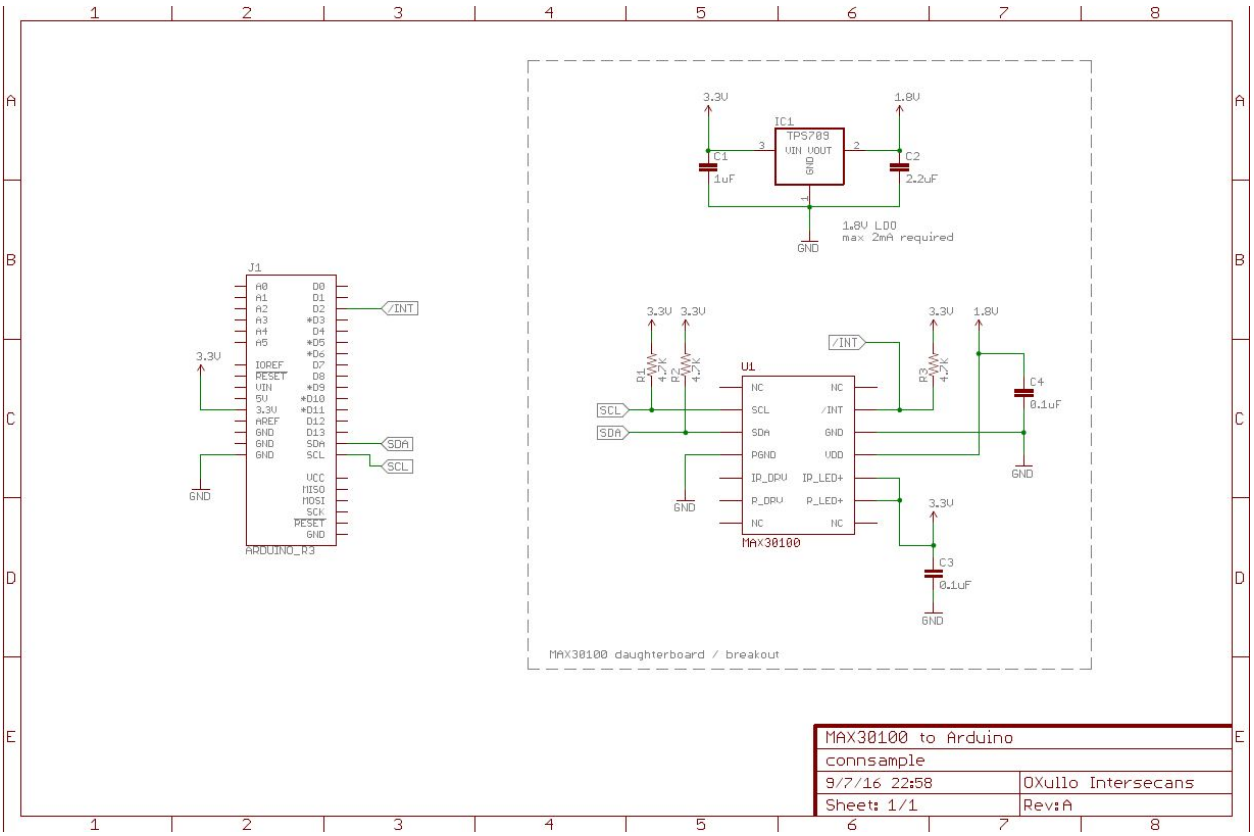




Using this sensor, we were able to achieve results, an example of which can be seen from the graph, using example code and wiring by Oxullo and the MAX30100 arduino library.



We were able to read an SpO_2 level that stayed around 94%, which given that this is in good range of oxygen concentration, seems like it was good data.



When using the sensor, a call to its register to retrieve the current of the photodiode from the Red and IR LED is made, and these values are typically within the range of 20 to 100, which should be in the units of mA.

Using this data, the example SpO2 can be calculated, assuming a value of 87 mA for the Red LED and 61 mA for the IR LED.

$$R = \frac{ln(I_R(\lambda_R))}{ln(I_{IR}(\lambda_{IR}))} = \frac{ln(87)}{ln(61)} = 1.086$$

$$\%SpO_2 = 110 - 25 \bullet R = 82.85\ \%$$

The partial derivatives can be derived as:

$$\frac{\delta R}{\delta I_R} = \frac{1}{I_R \cdot ln(I_{IR})}$$

$$\frac{\delta R}{\delta I_{IR}} = -\frac{ln(I_R)}{I_{IR} \cdot ln(I_R)}$$

The sensitivity can be estimated as follows, assuming the photodiode is more sensitive to the Red LED:

$$\frac{\delta R}{\delta I_R} = \frac{1}{I_R \cdot ln(I_{IR})} = \frac{1}{87 \cdot ln(61)} = 0.002796 \ \frac{1}{mA}$$

$$\Delta SpO_2 = 110 - 25 \cdot \frac{dR}{dI_R}$$

$$\Delta SpO_2 = \ -25 \cdot 0.002796 = \ -0.069902 \ \frac{SpO_2}{R}$$

Which can be estimated as a sensitivity of 0.1% SpO_2 per unit R.

The resolution from the 16 bit ADC can be estimated as:

$$Resolution \ = \frac{20 \ mA}{2^{16} \ bits} = 305.2 \ nA \ per \ bit$$

These values should indicate that the sensor is sensitive enough for our testing purposes.

It is important to note that our FiO_2 must be governed by the respiratory minute volume equation, which dictates the maximum and minimum amount of oxygen a person can and/or should safely inhale and exhale.

$$MV = TV \ x \ RR$$

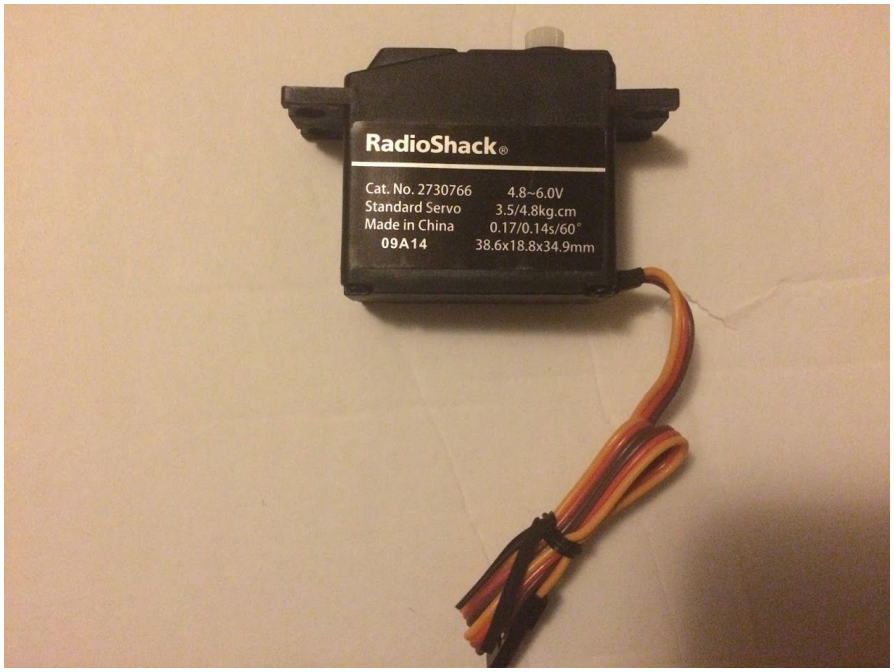
MV : *minute ventilation*
TV : *tidal volume*
RR : *respiratory rate*

An adult human needs about 6-8 L of oxygen per minute in order to properly maintain tissues and remove carbon dioxide (Akif, 2010). In contrast, a child needs about 4 - 5 L of oxygen per minute. An example of this rate for an adult can be seen below.

$$MV = 500 \frac{ml}{breath} \ x \ 12 \frac{breaths}{min} = 6000 \ \frac{ml}{min}$$

This applies to our project because for example, we would need to ensure that the FiO_2 does not fall below 6000 ml/min to ensure that a patient receives sufficient oxygen.

Next we needed to find a mechanism to actually rotate the mechanism. We chose to use a servo for proof-of-concept testing purposes, the Radio Shack standard servo.



This servo offers 3.54/4.8 kg·cm of torque, or 0.34 N·m torque.

We needed to calculate the amount of torque needed to turn the knob on the oxygen blender.

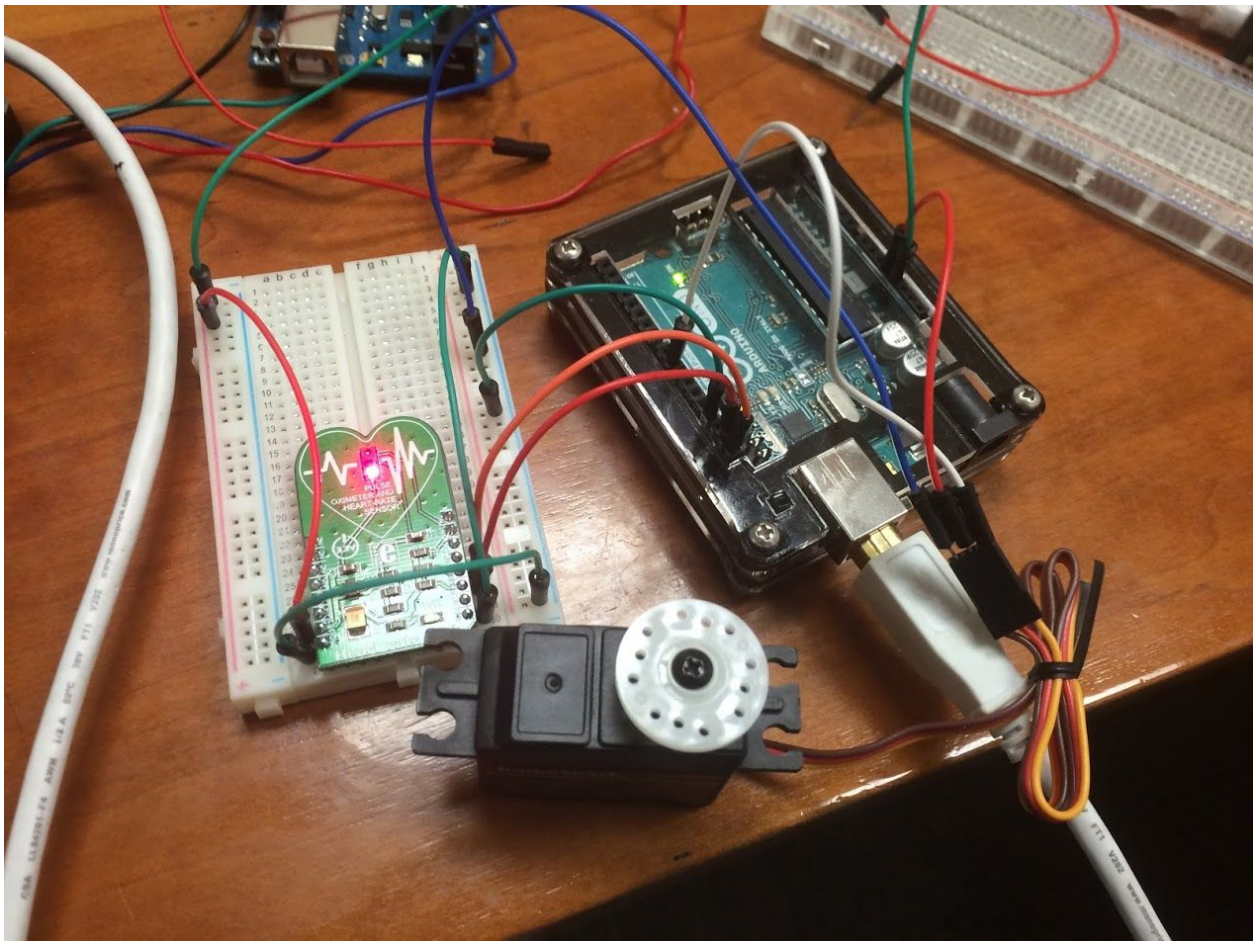
$$\tau = m \cdot g \cdot r$$

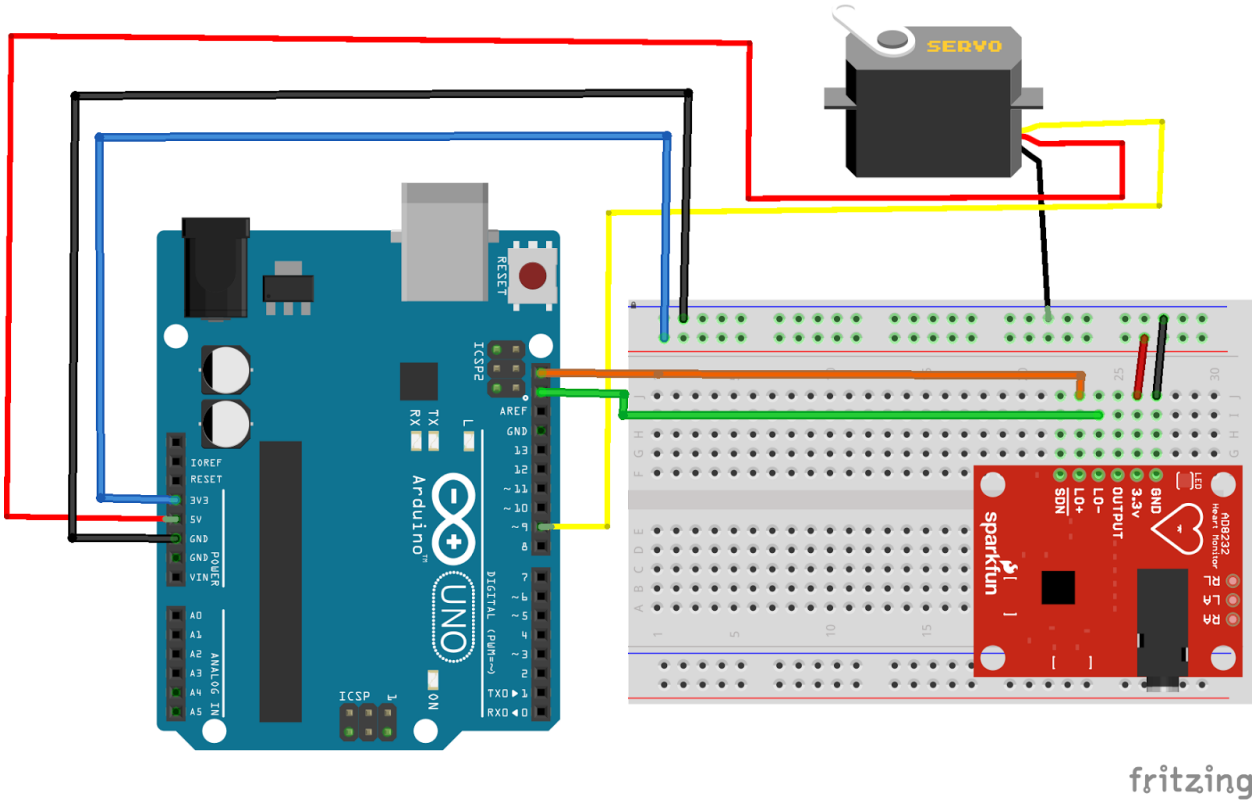
From our test, we needed a 0.136 kg weight at a distance of 0.15 m to turn the knob.

$$\tau = (0.136\ kg)(9.8\ \tfrac{m}{s^2})(0.15\ m) = 0.2\ N \cdot m$$

So, from this calculation, the servo provide enough torque to turn the knob.

Our example system can be seen below, with the servo turning in response to the SpO_2 level read by the microcontroller. This demonstrates that an external-rotation feedback control could potentially be used to control the oxygen blender's FiO_2. We believe this proof-of-concept is enough to continue working on the project. Following the setup is a fritzing diagram and some example output from the sensor. For the fritzing diagram, we could not find a schematic of our pulse oximeter so we used a heart rate sensor as a substitute for the diagram.






```
SpO2:95%
SpO2:95%
SpO2:94%
SpO2:94%
SpO2:94%
Mean:   37
Low SpO2
SpO2:94%
SpO2:94%
SpO2:94%
SpO2:96%
SpO2:96%
SpO2:96%
SpO2:98%
SpO2:98%
SpO2:95%
SpO2:95%
SpO2:95%
Mean:   95
Good SpO2
```

The main code we used and made can be seen below which uses some arduino libraries from Oxullo.

```
/*
 * SpO2 Sketch 3
 * Automated Oxygen Blender device
 */
#include <Wire.h>
#include <Servo.h>
#include "MAX30100_PulseOximeter.h"

int reportTime = 60000;
int spO2_array[10];
int index = 0;
const int stepsPerRevolution = 400;
int average_reading = 0;
int target_spO2 = 93;
int low_spO2 = 88;
int fiO2 = 100;
int low_fiO2 = 50;
int servoPin = 9;
int warning_cleared = 0;

//buttons to interface with person
const int button1 = 2;
const int button2 = 3;

Servo myservo;  // create servo object to control a servo
int pos = 0;    // variable to store the servo positio

///Create object
PulseOximeter pulseOx;

uint32_t tsLastReport = 0;

//10 minute idle time
int idleTime = 600000;

void setup()
{
  Serial.begin(115200);
  myservo.attach(servoPin);
  fiO2 = 100;
  //initialize servo to fully rotate to test
  myservo.write(180);
  pinMode (button1, INPUT);
 //start pulse oximetry
  pulseOx.begin();
  pulseOx.setOnBeatDetectedCallback(onBeatDetected);
}

/*
 * general loop to continuously run the program
```

```
 */
void loop()
{
  //update readings
  pulseOx.update();
  //update every minute
  if (millis() - tsLastReport > reportTime) {
    Serial.print(pulseOx.getHeartRate());
    Serial.print("SpO2: ");
    Serial.println(pulseOx.getSpO2());
    Serial.println("%");

    tsLastReport = millis();

    //store values into an array for averaging
    spO2_array[index] = pulseOx.getSpO2();

    //average spO2 over a 30 minute period
    if (index > 29)
    {
      Serial.print("Average:\t");
      average_reading = (average(spO2_array, 30));
      Serial.println(average_reading);
      index = -1;

      if (average_reading < target_spO2)
        lowReading();
      else
        highReading();
    }
    index++;
  }
}

/*
   Compute Average
*/
int average(int spO2_array[], int array_size)
{
  int index = 0, sum = 0;
  for (index = 0; index < array_size; index++)
  {
    sum += spO2_array[index];
  }
  sum = sum / array_size;
  return sum;
}

void lowReading()
{
```

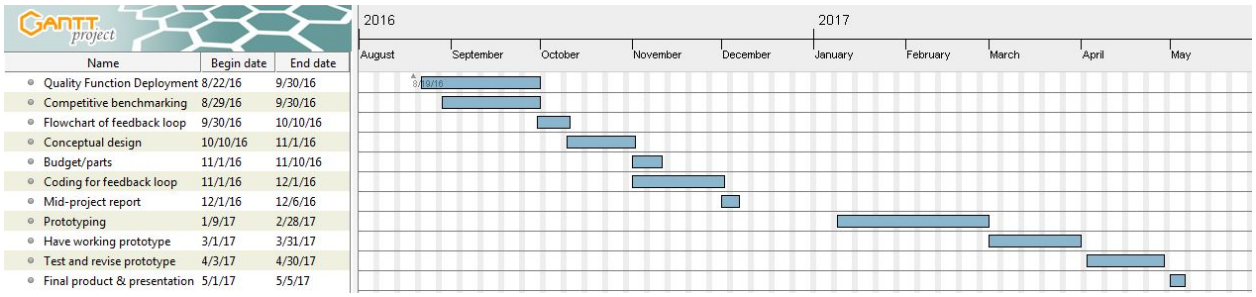
```
  Serial.println("Low SpO2");
  increase_fiO2();
}

void increase_fiO2()
{
  fiO2 = fiO2 + 10;
  moveServo(fiO2);
}

void decrease_fiO2()
{
  int option = 0;
  fiO2 = fiO2 - 10;

  if(fiO2 < 0) fiO2 = 0;

  //fiO2 will be set below 50%
  //Needs confirmation to continue
  if ((fiO2 < low_fiO2) && (warning_cleared == 0))
  {
    option = warning();

    //safety mechanism, increase FiO2 if no response
    //after a set amount of time
    if(option == 0)
    {
      increase_fiO2();
    }
    //continue to decrease fiO2
    else if (option == 1)
    {
      moveServo(fiO2);
    }
    //user chose to reverse fio2
    else if (option == 2)
    {
      increase_fiO2();
    }
  }
  else if ((fiO2 < low_fiO2) && (warning_cleared == 1))
  {
    moveServo(fiO2);
  }


}

void highReading()
{
```

```
  Serial.println("Good SpO2, decreasing fiO2");
  decrease_fiO2();
}

/*
 * option 0: retain fio2
 * option 1: continue to decrease fio2
 * option 2: reverse and incresae fio2
 * option 3: idle, no response, reset fio2 to high
 */
int warning()
{
  int continueLoop = 1, option = 0;
  uint32_t idleReport = millis();
  while (continueLoop == 1)
  {
    Serial.println("Warning, low FiO2");

    if(millis() - idleReport > idleTime)
    {
      idleReport = millis();
      continueLoop = 0;
      option = 3;
    }
    //warning cleared, continue to decrease
    if (digitalRead(button1) == HIGH)
    {
      continueLoop = 0;
      option = 1;
      warning_cleared = 1;
    }

    if(digitalRead(button2) == HIGH)
    {
      continueLoop = 0;
      option = 2;
    }
  }

  return option;
}

void moveServo(int fiO2)
{
  pos = fiO2/100*180;
  myservo.write(pos);
}
```

Creativity and Innovation

This design is creative because we allow the prototype to be attached to existing air-oxygen blenders. The products and ideas that are out now are designed to be devices that must be purchased as a standalone device with proprietary hardware. This can lead to innovation by creating products that attach to existing devices rather than creating proprietary devices which would decrease cost and complexity of the device.

Prototype Planning

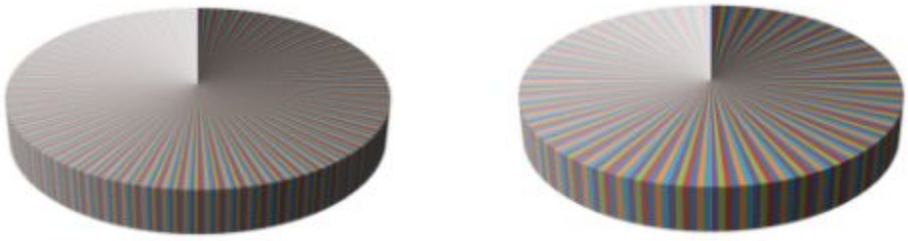


Name	Begin date	End date
Quality Function Deployment	8/22/16	9/30/16
Competitive benchmarking	8/29/16	9/30/16
Flowchart of feedback loop	9/30/16	10/10/16
Conceptual design	10/10/16	11/1/16
Budget/parts	11/1/16	11/10/16
Coding for feedback loop	11/1/16	12/1/16
Mid-project report	12/1/16	12/6/16
Prototyping	1/9/17	2/28/17
Have working prototype	3/1/17	3/31/17
Test and revise prototype	4/3/17	4/30/17
Final product & presentation	5/1/17	5/5/17

In the future, we plan to 3D print an assembly to house the servo and the arduino while attaching to the air-oxygen blender. We also plan on using the feedback code developed in order to control the servo. We will, of course, test and revise as necessary to create a functional prototype.

For the future, we would like to replace the pulse-oximeter with a standard hospital-grade pulse oximeter, and replace the servo with a stepper motor and an encoder. We may also replace the Arduino with a Raspberry pi to offer better internet-of-things connectivity for remote monitoring and decision making using an application.

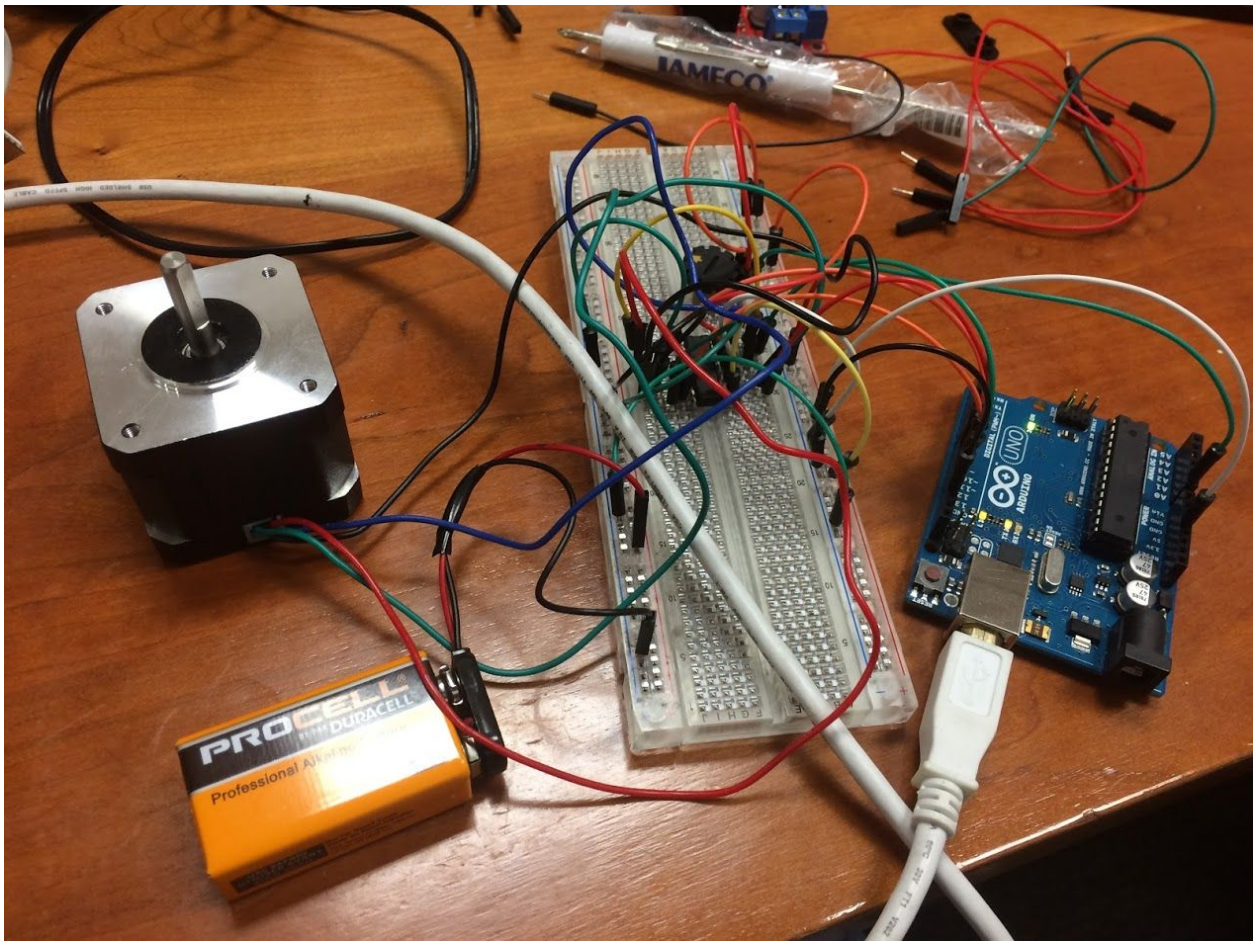
The stepper motor can be seen in the following image. It is the 0.9 degree Nema 17 Stepper Motor Bipolar. The 0.9 degree 400 steps/revolution, compared to 200 steps/revolution of a 1.8 degree motor. This stepper motor was picked over a 1.8 degree stepper motor, because this stepper motor would have more increments in revolutions, to be able to turn the knob as accurate as can be. The trade off is that while a 0.9 degree motor offers higher turning precision, it will have less torque compared to a 1.8 degree motor. Since the application I need the motor for is to turn a small knob, this motor should provide enough torque to turn it. It is listed as having 0.36 N•m, which is enough to turn the knob.



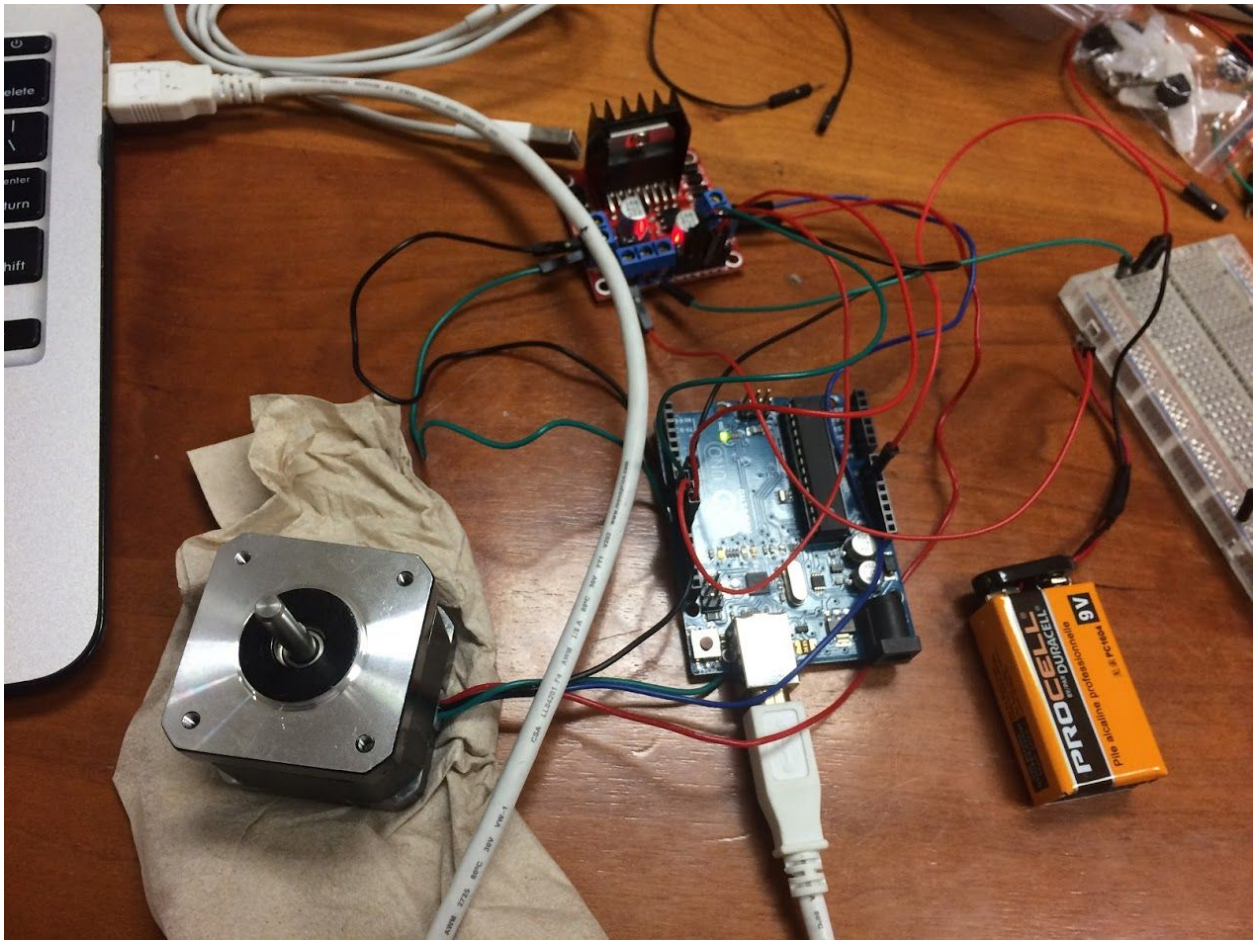
0.9 Degree Step Angle

1.8 Degree Step Angle

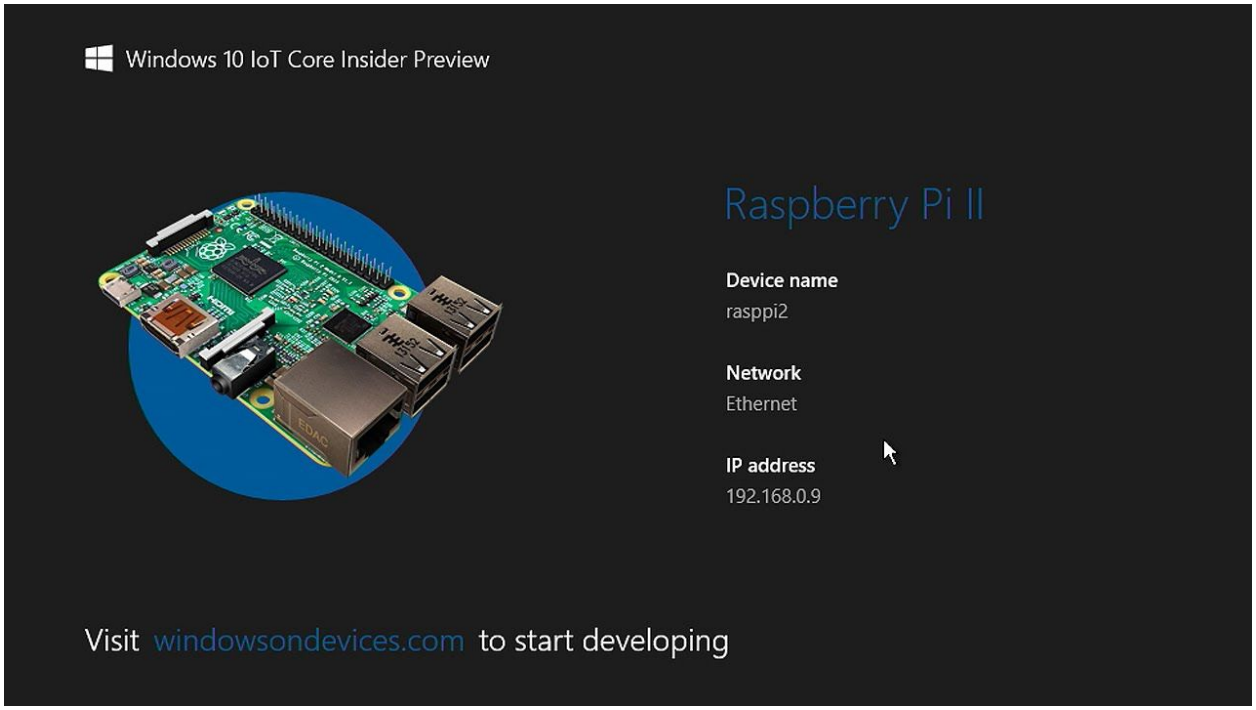
The stepper motor would require a motor driver to power it from a micro controller. Initially we thought about using the Texas Instruments SN754410, but it would complicate the wiring process, as seen below.



So instead we chose the Qunqi L298N, which states it can drive one bipolar stepper motor at 2A which is enough for our purposes.


PROCELL
Pile alcaline professionnelle
9V

The raspberry pi with Windows Internet of Things could prove useful in our device.



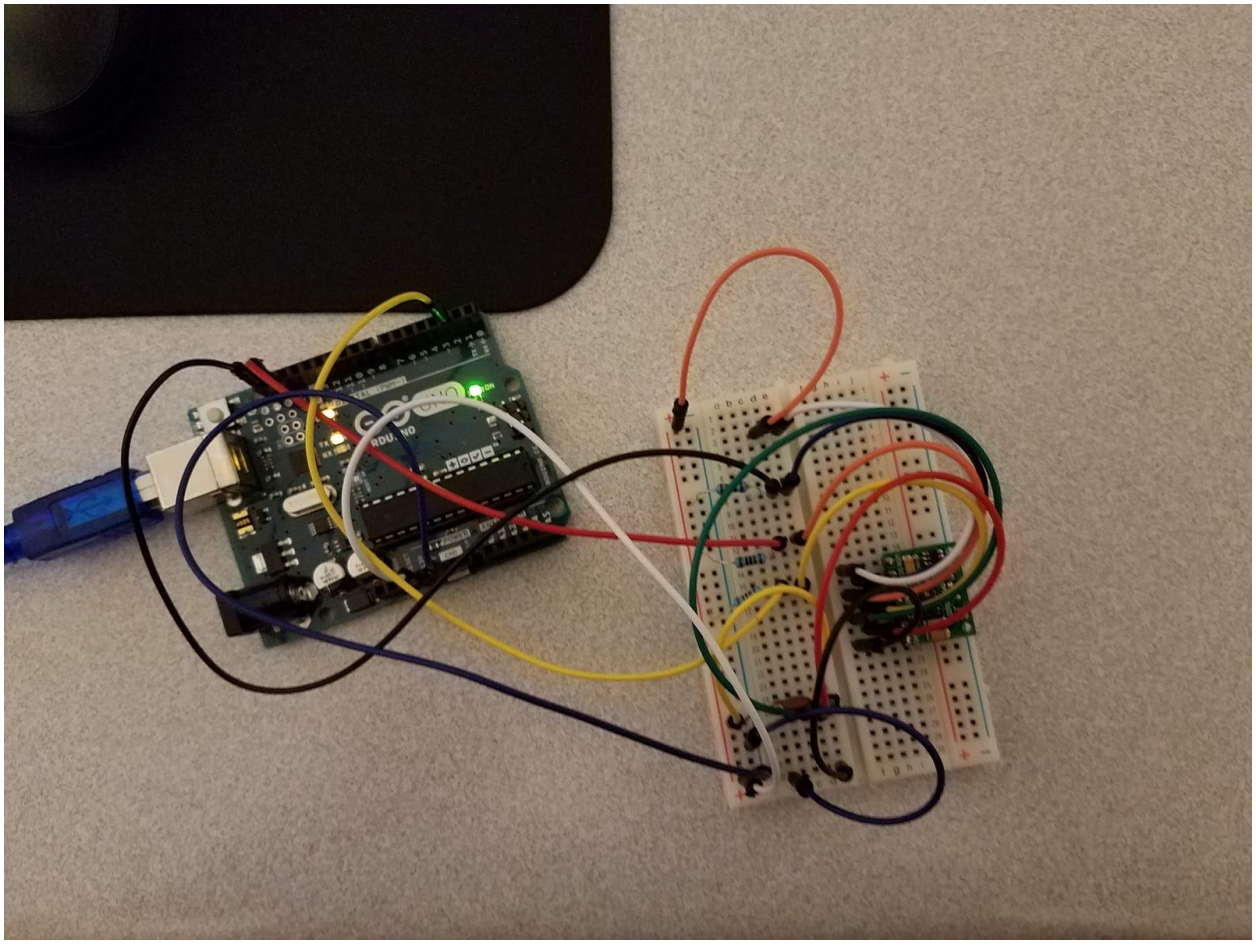
We will also need a rotary encoder, such as the Yumo E6A2.

Since the device will rely on the stepper motor, it will be crucial to ensure that the rotation mechanism is accurate and corresponds to a set FiO_2. To do this, we will need to implement a closed-loop feedback control mechanism, using a PID controller and possibly kalman filtering.

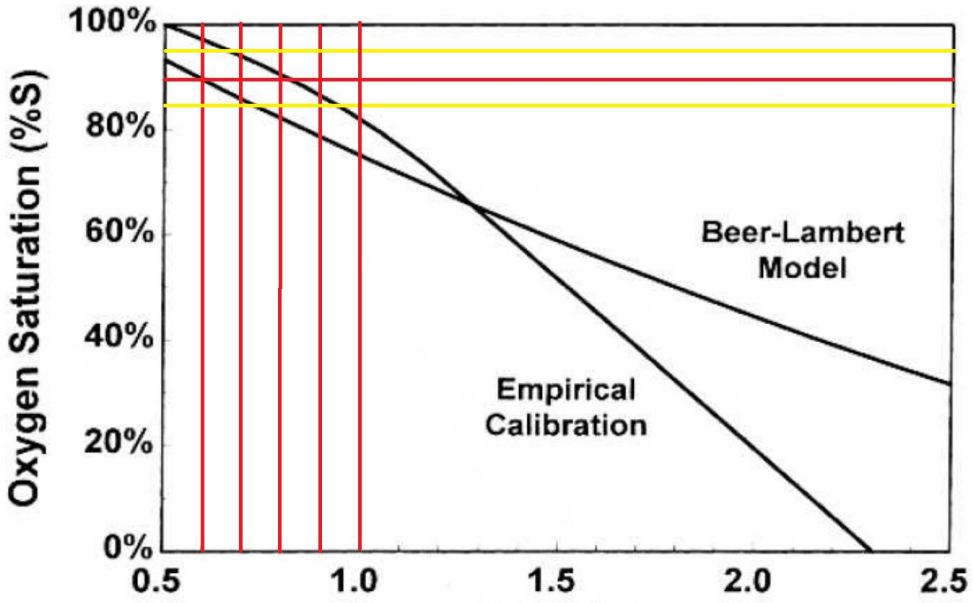
Group Member Contributions

Luis Perea: I bought all of the components out of pocket, I tested all of the components, I created the midterm report and the midterm presentation, and I made the final report and presentation along with the calculations, equations, and demonstration.

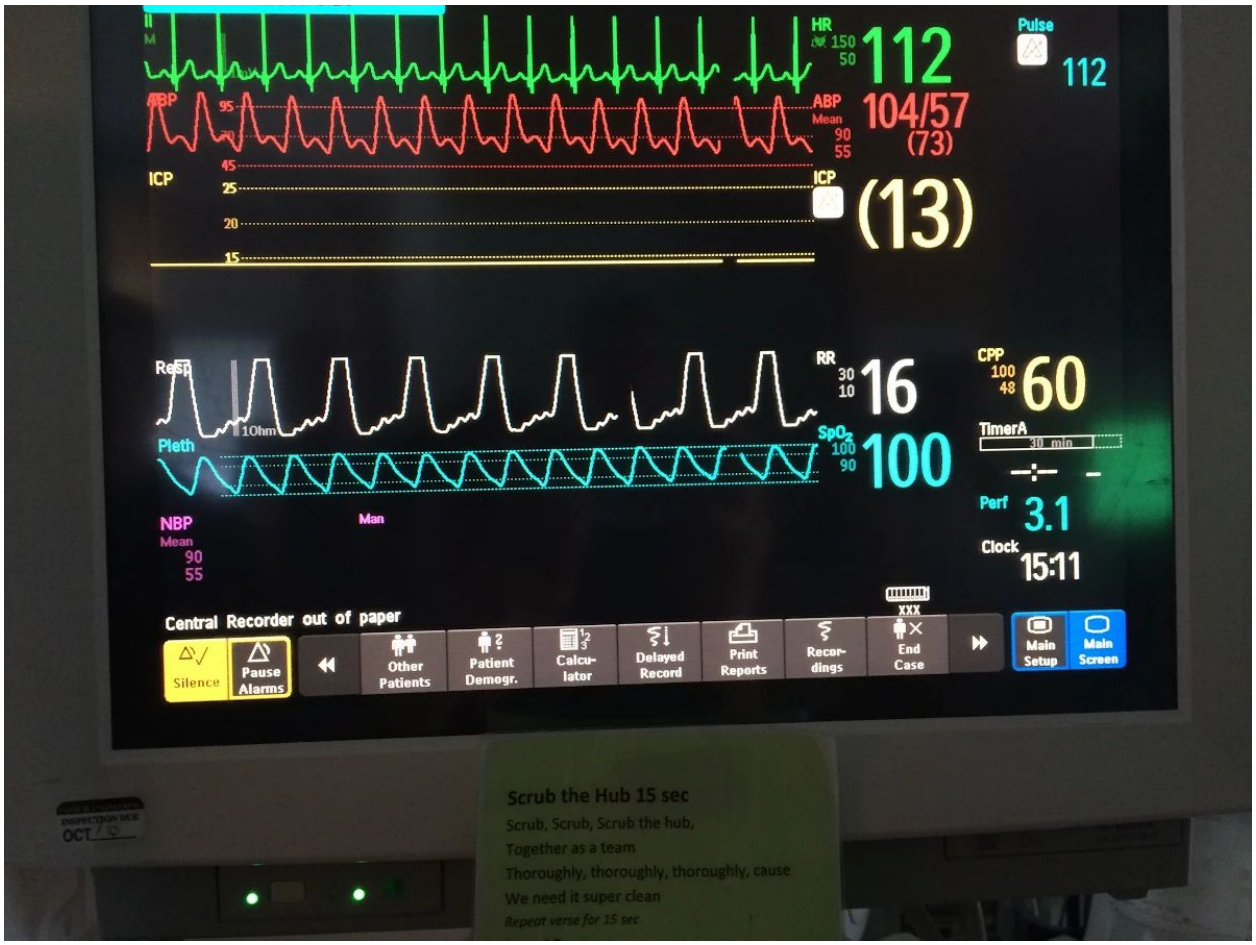
Justin Joseph: I had created the initial flow chart for the feedback loop, which was then revised and expanded upon. I also programmed the logic based on that for the Arduino using the MAX30100 sensor. To test the code and the MAX30100 pulse oximeter, I had used the wiring diagram to create the breadboard circuits. I, along with Luis, have worked on the midterm and final reports and presentations. I also helped with integration of the separate code with the main program, and I programmed the user interface displayed using the raspberry pi.



Graph of SpO2 vs Current and Wavelength Ratio Between Infrared and Red LED from an example paper of how to use pulse oximetry



Example of SpO_2 used in real-world settings



Oxullo MAX30100 SpO_2 arduino library

```
Arduino-MAX30100 oximetry / heart rate integrated sensor library
Copyright (C) 2016  OXullo Intersecans <x@brainrapers.org>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
```

```
*/

#include <math.h>

#include "MAX30100_SpO2Calculator.h"

// SaO2 Look-up Table
// http://www.ti.com/lit/an/slaa274b/slaa274b.pdf
const uint8_t SpO2Calculator::spO2LUT[43] = {100,100,100,100,99,99,99,99,99,99,98,98,98,98,
                                             98,97,97,97,97,97,97,96,96,96,96,96,96,95,95,
                                             95,95,95,95,94,94,94,94,94,93,93,93,93,93};

SpO2Calculator::SpO2Calculator() :
    irACValueSqSum(0),
    redACValueSqSum(0),
    beatsDetectedNum(0),
    samplesRecorded(0),
    spO2(0)
{
}

void SpO2Calculator::update(float irACValue, float redACValue, bool beatDetected)
{
    irACValueSqSum += irACValue * irACValue;
    redACValueSqSum += redACValue * redACValue;
    ++samplesRecorded;

    if (beatDetected) {
        ++beatsDetectedNum;
        if (beatsDetectedNum == CALCULATE_EVERY_N_BEATS) {
            float acSqRatio = 100.0 * log(redACValueSqSum/samplesRecorded) /
log(irACValueSqSum/samplesRecorded);
            uint8_t index = 0;

            if (acSqRatio > 66) {
                index = (uint8_t)acSqRatio - 66;
            } else if (acSqRatio > 50) {
                index = (uint8_t)acSqRatio - 50;
            }
            reset();

            spO2 = spO2LUT[index];
        }
    }
}

void SpO2Calculator::reset()
{
    samplesRecorded = 0;
    redACValueSqSum = 0;
```

```
    irACValueSqSum = 0;
    beatsDetectedNum = 0;
    spO2 = 0;
}

uint8_t SpO2Calculator::getSpO2()
{
    return spO2;
}
```

Pulse Oximeter Library

```
/*
Arduino-MAX30100 oximetry / heart rate integrated sensor library
Copyright (C) 2016  OXullo Intersecans <x@brainrapers.org>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include <Arduino.h>

#include "MAX30100_PulseOximeter.h"


PulseOximeter::PulseOximeter() :
    state(PULSEOXIMETER_STATE_INIT),
    tsFirstBeatDetected(0),
    tsLastBeatDetected(0),
    tsLastSample(0),
    tsLastBiasCheck(0),
    tsLastCurrentAdjustment(0),
    tsLastTemperaturePoll(0),
    redLedPower((uint8_t)RED_LED_CURRENT_START),
    temperature(0),
    onBeatDetected(NULL)
{
}

void PulseOximeter::begin(PulseOximeterDebuggingMode debuggingMode_)
{
    debuggingMode = debuggingMode_;

    hrm.begin();
    hrm.setMode(MAX30100_MODE_SPO2_HR);
    hrm.setLedsCurrent(IR_LED_CURRENT, RED_LED_CURRENT_START);

    irDCRemover = DCRemover(DC_REMOVER_ALPHA);
```

```
    redDCRemover = DCRemover(DC_REMOVER_ALPHA);

    state = PULSEOXIMETER_STATE_IDLE;

    // Start temperature sampling and wait for its completion (blocking)
    hrm.startTemperatureSampling();
    while (!hrm.isTemperatureReady());
    temperature = hrm.retrieveTemperature();
}

void PulseOximeter::update()
{
    checkSample();
    checkCurrentBias();
    checkTemperature();
}

float PulseOximeter::getHeartRate()
{
    return beatDetector.getRate();
}

uint8_t PulseOximeter::getSpO2()
{
    return spO2calculator.getSpO2();
}

uint8_t PulseOximeter::getRedLedCurrentBias()
{
    return redLedPower;
}

float PulseOximeter::getTemperature()
{
    return temperature;
}

void PulseOximeter::setOnBeatDetectedCallback(void (*cb)())
{
    onBeatDetected = cb;
}

void PulseOximeter::checkSample()
{
    if (millis() - tsLastSample > 1.0 / SAMPLING_FREQUENCY * 1000.0) {
        tsLastSample = millis();
        hrm.update();
        float irACValue = irDCRemover.step(hrm.rawIRValue);
        float redACValue = redDCRemover.step(hrm.rawRedValue);
```

```
        // The signal fed to the beat detector is mirrored since the cleanest monotonic spike
is below zero
        float filteredPulseValue = lpf.step(-irACValue);
        bool beatDetected = beatDetector.addSample(filteredPulseValue);

        if (beatDetector.getRate() > 0) {
            state = PULSEOXIMETER_STATE_DETECTING;
            spO2calculator.update(irACValue, redACValue, beatDetected);
        } else if (state == PULSEOXIMETER_STATE_DETECTING) {
            state = PULSEOXIMETER_STATE_IDLE;
            spO2calculator.reset();
        }

        switch (debuggingMode) {
            case PULSEOXIMETER_DEBUGGINGMODE_RAW_VALUES:
                Serial.print("R:");
                Serial.print(hrm.rawIRValue);
                Serial.print(",");
                Serial.println(hrm.rawRedValue);
                break;

            case PULSEOXIMETER_DEBUGGINGMODE_AC_VALUES:
                Serial.print("R:");
                Serial.print(irACValue);
                Serial.print(",");
                Serial.println(redACValue);
                break;

            case PULSEOXIMETER_DEBUGGINGMODE_PULSEDETECT:
                Serial.print("R:");
                Serial.print(filteredPulseValue);
                Serial.print(",");
                Serial.println(beatDetector.getCurrentThreshold());
                break;

            default:
                break;
        }

        if (beatDetected && onBeatDetected) {
            onBeatDetected();
        }
    }
}

void PulseOximeter::checkCurrentBias()
{
    // Follower that adjusts the red led current in order to have comparable DC baselines
between
    // red and IR leds. The numbers are really magic: the less possible to avoid oscillations
```

```
    if (millis() - tsLastBiasCheck > CURRENT_ADJUSTMENT_PERIOD_MS) {
        bool changed = false;
        if (irDCRemover.getDCW() - redDCRemover.getDCW() > 70000 && redLedPower <
MAX30100_LED_CURR_50MA) {
            ++redLedPower;
            changed = true;
        } else if (redDCRemover.getDCW() - irDCRemover.getDCW() > 70000 && redLedPower > 0) {
            --redLedPower;
            changed = true;
        }

        if (changed) {
            hrm.setLedsCurrent(IR_LED_CURRENT, (LEDCurrent)redLedPower);
            tsLastCurrentAdjustment = millis();
        }

        tsLastBiasCheck = millis();
    }
}

void PulseOximeter::checkTemperature()
{
    if (millis() - tsLastTemperaturePoll > TEMPERATURE_SAMPLING_PERIOD_MS) {
        if (hrm.isTemperatureReady()) {
            temperature = hrm.retrieveTemperature();
        }
        hrm.startTemperatureSampling();

        tsLastTemperaturePoll = millis();
    }
}
```

Processing MAX30100 Library

```
/*
Arduino-MAX30100 oximetry / heart rate integrated sensor library
Copyright (C) 2016  OXullo Intersecans <x@brainrapers.org>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
```

```
along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

// Grapher helper for the Arduino MAX30100 library

import processing.serial.*;

// NOTE: when using PULSEOXIMETER_DEBUGGINGMODE_RAW_VALUES
// set this to -1 to enable the auto range mode
final int ABSMAX = 800;
// Adjust to the serial port. Under OSX, UNO platforms and alike are auto-detected.
final String serialPort = "/dev/tty.usbmodemFD13131";


final int WIDTH = 1200;
final int HEIGHT = 600;
final int CHANNELS = 2;
final color[] colors = {color(0, 0, 0), color(255, 0, 0), color(0, 255, 0), color(0, 0, 255)};


float[][] series = new float[CHANNELS][WIDTH];
float heartRate = 0;
int spO2 = 0;
boolean beatDetected = false;
int ptr = 0;

Serial myPort;

void settings()
{
  size(WIDTH, HEIGHT);
}

void setup ()
{
  String attemptPort = serialPort;

  for (int i=0 ; i < Serial.list().length ; ++i) {
    String port = Serial.list()[i];
    if (port.matches(".+tty\\.usbmodem.+")) {
      attemptPort = port;
      break;
    }
  }

  println("Opening port " + attemptPort);

  myPort = new Serial(this, attemptPort, 115200);

  stroke(0);
```

```
  fill(0);
  textSize(8);
}

void draw ()
{
  if (beatDetected) {
    background(255, 200, 200);
    beatDetected = false;
  } else {
    background(255);
  }

  stroke(30);

  line(0, height/2, width, height/2);

  float maxv=0, minv;
  for (int s=0 ; s < CHANNELS ; ++s) {
    float[] samples = series[s];
    maxv = max(maxv, abs(max(samples)), abs(min(samples)));
    if (ABSMAX != -1) {
      maxv = min(maxv, ABSMAX);
    }
  }

  // Avoids map() errors
  if (maxv == 0) {
    maxv = 1;
  }
  minv = -maxv;

  for (int s=0 ; s < CHANNELS ; ++s) {
    stroke(colors[s]);

    float[] samples = series[s];
    float seriesMax = max(samples);

    text("ch " + s + " cur:" + samples[ptr] + " max:" + seriesMax + " min:" + min(samples), 0,
8 + 10 * s);

    boolean maxDisplayed = false;
    for (int i = 0 ; i < WIDTH ; ++i) {
      if (i > 0) {
        float ipy = HEIGHT - map(samples[i-1], minv, maxv, 0, HEIGHT);
        float iy = HEIGHT - map(samples[i], minv, maxv, 0, HEIGHT);

        if (abs(samples[i] - seriesMax) < 0.001 && !maxDisplayed) {
          text("v=" + samples[i], i, iy);
          maxDisplayed = true;
```

```
        }

        line(i - 1, ipy, i, iy);
      }
    }
  }

  text("Rate: " + heartRate, 200, 8);
  text("SpO2: " + spO2 + "%", 200, 18);
}

void serialEvent (Serial myPort)
{
  String sLine = myPort.readStringUntil('\n');

  if (sLine == null) {
    return;
  }

  if (sLine.substring(0, 2).equals("R:")) {
    String[] sValues = split(sLine.substring(2), ',');

    for (int i=0 ; i < sValues.length ; ++i) {
      float sample = float(sValues[i]);

      if (Float.isNaN(sample)) {
        continue;
      }

      series[i][ptr] = sample;
    }

    ptr = (ptr + 1) % WIDTH;
  } else if (sLine.substring(0, 2).equals("H:")) {
    heartRate = float(sLine.substring(2));
  } else if (sLine.substring(0, 2).equals("B:")) {
    beatDetected = true;
  } else if (sLine.substring(0, 2).equals("C:")) {
    println(sLine);
  } else if (sLine.substring(0, 2).equals("O:")) {
    spO2 = int(float(sLine.substring(2)));
  }
}
```

Semester 2

Automatic HIFLOW Oxygen Weaning Device

Spring 2017

Semester 2

CSEE 4911, Spring 2017

Computer Systems Engineering

College of Engineering

The University of Georgia

Executive Summary

Oxygen flow through a HIFLOW nasal cannula system is a noninvasive method that provides oxygen to a patient through a nasal tube. Compared to endotracheal systems, there is reduced risk of patient discomfort in a HIFLOW oxygen system. However, current HIFLOW oxygen systems require manual flow rate calibration by a bedside nurse technician, and overexposure to oxygen can result in fatal consequences such as hypoxia. An automated oxygen system that is capable of reading a patient's oxygen levels and reduce the oxygen flow to rate through a negative feedback system until the patient oxygen levels are stable is needed because it reduces the risk of miscalculated oxygen concentration and relieves the nurse to pursue other duties.

There is currently no device in the United States that is capable of reading these oxygen sensors, having the ability to adjust oxygen flow rate, and alert a physician when oxygen levels are critical. We will design a device capable of these functions and be compatible with current HIFLOW systems, as per the design requirements of the client. So far, we have implemented a breadboard circuit and basic logic. The current circuit includes a pulse oximeter and servo motor.

The goals of this project are to develop a new automated oxygen hiflow device. It is to use a feedback mechanism of a patient's SPO2 (blood oxygen level) data to adjust the FIO2 (oxygen flow) rate. An oxygen blender is the device that uses a rotate-style knob to adjust oxygen flow. This project should be able to either completely replace the oxygen blender or serve as a mediator to turn the knob.

The result of this project is a simple feedback mechanism between a pulse oximeter, a raspberry pi, and a stepper motor. The raspberry pi reads in SPO2 data from the pulse oximeter to rotate the stepper motor left or right, which should be attached to the oxygen blender to adjust the oxygen flow rate.

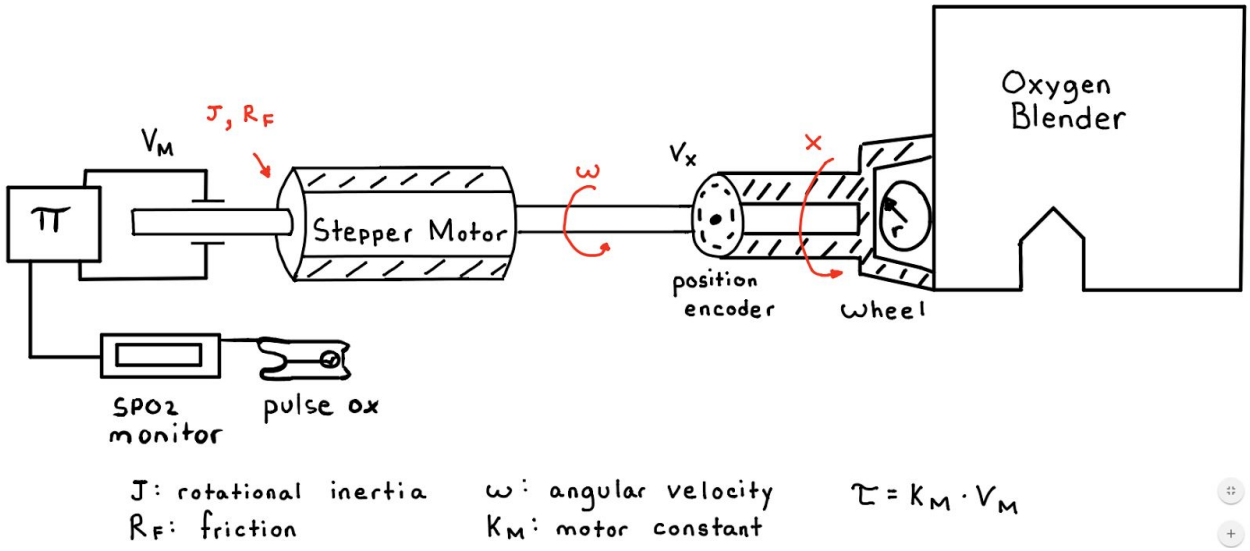
While the design is simple, a well-developed system would have the application of modernizing oxygen hiflow systems. It would allow a hospital to become more connected with the patients by monitoring data remotely, freeing up time of the nurses. This would also reduce the risk of hypoxia and hyperoxia by allowing more fine-tuned control over the FIO2 based on often-monitored SPO2 data.

Evaluation

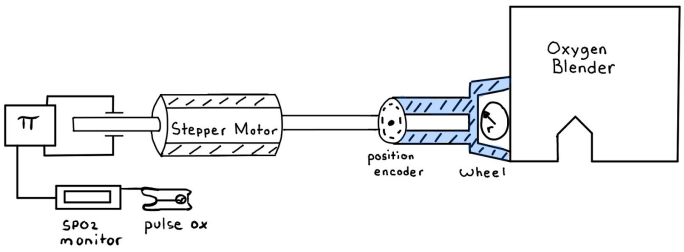
We have updated the prototype design from last semester to include the updated components. Specifically it now includes the raspberry pi, motor driver, pulse oximeter and SpO2 monitor.

In order to create a self-adjustment function on the oxygen blender, our team decided to attach the knob of the blender to a motor that could be controlled by a computer program. Between a stepper motor and servo motor, we determined that the stepper motor is the best option due to its wide rotational range and cost efficiency. The stepper motor contains a higher pole count than the servo motor, giving it a benefit at lower speeds. Since the increments on the blender knob are small, a motor moving at lower speeds is more favorable. The stepper motor is also more profitable because its high pole count does not require an encoder, while servo motors do.

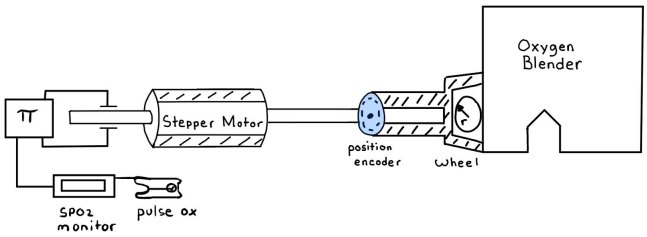
Our initial prototype was also coded to work with the Arduino microcontroller. However, upon obtaining a Masimo pulse oximeter for testing, we determined that the pulse ox is more compatible with Raspberry Pi, which also includes a screen interface, allowing ease of use for healthcare professionals.

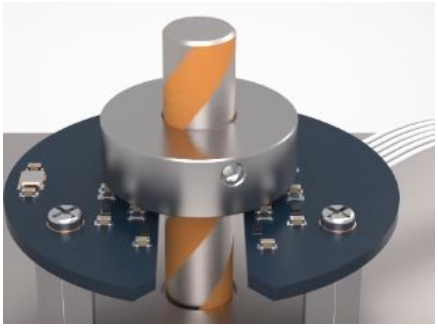


I will restate the updated components and their purpose in the design below.

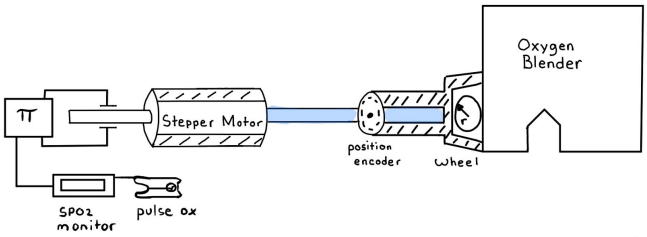


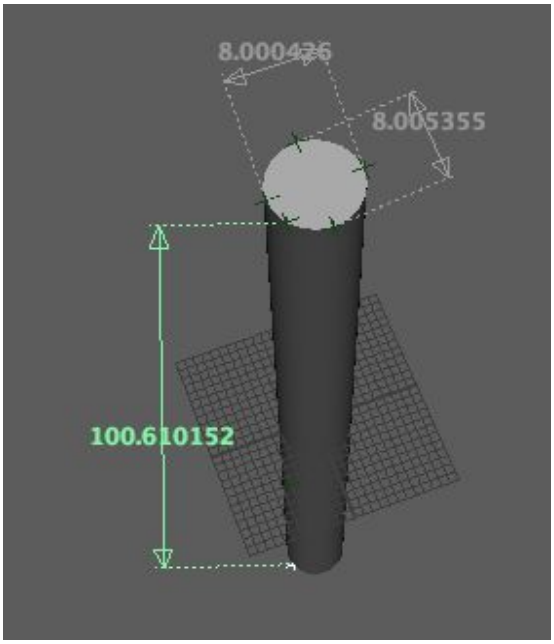
This piece is the 3D printed knob that will be used to rotate the FIO2 knob on the oxygen blender. It was designed in autocad to fit the rod attachment. The necessary torque to turn the knob is calculated as: $(0.136\ kg)(9.8\ \frac{m}{s^2})(0.15\ m) = 0.2\ N \cdot m$



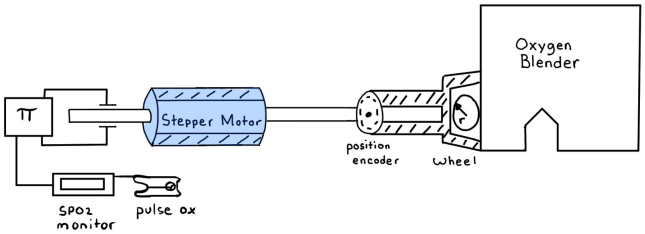


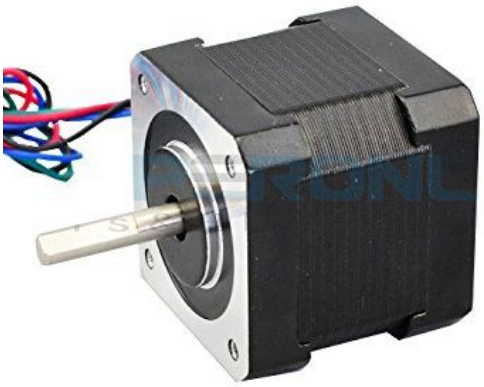
The RLS position encoder has 14-bit resolution and 0.2° sensitivity.



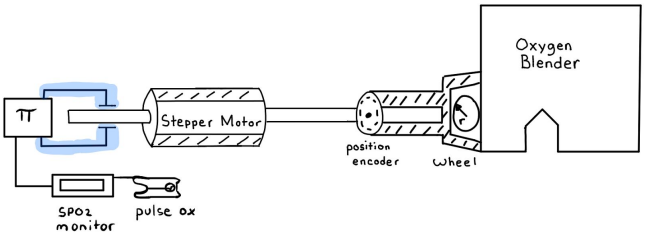


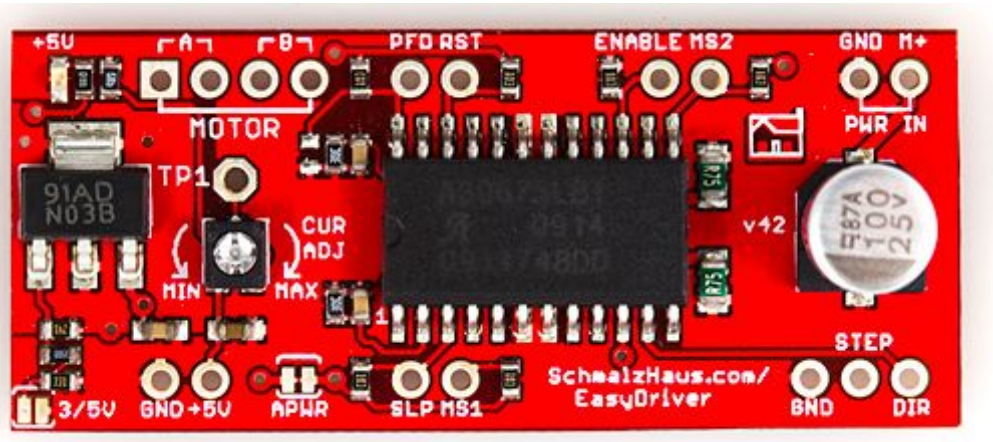
The 3D printed rod was made to attach to the stepper motor shaft. It was designed in Maya and is 8mm in diameter.



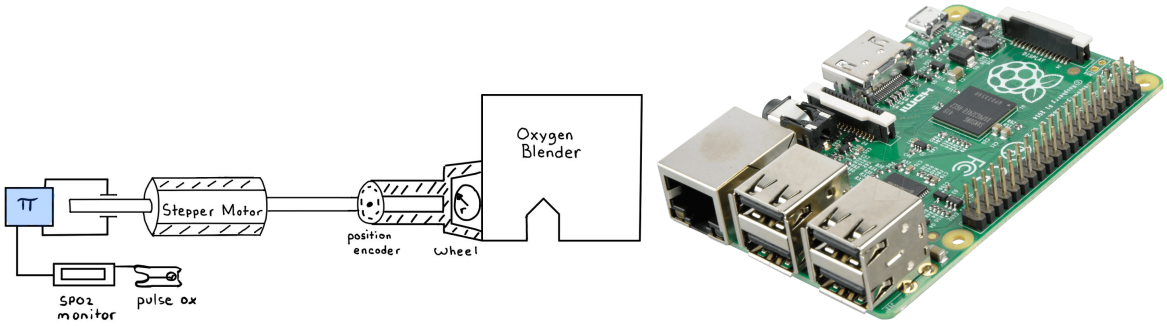


The Nema 17 stepper motor is a bipolar stepper motor with 0.9° rotation per step and provides 3.5kg•cm torque.

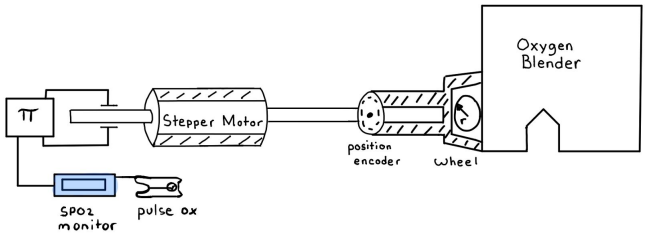




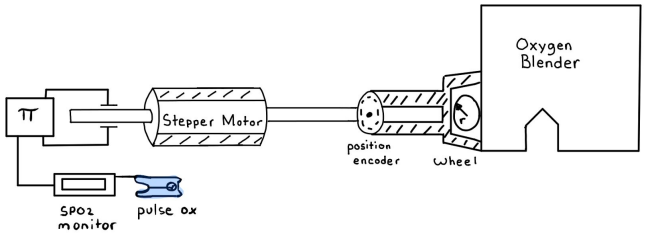
The stepper motor driver should be able to interface between the raspberry pi and the stepper motor. It should apply up to 700 mA/phase which should be enough to power the stepper motor to provide enough torque to rotate the knob.



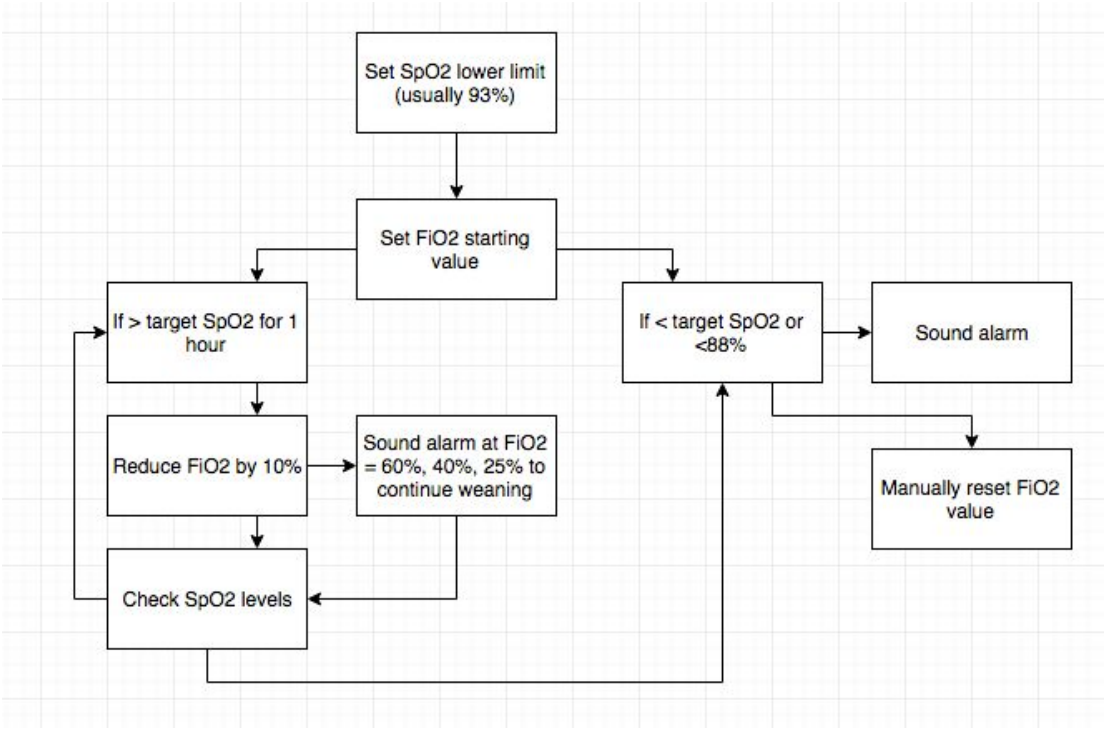
The raspberry pi will serve as a microcontroller and an IoT communications device. The “microcontroller” purpose is to calculate the PID motor control and read in the pulse oximeter data, while the IoT purpose is to monitor and upload the pulse oximeter data to a database.



The Masimo Radical 7 SpO2 monitor displays the data from the pulse oximeter and is able to convert it to the RS232 data format for serial communication.

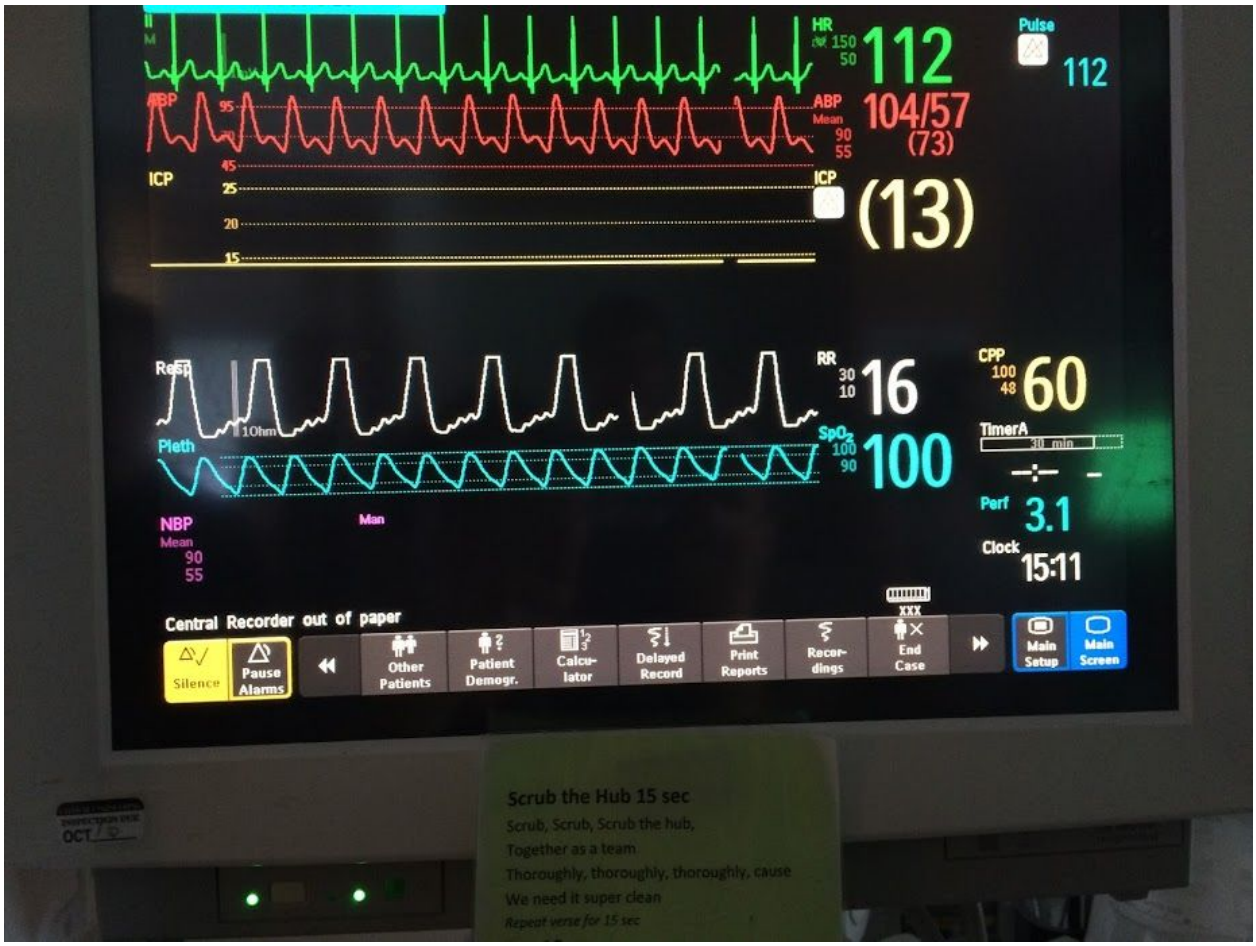


The Masimo Rainbow Set pulse oximeter features 16-bit resolution, “1% sensitivity”, in that it can detect a change of 1% SpO2 levels, and +/- 3% accuracy.

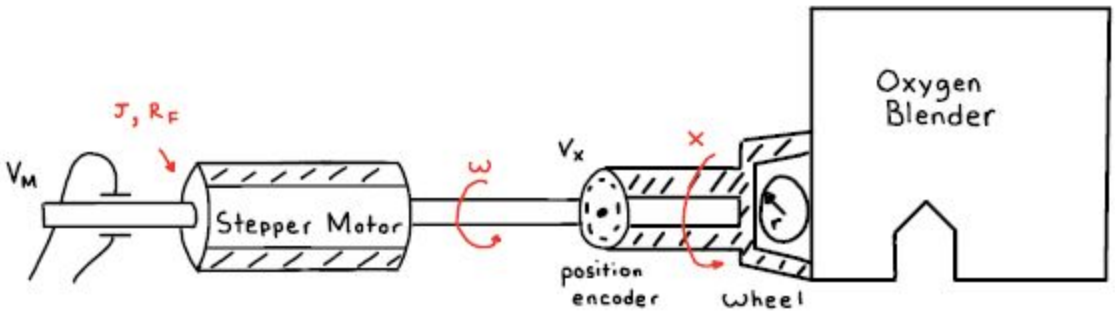


Through this prototype we were able to create a design that was able to meet the specifications provided by our client. The program was created to follow the logic of the flowchart as seen above. Specifically it uses a feedback loop from SPO2 to adjust FIO2. There remains much to be done. We were not able to accommodate the absolute encoder in time. We would have liked to introduce PID control into the system. The PID control with the encoder would have allowed for more accurate tunings of the oxygen blender, as well as guaranteeing the position of the oxygen blender knob. This would lead to a much more developed system, even more accurate to what the client wanted, as well as a much safer system overall. Additionally we would have liked to develop the database further and created a mobile phone application to allow for remote monitoring on websites, phones, and intranets of the hospital.

We would also have liked to be able to integrate the device into a Phillips VUElink as used by the hospital, as it serves as the hub for patient data input and output.



The figures below show estimated calculations for the PID control of the stepper motor.



J: rotational inertia
R_F: friction
ω: angular velocity
K_M: motor constant

(1) $J\dot{\omega} + R_F\omega = K_M V_M$ $\qquad \tau = K_M \cdot V_M$

$J \cdot s\,\omega(s) + R_F\,\omega(s) = K_M V_M(s)$

$\omega(s)[Js + R_F] = K_M V_M(s)$

$$\omega(s) = \frac{K_M}{Js + R_F} \cdot V_M(s)$$

each rotation of $\phi = 2\pi$ advances the wheel encoder by $C = 2\pi r$

$$X(s) = \frac{C}{2\pi}\,\phi(s) \qquad \omega(t) = \dot{\phi}(t) \qquad \omega(s) = s\,\phi(s)$$

$$X(s) = \frac{C}{2\pi}\,\phi(s) = \frac{C}{2\pi} \cdot \frac{1}{s}\,\omega(s)$$

$$X(s) = \frac{C}{2\pi} \cdot \frac{1}{s} \cdot \frac{K_M}{Js + R_F} \cdot V_M(s)$$

$$V_x(s) = Ke \cdot X(s)$$

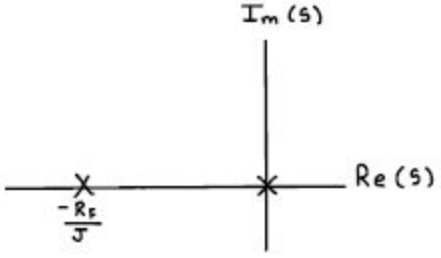
$$V_x(s) = \frac{K_M\,Ke\,C}{2\pi} \cdot \frac{1}{s} \cdot \frac{1}{Js + R_F} \cdot V_M(s)$$

$$P(s) = \frac{V_x(s)}{V_M(s)} = \frac{K_M Ke\,C}{2\pi} \cdot \frac{1}{s} \cdot \frac{1}{Js + R_F} = \frac{K_M Ke\,C}{2\pi} \cdot \frac{1}{s\left(s + \frac{R_F}{J}\right)}$$

$$P(s) = \frac{V_x(s)}{V_M(s)} = \frac{K_M Ke\,C}{2\pi J} \cdot \frac{1}{s\left(s + \frac{R_F}{J}\right)}$$

$$P(x) = \frac{K_M Ke \cdot r}{J} \cdot \frac{1}{s\left(s + \frac{R_F}{J}\right)}$$

poles: $s = 0$, $s = \frac{-R_F}{J}$



(3) poles of $H(s)$:

$$s^2 + \frac{R_F}{J} s + \alpha K_p = 0$$

$$s^2 + \frac{R_F}{J} = -\alpha K_p$$

$$s^2 + \frac{R_F}{J} s + \left(\frac{R_F}{2J}\right)^2 = \left(\frac{R_F}{2J}\right)^2 - \alpha K_p$$

$$\left(s + \frac{R_F}{2J}\right)^2 = \left(\frac{R_F}{2J}\right)^2 - \alpha K_p$$

$$s + \frac{R_F}{2J} = \pm\sqrt{\left(\frac{R_F}{2J}\right)^2 - \alpha K_p}$$

$$s = -\frac{R_F}{2J} \pm \sqrt{\left(\frac{R_F}{2J}\right)^2 - \alpha K_p}$$

case 1: $K_p = 0$

$$s\left(s + \frac{R_F}{J}\right) = 0$$

$$p_{1,2} = 0, \; -\frac{R_F}{J}$$

case 2: low K_p

$$K_p \leq \frac{1}{\alpha}\left(\frac{R_F}{2J}\right)^2 \qquad p_{1,2} = -\frac{R_F}{2J} \pm \sqrt{\left(\frac{R_F}{2J}\right)^2 - \alpha K_p}$$

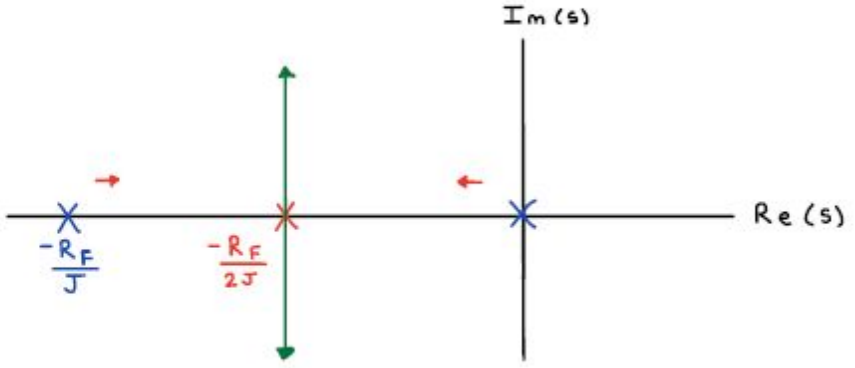
case 3: $K_p = \frac{1}{\alpha}\left(\frac{R_F}{2J}\right)^2$

$$p_{1,2} = -\frac{R_F}{2J}, \text{ double pole}$$

case 4:, complex poles, high K_p:

$$K_p > \frac{1}{\alpha}\left(\frac{R_F}{2J}\right)^2 \qquad p_{1,2} = -\frac{R_F}{2J} \pm j\sqrt{\alpha K_p - \left(\frac{R_F}{2J}\right)^2}$$

complex beyond $p_{1,2} = -\frac{R_F}{2J}$



Luis Perea

(4)

$$H(s) = \frac{\alpha k_p}{s^2 + \frac{R_F}{J} s + \alpha k_p} = \frac{\omega^2}{s^2 + 2\xi\omega s + \omega^2}$$

$$\omega = \sqrt{\alpha k_p}$$

$$2\xi\sqrt{\alpha k_p} = \frac{R_F}{J}$$

$$\xi = \frac{R_F}{2J} \cdot \frac{1}{\sqrt{\alpha k_p}} \qquad k_p = \left(\frac{\frac{R_F}{2J\xi}}{\alpha}\right)^2$$

$R_F = 0.3\ Nm\cdot s \quad J = 0.03\ Nm\cdot s^2 \quad Ke = 1\ \frac{V}{cm} \quad Km = 6\ \frac{Nm}{V} \quad r = 2\ cm$

$$\alpha = \frac{Km\,Ke\cdot r}{J} = \frac{6\ \frac{Nm}{V}\cdot 1\ \frac{V}{cm}\cdot 2\ cm}{0.03\ Nm\cdot s^2} = 400\ \frac{1}{s^2}$$

$$\frac{R_F}{2J} = \frac{0.3\ Nm\cdot s}{2(0.03\ Nm\cdot s^2)} = 5\ \frac{1}{s}$$

a) No overshoot : $\xi = 1$

$$k_p = \left(\frac{\frac{R_F}{2J\xi}}{\alpha}\right)^2 = \frac{\left(\frac{0.3\ Nm\cdot s}{2(0.03\ Nm\cdot s^2)\cdot 1}\right)^2}{400\ \frac{1}{s^2}} = \frac{25\ \frac{1}{s^2}}{400\ \frac{1}{s^2}} = \frac{1}{16} = 0.0625$$

$$H(s) = \frac{25}{s^2 + 10s + 25}$$

b) small overshoot : $\xi = \frac{1}{\sqrt{2}}$

$$k_p = \left(\frac{\frac{R_F}{2J\xi}}{\alpha}\right)^2 = \frac{\left(\frac{0.3\ Nm\cdot s}{2(0.03\ Nm\cdot s^2)\cdot \frac{1}{\sqrt{2}}}\right)^2}{400\ \frac{1}{s^2}} = \frac{50\ \frac{1}{s^2}}{400\ \frac{1}{s^2}} = \frac{1}{8} = 0.125$$

$$H(s) = \frac{50}{s^2 + 10s + 50}$$

Luis Perea

c) Settling time t_s

$$t_s = \frac{-\ln(0.02)}{\xi \omega}$$

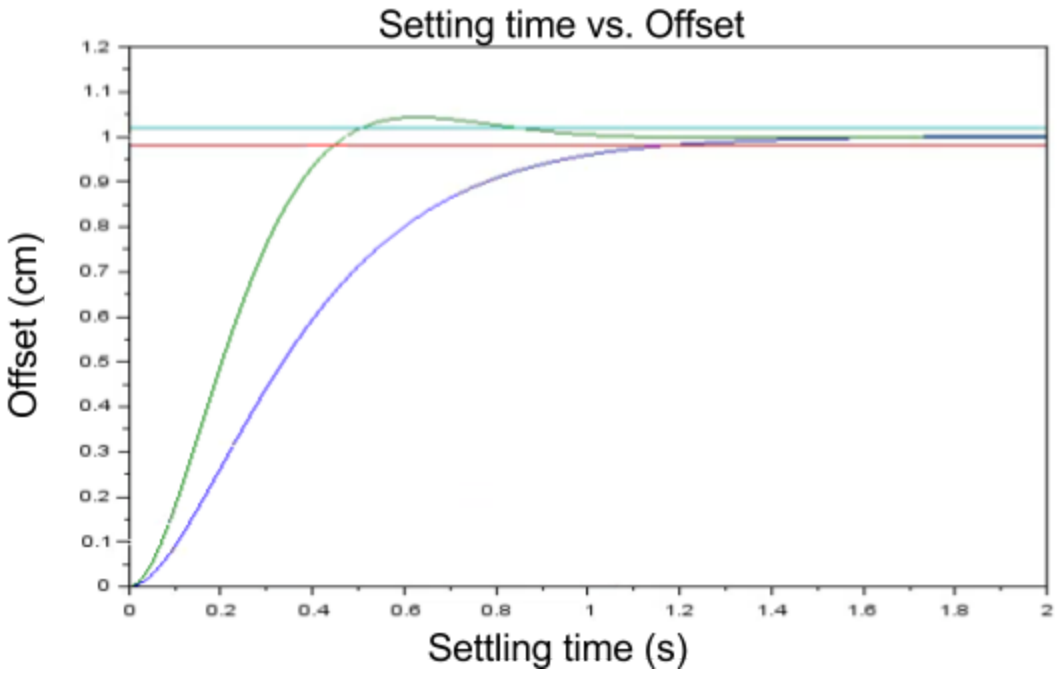
t_s for $k_p = 0.0625$, $\xi = 1$

graph: $t_s \approx 1.167 \approx 1.2$

t_s for $K_p = 0.125$, $\xi = \frac{1}{\sqrt{2}}$

$$t_s = -\frac{\ln(0.02)}{\frac{1}{\sqrt{2}} \cdot \sqrt{400 \cdot 0.125}} = 0.782405\,s \approx 0.8$$

graph: $t_s \approx 0.8433 \approx 0.8$



The figure above shows the settling time for two different damping coefficients. The first green line represents an overdamped case which overshoots the intended position, while the blue line shows a critically damped case which approaches the target position more smoothly. The first case approaches quickly but is more prone to error. Therefore, the second case with a settling time of 0.8s is more desirable in the real conditions. This simulation provides estimates for the PID values to be used in the real hardware, given that the correct motor constants can be acquired.

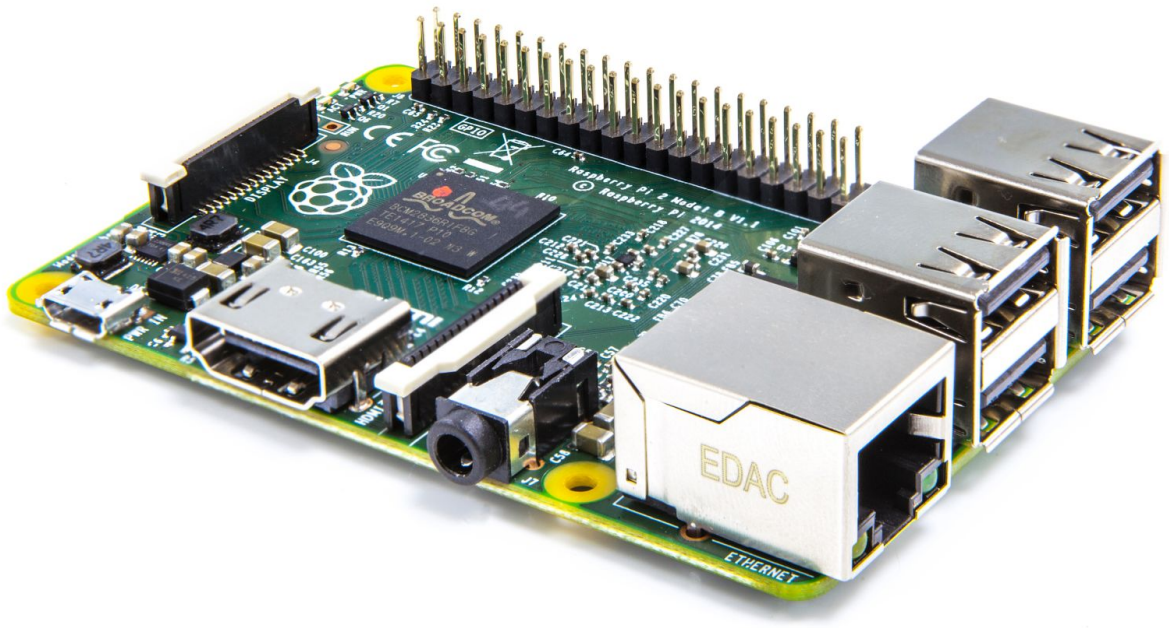
Creativity and Innovation

The design is creative because it combines an embedded system with an Internet-of-Things protocol with hospital equipment. Many devices in a common hospital have not been modernize, so updating devices to fit with the modern era of interconnectivity, automation, and information-sharing is the true innovation of the project. This project shows that it would not be necessary to completely revamp the current devices, but that it would be possible to create attachments to add internet-of-things connectivity retroactively. This is the other innovation component of the project, as to an oxygen blender, we believe we are the first group to use a raspberry pi and a stepper motor to adjust the FIO2 of an oxygen blender.

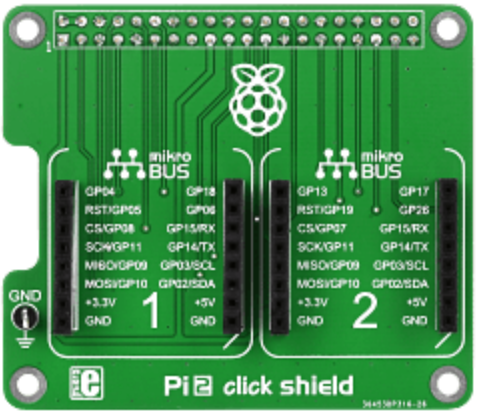
Final Design

Raspberry Pi

Switching to the raspberry pi was a decision to improve the capabilities of the system. While the Arduino offered a low-cost embedded system solution, the raspberry pi would provide more options and flexibility since it not a single processor but a full computer. The biggest appeal of the raspberry pi was it's built-in capabilities for external communication, such as using a touch screen to display a GUI, connecting to the internet easily, or connecting to multiple devices. While the raspberry pi would be more expensive, the expanded ability for an Internet-of-Things device was appealing, especially for a project whose purpose is to introduce autonomy and remote monitoring.



To begin the transition to the raspberry pi platform, an adapter for the heart-rate-click was acquired. This adapter would allow the raspberry pi to communicate with the heart-rate-click through the GPIO pins, functioning similar to a shield on an arduino.



After testing, it became apparent that the heart-rate-click would not provide suitable data for the pulse oximetry. It was crucial that the pulse-oximetry reading be as accurate as possible, considering that the SPO2 data would be vital to the operation of the system. Having noisy or inconsistent data could potentially be the difference between life and death, or at least be the cause of multiple false alarms. This situation would not be tolerable however, especially in a medical setting. We then asked our clients for advice, specifically on what pulse oximeter is used in real-world applications.

KP Kamat, Pradip <Pradip.Kamat@choa.org>
Thu 12/15/2016, 4:55 PM
Luis Perea

Reply all

Inbox

Masimo
Sorry for delayed response

Sent from my iPhone

LP Luis Perea
Fri 12/9/2016, 10:54 PM

Hello Dr. Kamat!
Would you be able to tell me a brand and model information for the pulse oximeter that CHOA uses?
I would like to buy one to use for the project. Thank you!

Pulse Oximeter

MASIMO is the brand of pulse oximeter that is used at Children's Healthcare of Atlanta and many other hospitals in the United States. This brand is well used due to it's extensive testing, reliability, and durability. When it comes to patient safety, it is important to be utilizing the most reliable technology to ensure accurate readings in potentially life-threatening situations. The pulse oximeter contains 16-bit resolution, 1% SpO2 sensitivity, and +/- 3% accuracy (Manning, 2010). The pulse oximeter has been through numerous FDA tests and evaluations that there was no doubt it would be suitable to use as a data acquisition device. This way, the project would be using the same device and protocol as the hospital itself, reducing the need for training and maintenance, and allowing for backwards-compatibility with pulse oximeters currently in use. With this in mind, we started researching the different products and sensors, and found that a few of the products feature an RS-232 serial output. This product that was decided to be purchased: the Masimo Radical 7. It serves as an analysis system between the pulse oximeter and an external device such as a bedside monitor.

To go along with this, it would be necessary to acquire the actual pulse oximeter, the Masimo Rainbow Pulse Oximeter.

The Masimo Radical 7 system mainly consists of the detachable monitor screen combined with the sensor. The majority of the device comes from the detachable docking station which serves to charge the detachable monitor screen and provides the various outputs. The docking station features different communication outputs, depending on the model. The model shown below is the RDS-1, with three different ports. The serial port, the analog port, and the nurse call port.

For the purpose of the project, we will use the serial port to connect to the raspberry pi. Since the Raspberry pi lacks the RS-232 protocol, we have acquired a USB-to-RS232 adapter to attempt communication between the two devices. We acquired the UGREEN adapter with the PL2303 chipset after some trial and error, as the docking station port has a small width clearance. The PL2303 chipset should emulate the intel hardware required for the RS-232 interface and should be usable with the raspberry pi after some driver modifications.

The importance of this system is that it is medical-grade equipment. Notably the device features multiple filters to achieve the highest sensitivity in the market (Shah). Pulse oximeters “sensitivity” is measured in their ability to measure “true alarms”, or situations of dangerous SpO2 levels, while avoiding “false alarms” where the machine falsely detects a SpO2 value below 90% than the true value. For clinical trials, the masimo sensor received a 100% sensitivity.

The figure above shows which wavelengths correspond to what compound. This is how the pulse oximeter senses changes in each compound.

Below is an image of Luis Perea testing the Masimo pulse oximeter.

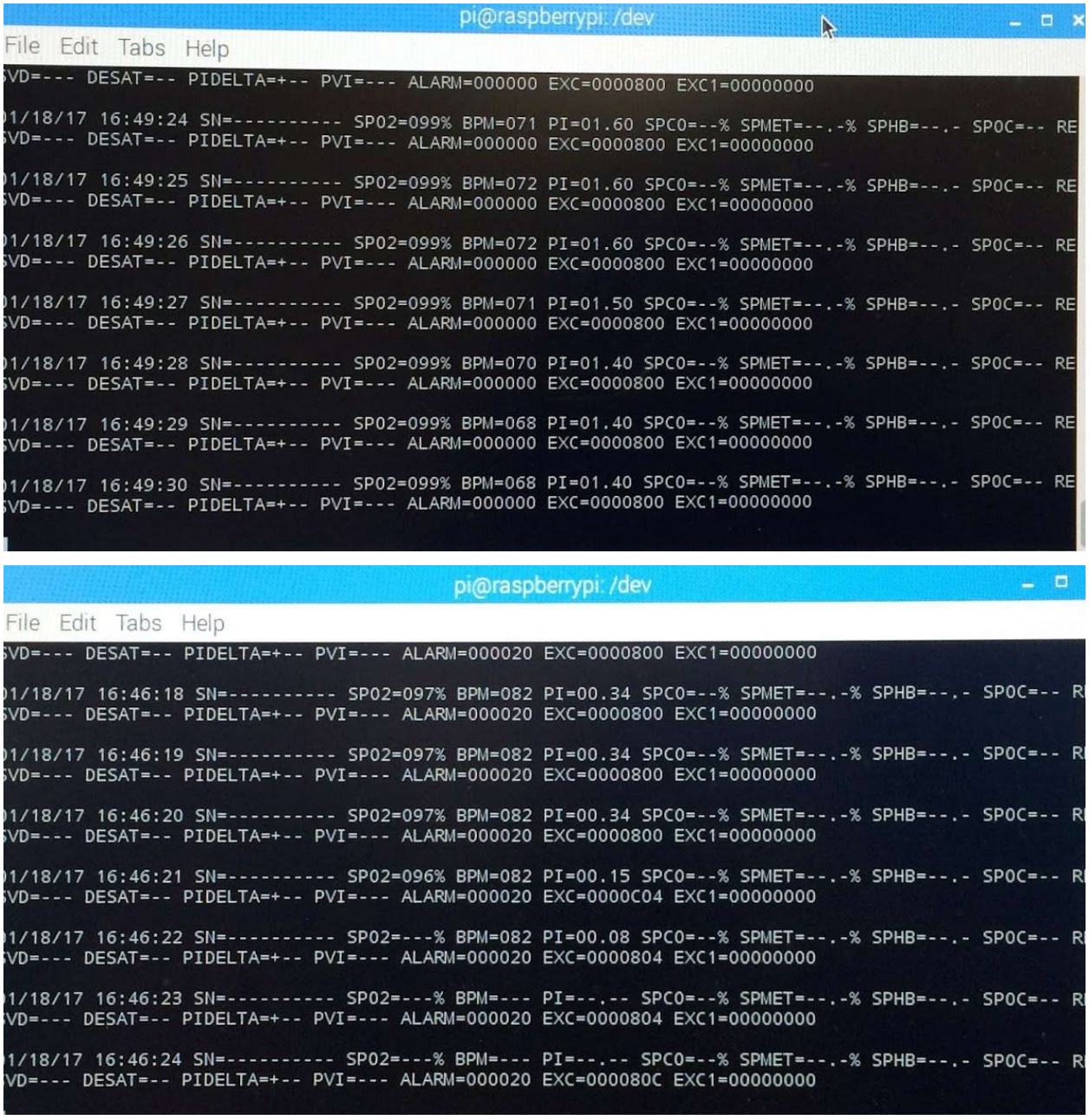
At the moment, it reported that the SpO2 levels were 99%, and that heart rate was 76 beats per minute. A heart beat waveform can be seen along the bottom.

To start, Luis connected the Raspberry Pi to the Masimo RDS-1 dock using the USB-to-Serial adapter. Luis then set patient monitor to output its data through the serial port in the ASCII data format. On the command terminal of the raspberry pi, Luis issued the following command:

“stty -F /dev/ttyUSB0 9600 cs8 -parenb -cstopb -crtscts”

To set the Baud rate to 9600, 8 bits per character, no parity check, 1 bit start 1 bit stop, and no handshaking. With this, Luis was able to use the command:

“cat /dev/tttyUSB0” to view the output stream.



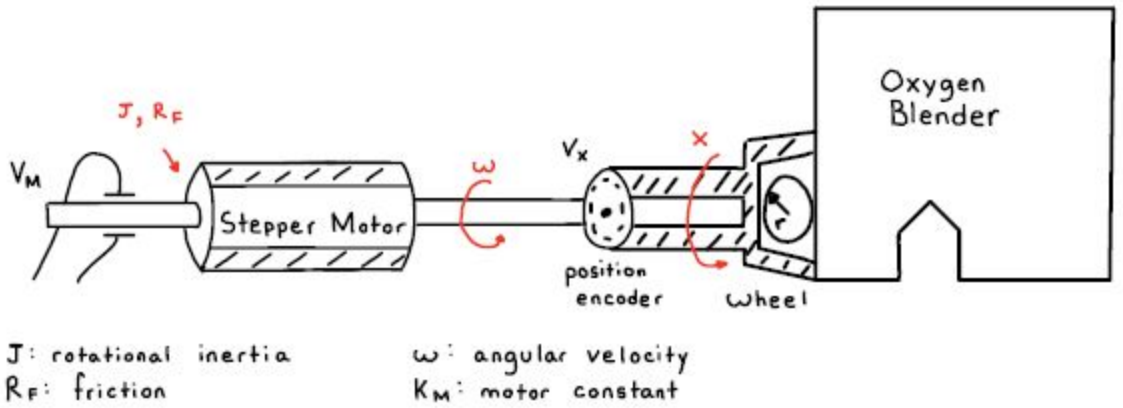
The following python script was used to achieve the terminal output.

```
#!=/usr/bin/python
        import serial
        serial_monitor = serial.Serial (
        port = 'dev/ttyUSB0',\
        baudrate = 9600,\
        parity = serial.PARITY_NONE,\
        stopBits = serial.STOPBITS_ONE,\
        byteSize = serial.EIGHTBITS,\
        timeout = 0)

        while TRUE:
              line = serial_monitor.readLine()
              if (line == 1):
                 string = line;
                 words = string.split()
                 print(words[3])
                 print(words[4])
        serial_monitor.close()
```

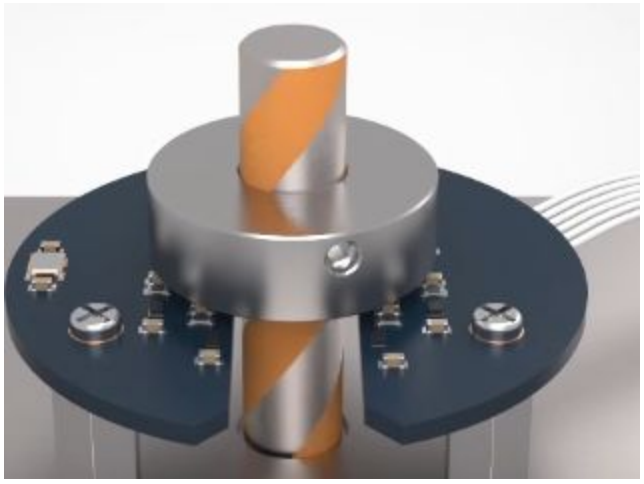
Absolute Encoder

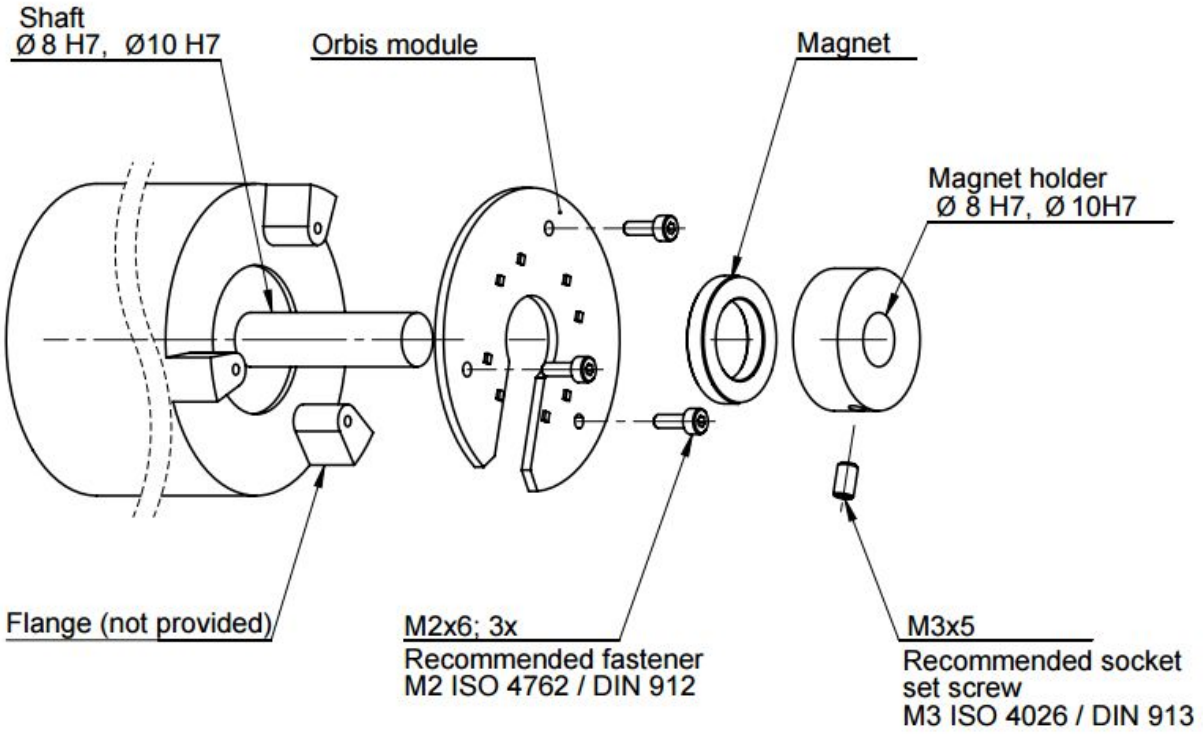
After that, Luis began testing an absolute encoder, the RLS Orbis absolute rotary encoder, which features SPI communication and 14-bit resolution. Hopefully this will be able to automatically determine how far the knob has been rotated by the stepper motor in a closed-feedback loop.

Besides the software, the other main component of this project must be the feedback control system to let the embedded microcontroller/processor know where the rotating knob is in relation to the oxygen blender's control. As mentioned previously, we will be attempting to create a closed-feedback loop using the raspberry pi as the embedded microprocessor for program decision making, a stepper motor to act as an actuator for the rotating knob, and an absolute rotary encoder to provide a closed-feedback loop of the degree-rotation of the knob based on the rotation of the rod that connects the stepper motor to the knob. A schematic of this proposed system can be seen below.



The position encoder will be implemented using the RLS Orbis true absolute rotary encoder. The encode consists of two main pieces, the main pcb board that houses the processor, and a magnet to serve as a Hall Effect sensor to decrypt the angle of rotation.

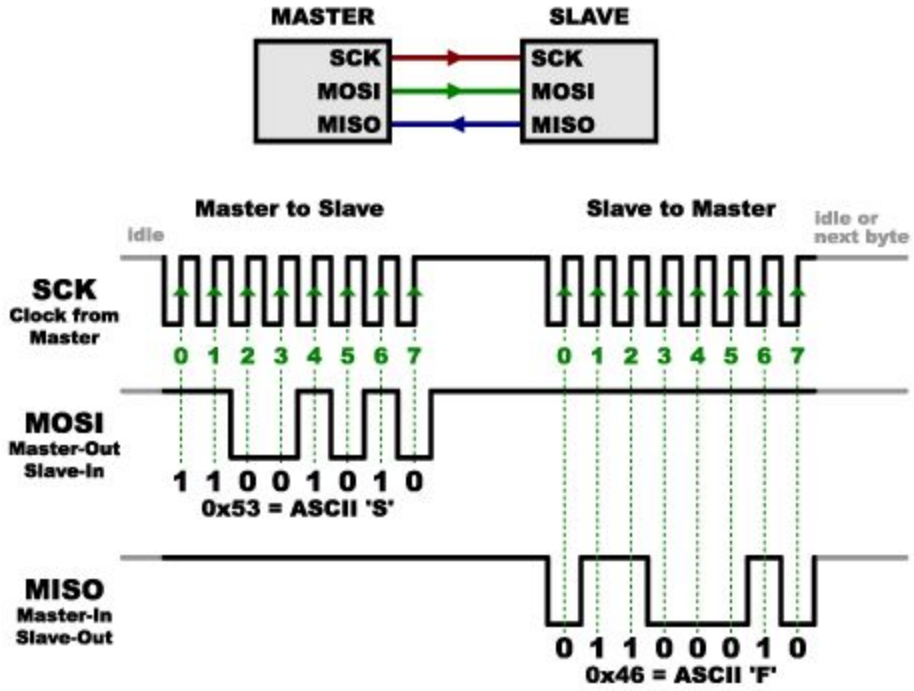




The version we acquired states it supports a 8mm diameter rod to fit between the magnet, however when we attempted to connect an 8x100mm rod, the rod did not fit through the magnet so we will later attempt to use a 7mm diameter rod. We also needed an adapter to connect the stepper motor's 5mm diameter shaft to the rod, and we will attempt to use this adapter for the 7mm rod and tighten it.

According to the datasheet, this encoder features 14-bit resolution, 4-6 V input supply voltage, SPI communication, a maximum rotation speed of 10,000 rpm, and a sensitivity of 0.2°. There were other options available, each with a different communication system, such as Asynchronous Serial, PWM, SSI, BISS-C, and SPI. The interface that our team members are most familiar with is Serial, and we almost purchased this model but chose not to because we was not sure if we could convert an RS-232 compliant signal into a signal that an arduino or raspberry pi could read. we understood that the raspberry pi comes with the ability to use the SPI interface through its GPIO pins, and we decided that this would be the best version to test.

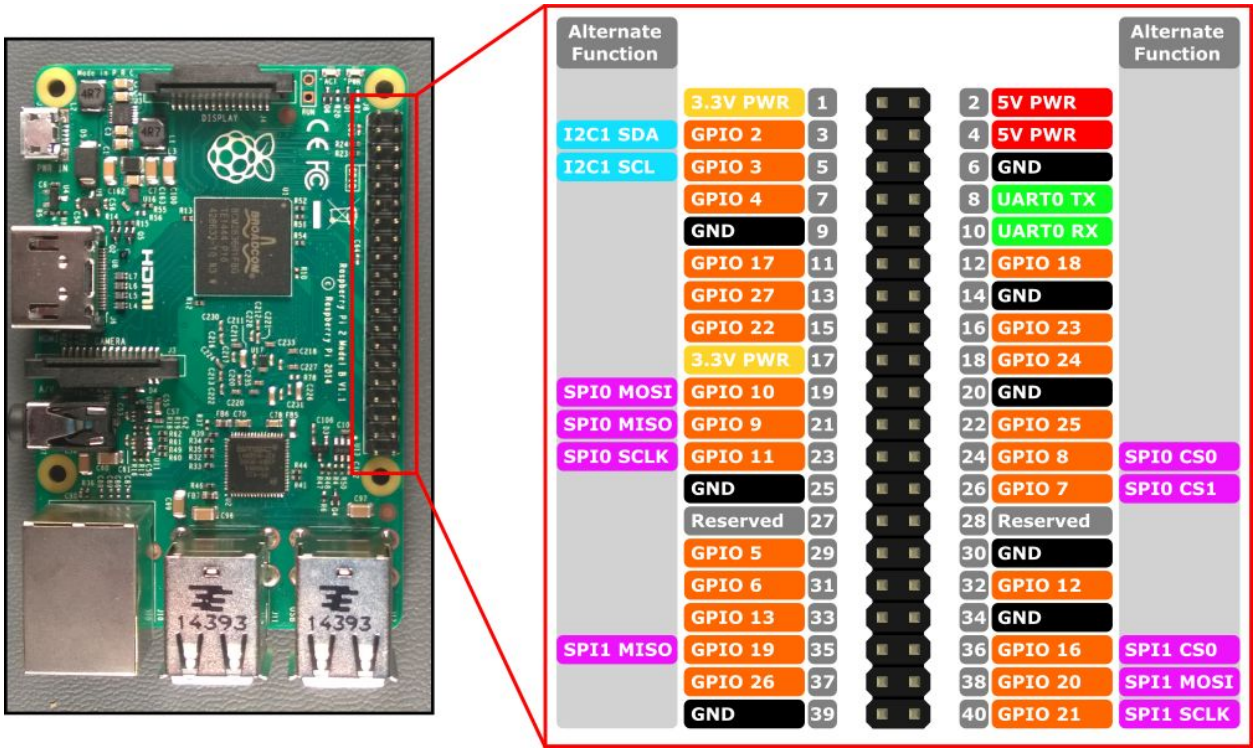
We read about the SPI communication interface to try to understand it more. Unlike Serial communication which requires start and stop bits to keep data from becoming noisy or disrupted, the SPI protocol has one device, called the "master", generate a clock signal which the receiving device, called the "slave", uses to know when to read in data from a separate data line. In order for the two devices to communicate bidirectionally, there must be a cable for data from master to slave, and a cable for data from slave to master. The benefit of this is that the data is in sync across the devices, as seen in the figure below (Grusin, 2013).



The above image represents an example of how SPI communication works. The important concept of SPI is that the two devices are always in sync because of the master's clock signal being used to tell the slave device when to read and write data to and from the data lines. This makes the use of stop and start bits obsolete from asynchronous serial communication.

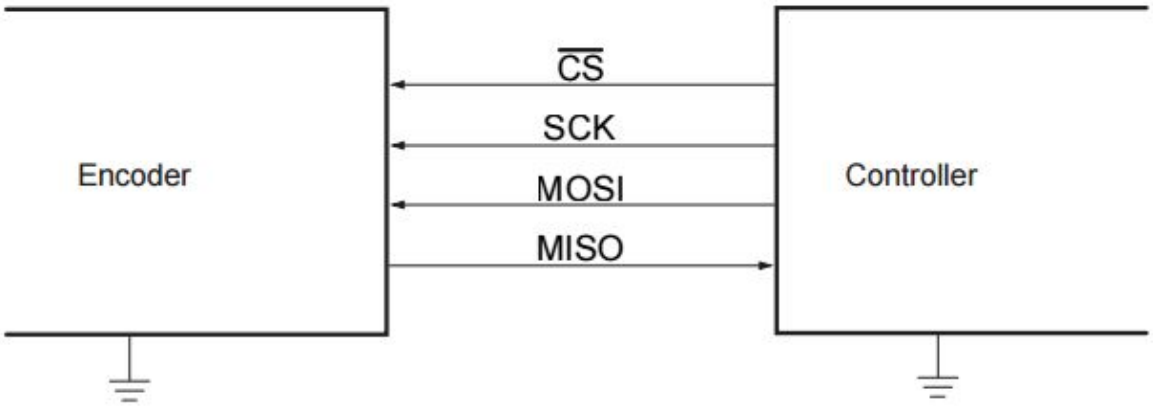
The important connections are the MISO, MOSI, and SCLK pins as they are the fundamental connections for the SPI protocol. The connections between the rotary encoder and the GPIO pins can be seen in the table below.


R-Pi GPIO
left
bottom
P1-01
top
P1-02
Board Edge
3V3 Power
5V Power
R1: GPIO 0 (SDA)
R2: GPIO 2 (SDA)
5V Power
R1: GPIO 1 (SCL)
R2: GPIO 3 (SCL)
Ground
GPIO 4
(GPCLK0)
GPIO 14 (TXD)
Ground
GPIO 15 (RXD)
GPIO 17
GPIO 18
(PCM_CLK)
R1: GPIO 21
R2: GPIO 27
Ground
GPIO 22
GPIO 23
3V3 Power
GPIO 24
GPIO 10 (MOSI)
Ground
GPIO 9 (MISO)
GPIO 25
GPIO 11 (SCLK)
GPIO 8 (CE0)
Ground
GPIO 7 (CE1)
P1-25
bottom
P1-26
top
right
R1: Revision 1
R2: Revision 2



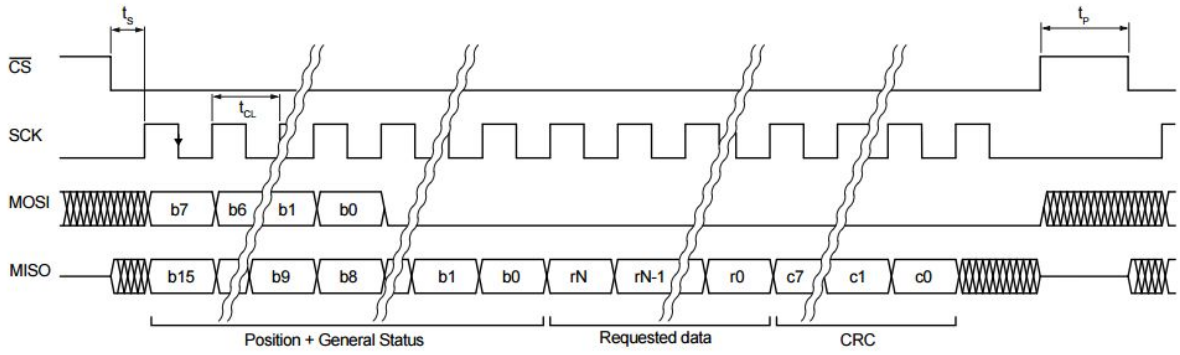
The above image shows which GPIO pins are required to interface between a raspberry pi and and an SPI device. The MISO/MOSI are data lines, the SCLK pin is for clock output, and the CE1/CE0 pins are for “chip selects”, or selecting which SPI device to connect to if there are two. To be able to use the SPI protocol for the raspberry pi, I had to enable it in the configuration settings and download a library wrapper for the raspberry pi, such as “wiringPi” or “spidev”. We used the spidev wrapper to test the SPI library. The Spidev wrapper is available for download at <https://github.com/doceme/py-spidev.git>.

The datasheet specifies that every data packet transmission begins with the CS line being low, which we believe means the CE0 or CE1 line, and that this low bit begins the transmission
The instructions for SPI communication from the encoder’s datasheet can be seen in the figure below.



Signal	Description
$\overline{CS}$	Active low. $\overline{CS}$ line is used for synchronisation between master and slave devices. During communication it must be held low. Idle is high. When $\overline{CS}$ is high, MISO line is in high-Z mode. This allows connection of multiple slaves in paralell, sharing all lines except $\overline{CS}$.
SCK	Serial clock. Shifts out the data on rising edge.
MOSI	Master output ⟶ Slave input. Command from the controller to encoder.
MISO	Master input ⟵ Slave output. Data is output on rising edge on SCK after $\overline{CS}$ low. When $\overline{CS}$ is high, MISO line is in high-Z mode.

SPI timing diagram (single-turn)



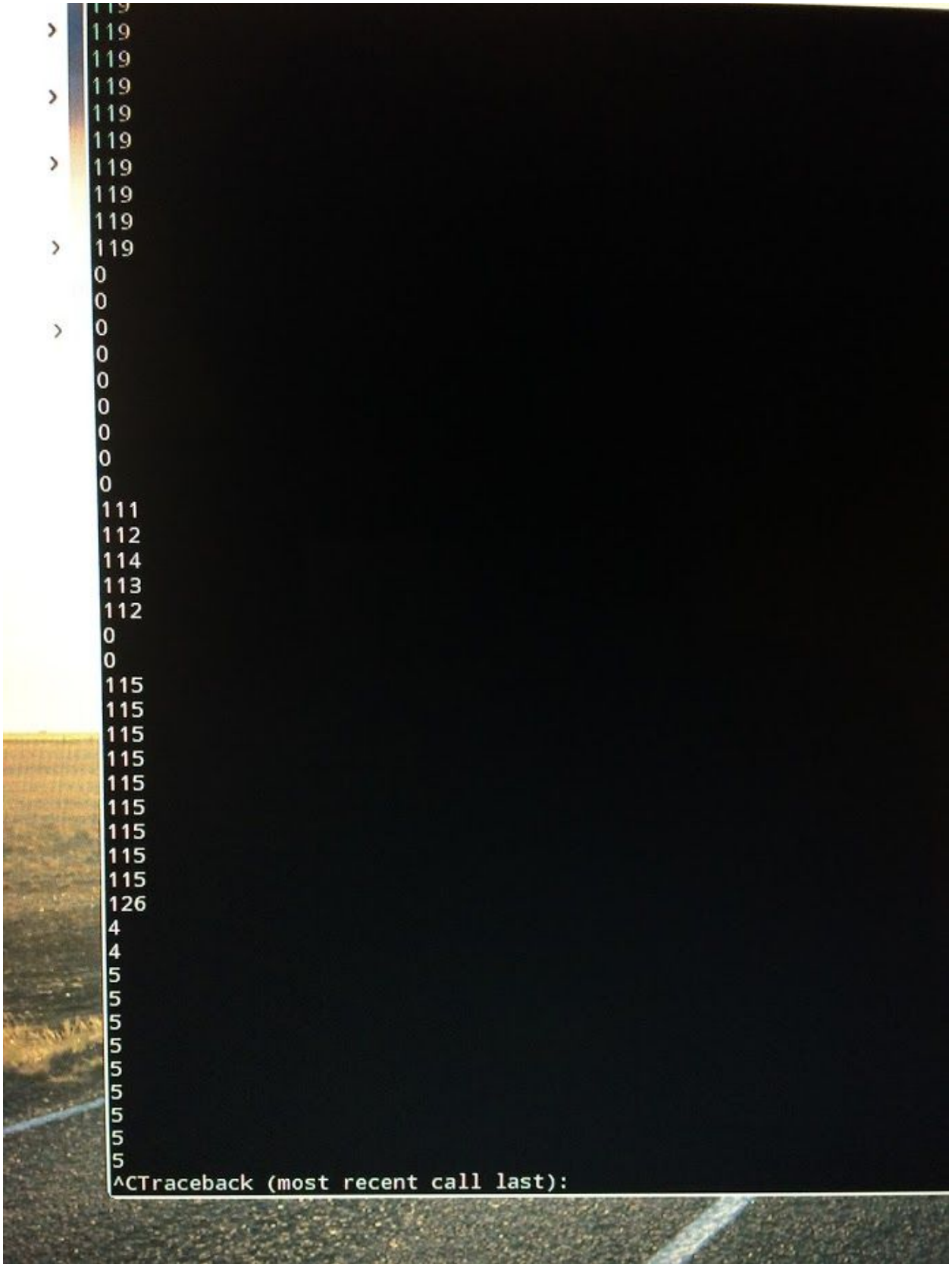
Communication parameters

Parameter	Symbol	Min	Typ	Max
Clock period	t_{CL}	250 ns		20 µs
Clock frequency	f_{CL}	50 kHz		4 MHz
Time after $\overline{CS}$ low to first SCK rising edge	t_S	1.25 µs		
Pause time	t_P		5 µs	

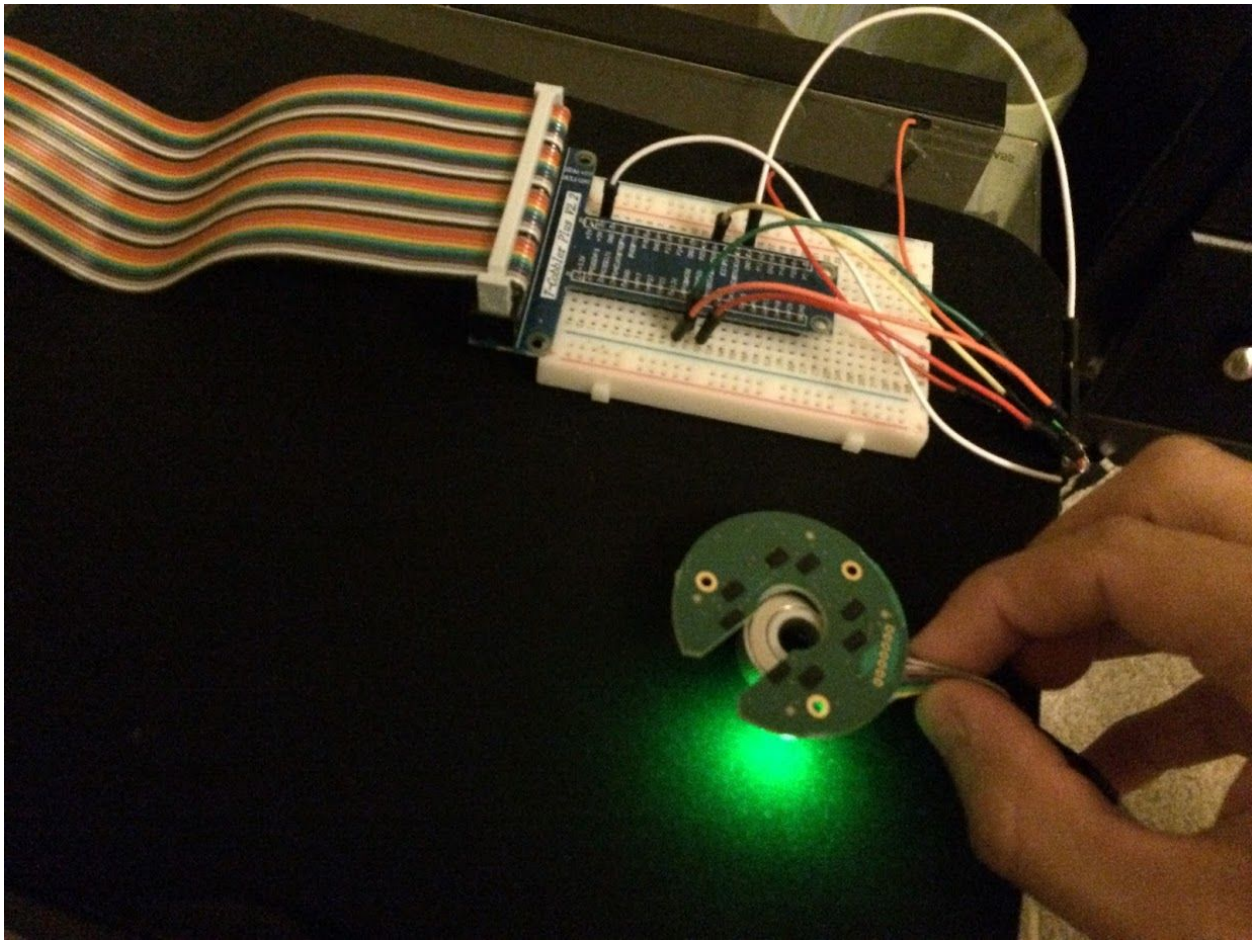
To start, we connected the wires of the encoder to the raspberry pi as instructed in the wiring table and the pin diagram. We wrote the following script to test the encoder.

```
#!/usr/bin/python

import spidev
import time

spi = spidev.SpiDev()
spi.open(0,0)

while True:
      resp = spi.xfer([0x00])
      print resp[0]
      time.sleep(1)
spi.close()
```

The beginning of the program imports the necessary libraries. Next an spidev object is instantiated and used to open the first CS device on line CE0. A while loop is then used to continuously read and write data from the encoder. There are two functions to send data, xfer, and xfer2. I noticed that xfer2 keeps the CS line low the entire time throughout transmission. According to the datasheet, all communication must be started with a low pulse of the CS line, which is what the xfer command does. Using xfer2 resulted in all 0's being read by the raspberry pi. The program then then print the data to the console. Example dataoutput can be seen in the figure below.


119
119
119
119
119
119
119
119
119
0
0
0
0
0
0
0
0
0
111
112
114
113
112
0
0
115
115
115
115
115
115
115
115
115
126
4
4
5
5
5
5
5
5
5
5
^CTraceback (most recent call last):

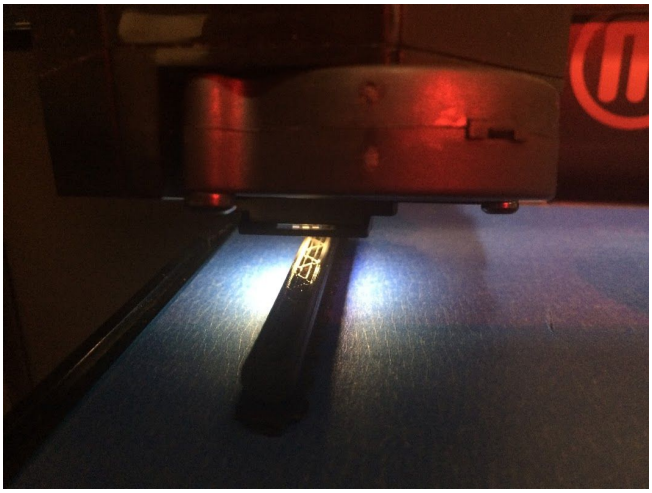
It appears that the dataoutput is an integer in the range of 0 to 360. If this is correct, then we believe the output is the current degree rotation of the magnet, which should be attached to the rod indicating the rotation of the rod. An image of this test can be seen in the figure below. The encoder also includes an LED that changes between red, yellow, and green which indicates the status of the encoder and it's ability to accurately read the magnet from the eight hall sensors.



We will continue working on the feedback mechanism and try to implement the encoder into the stepper motor. We will need to find a new adapter to fit the 5mm shaft to the 7mm magnet hole. We will also need to verify that the output is indeed the rotation of the magnet. After this, we will need to continue working on the software to store and analyze the pulse oximeter data and upload it to a database for remote monitoring.

Stepper Motor

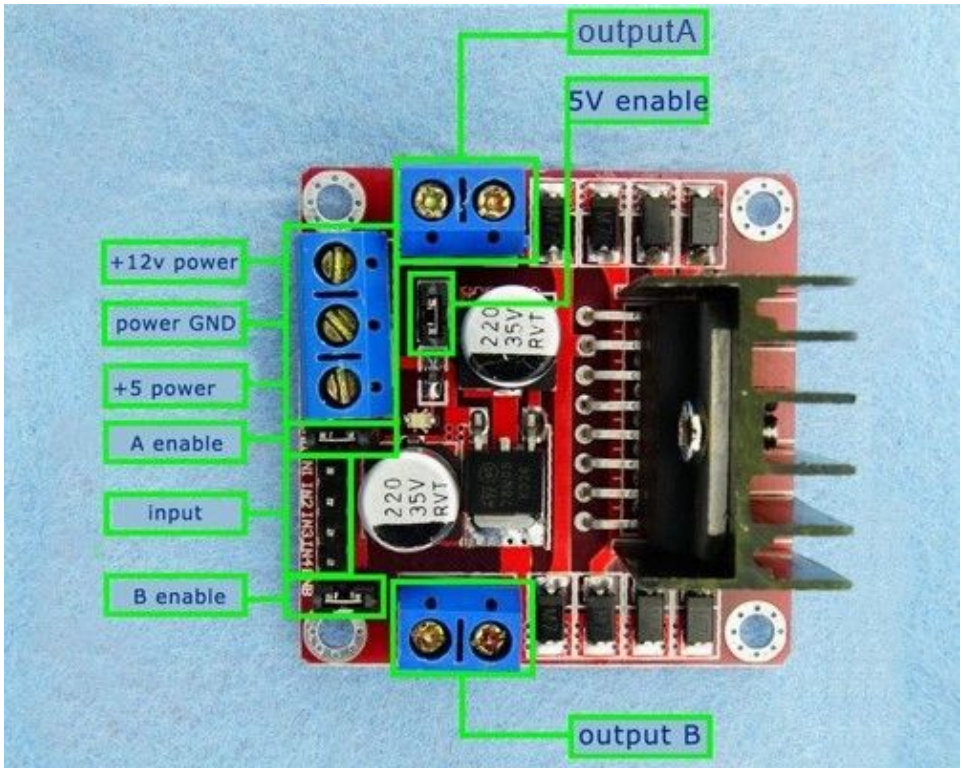
We still needed to find a better fit for the stepper motor shaft to rod adapter. It was rated for Ø8 size, which should be 0.8mm, but my 0.8mm rod did not fit. Instead we tried printing a slightly smaller than 0.8mm rod.



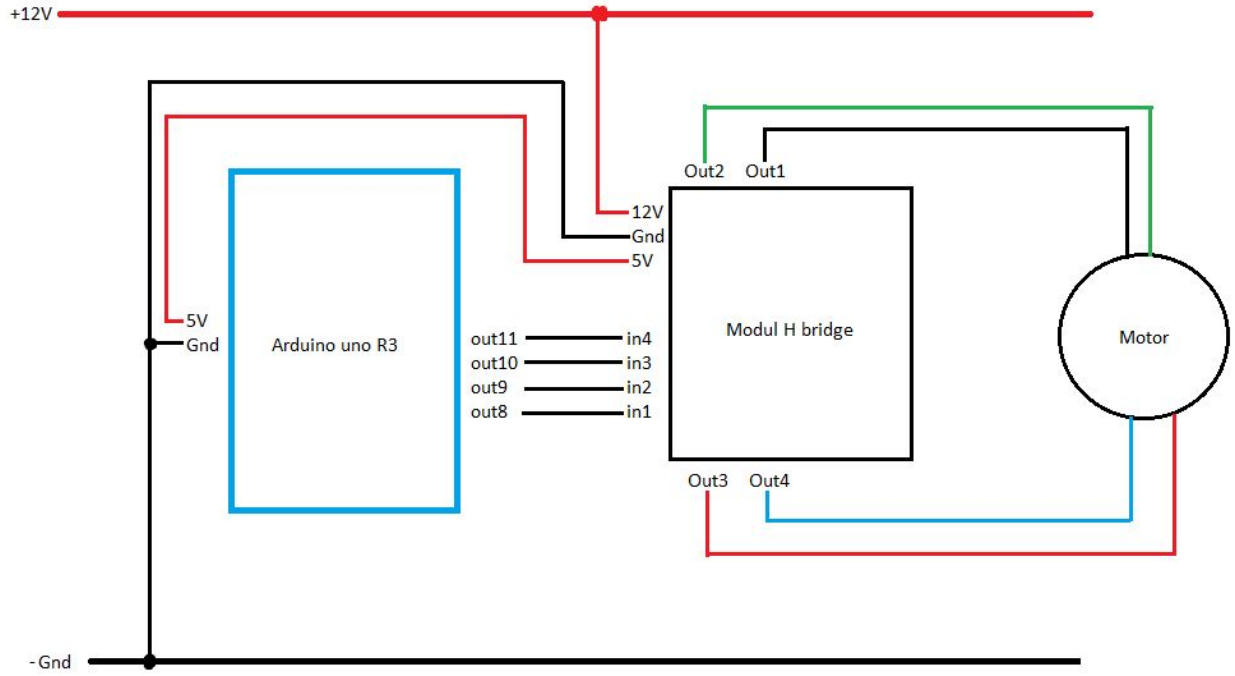
However this piece needed a lot of force to enter the adapter, so we will consider printing another smaller one. For now, I have found the screwdriver needed on the adapter, which was the smallest Allen key of about 1mm.

As for now, we will need to get the stepper motor working with the Raspberry Pi. The motor driver we used with the arduino was the L298N Dual H Bridge DC Motor, which is necessary to power a motor from a microcontroller. A diagram explaining a way to connect the motor driver to

the raspberry pi can be seen below. We are using this motor driver because it requires relatively few connections between the stepper motor and the I/O pins, and because this motor driver does not completely block access to the rest of the GPIO pins.

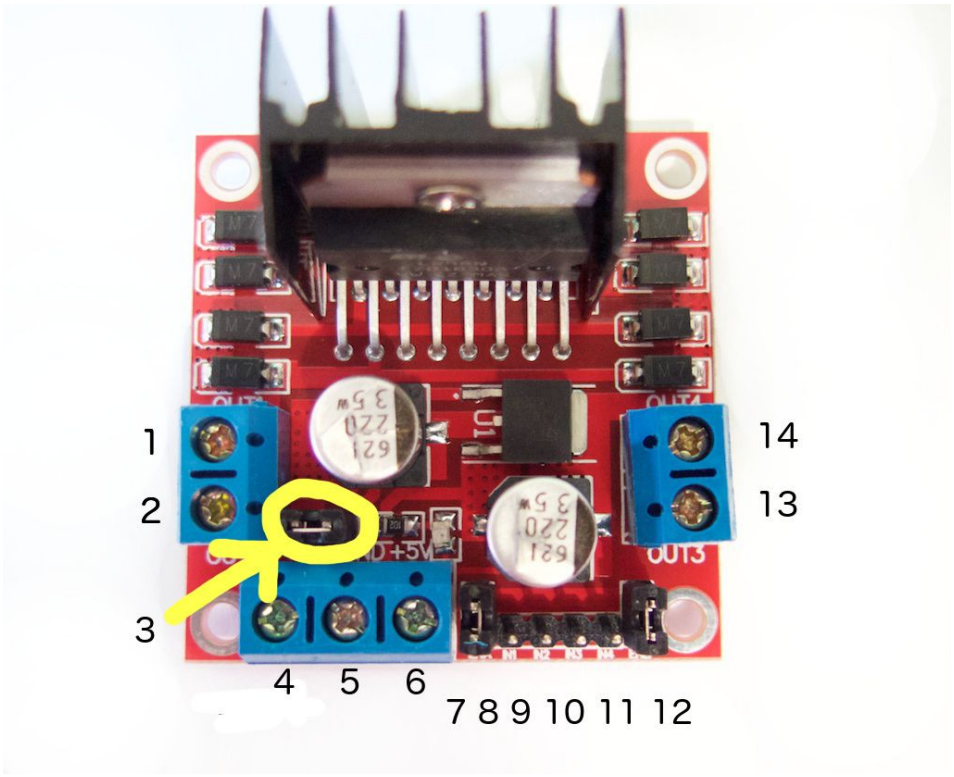


We will need to have both sides, output A and output B connected, as the stepper motor is a 4-wire Bipolar motor, meaning it has four different coils that need to be activated in a sequence to turn in a desired direction. This means we will need both A and B enabled, along with using all four inputs of the driver. It will look similar to the schematic below.



The 5V input is used as a logic input. For identification, we will refer to the wires of the stepper motor as follows:

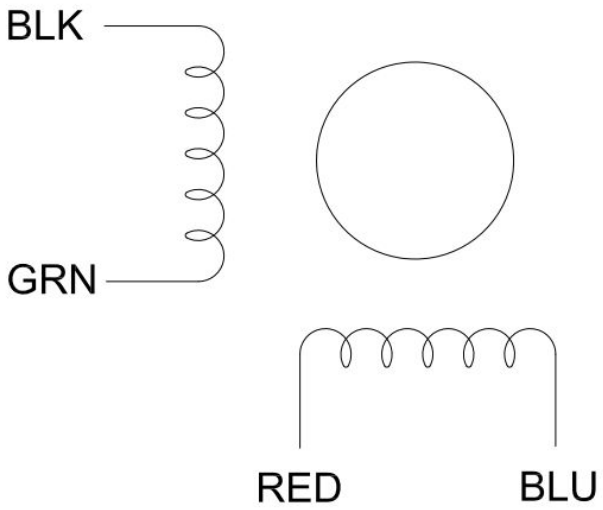
Black (A+)
Green (A-)
Red (B+)
Blue (B-)



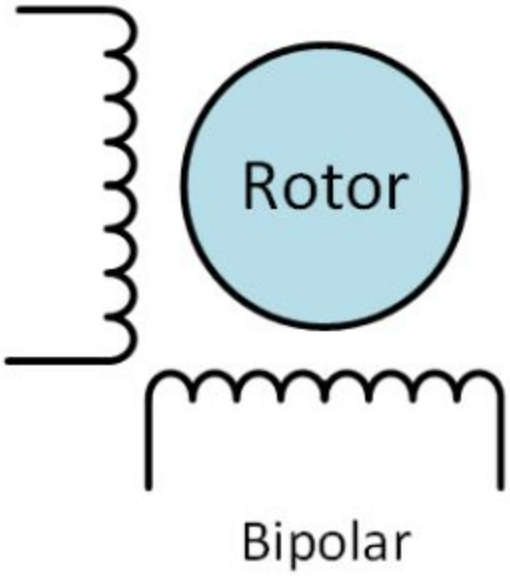
1. Stepper Motor A+
2. Stepper Motor A-
3. 12V Jumper - if an input voltage of 12V or more is applied, this pin would be removed to enable to 5V output
4. VCC, 5V - 32V.
5. GND, connect to voltage supply ground and microcontroller ground
6. 5V output if supply voltage is greater than 5V. Potentially could power microcontroller through this but better to use separate power supply.
7. Motor A enable.
8. Motor A+ input
9. Motor A- input
10. Motor B+ input
11. Motor B- input
12. Motor B enable
13. Stepper Motor B+
14. Stepper Motor B-

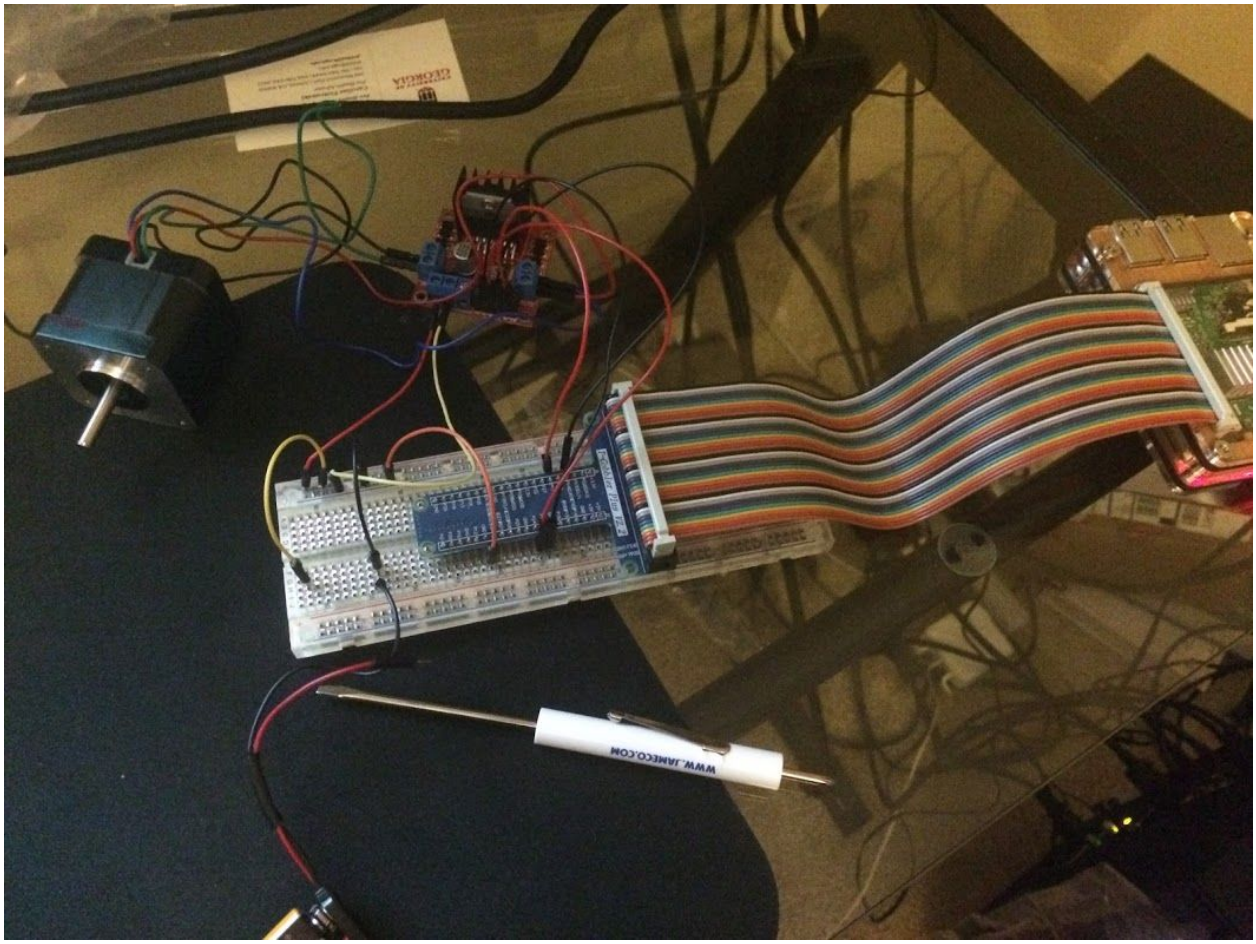
Connection 1 uses about 3.3V - 4.5 V during use.
Connection 2 uses about 1.1V - 3.3 V during use.

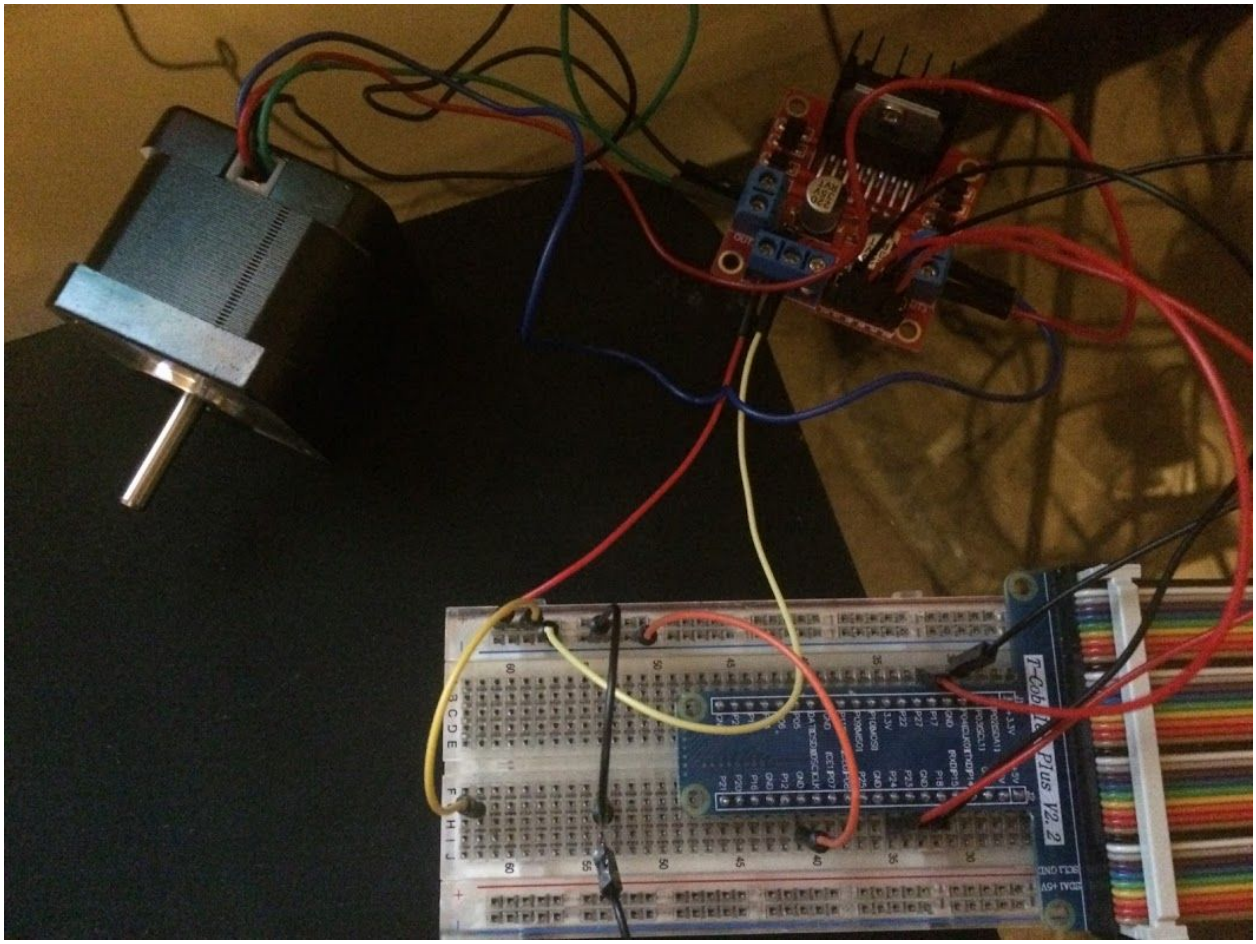
According to the NEMA 17 Stepper motor 0.9 deg data sheet (model number: 17HM15-0904S), the coils on the inside are connected as follows.



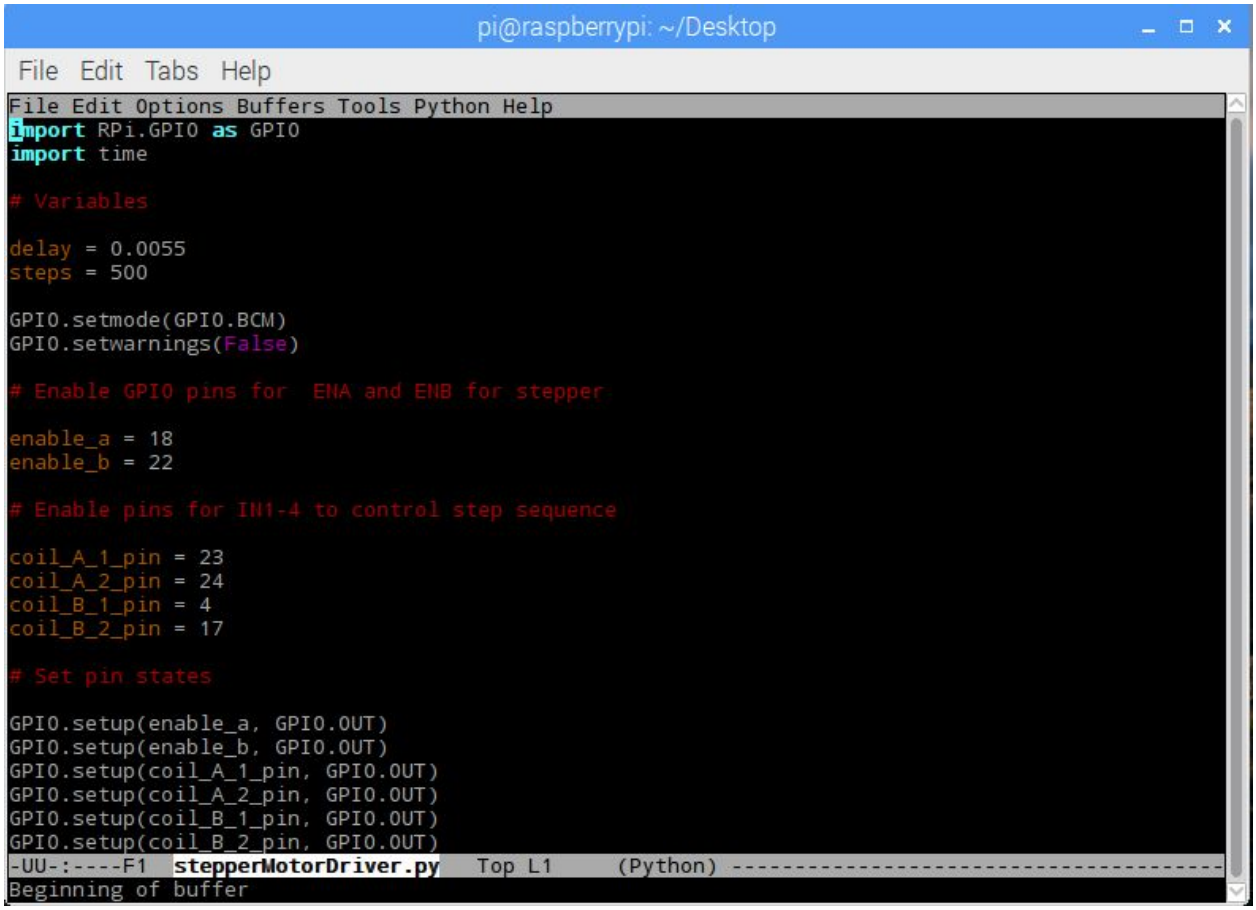
This confirms that the stepper motor is a bipolar stepper motor, where bipolar drivers use H-bridge circuitry to reverse the current flow through the phases. Energizing the phases with alternating the polarity causes the coils to work together to move the motor in one direction.




WWW.JAMECO.COM



We have been testing the stepper motor with the following python script based on Adafruit's stepper motor script:




```
#!/usr/bin/python
import time
import RPi.GPIO as GPIO

delay = 0.0055
steps = 400

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

# Enable pins for IN1-4 to control step sequence

coil_A_1 = 23
coil_A_2 = 24
coil_B_1 = 4
coil_B_2 = 17

# Set pin states
```

```
GPIO.setup(coil_A_1, GPIO.OUT)
GPIO.setup(coil_A_2, GPIO.OUT)
GPIO.setup(coil_B_1, GPIO.OUT)
GPIO.setup(coil_B_2, GPIO.OUT)

# Function for step sequence

def setStep(w1, w2, w3, w4):
  GPIO.output(coil_A_1, w1)
  GPIO.output(coil_A_2, w2)
  GPIO.output(coil_B_1, w3)
  GPIO.output(coil_B_2, w4)

# Forward

for i in range(0, steps):
    setStep(1,0,1,0)
    time.sleep(delay)
    setStep(0,1,1,0)
    time.sleep(delay)
    setStep(0,1,0,1)
    time.sleep(delay)
    setStep(1,0,0,1)
    time.sleep(delay)

# Reverse

for i in range(0, steps):
    setStep(1,0,0,1)
    time.sleep(delay)
    setStep(0,1,0,1)
    time.sleep(delay)
    setStep(0,1,1,0)
    time.sleep(delay)
    setStep(1,0,1,0)
    time.sleep(delay)
```

We found that the stepper motor did fully rotate in the forward direction, but stalled when trying to turn in the backwards direction. This may be because of wiring or the code and we will need to do additional tests to debug it. Another issue we see is trying to accurately control how many degrees to turn and in what direction, as we need to script the enable outputs in a correct sequence for a set amount of time to accurately turn in a given direction for a given amount of degrees. This is in contrast to a servo, which would directly turn to the angle we need. Unfortunately the servo lacks the full degree of motion this project would require, so the stepper motor and encoder combination will need to suffice.

Additionally the team would like the system to have a database connection for remote monitoring. To start, we would be using MySQL. We have made a draft for the code that parses the SPO2 pulse oximeter and inserts values into a database.

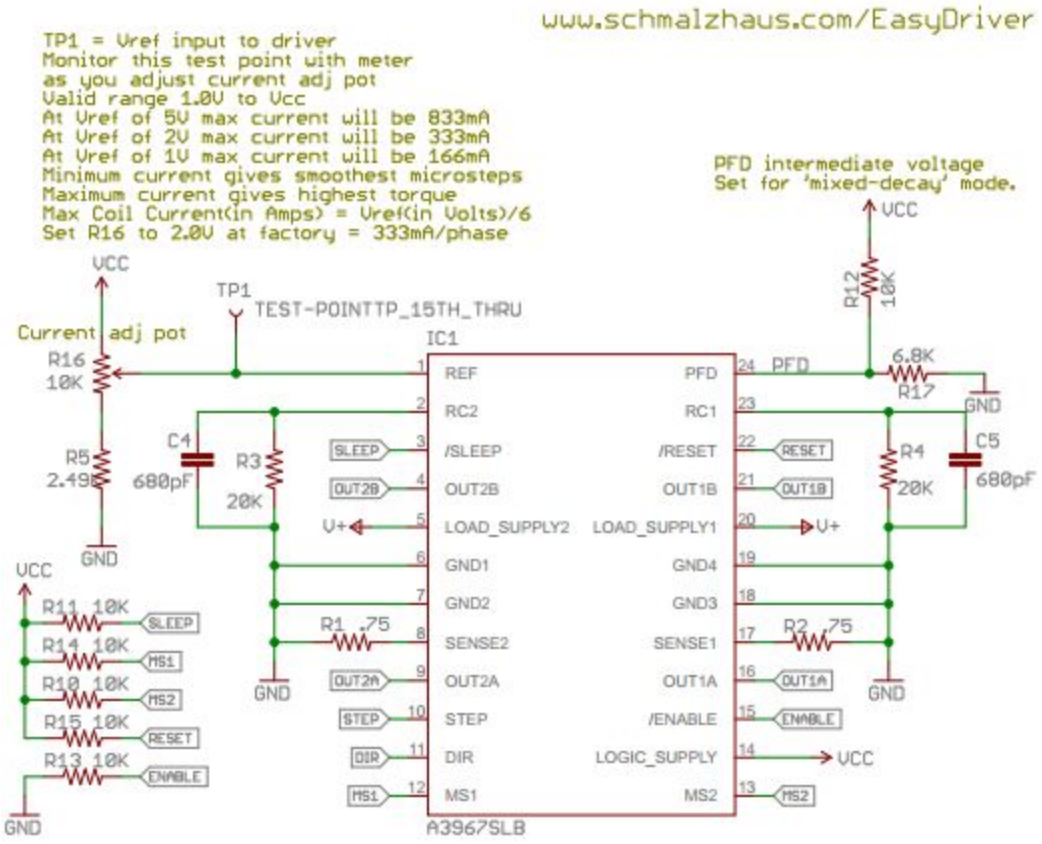
```
#!=/usr/bin/python

import serial
Import MySQLdb

#set up Serial Monitor at TTYUSB0
serial_monitor = serial.Serial (
port = ‘dev/ttyUSB0’,\
baudrate = 9600,\
parity = serial.PARITY_NONE,\
stopBits = serial.STOPBITS_ONE,\
byteSize = serial.EIGHTBITS,\
timeout = 0)

#open database connection
db = MySQLdb.connect(“host”, “user”, “login”, “database”)

cursor = db.cursor()

#select ID = 1
cursor.execute(“SELECT qSQL FROM DATABASE WHERE id = 1”)

#get Row
results = cursor.fetchone()

qSQL = results[0]

cursor.execute(qSQL)
```

```
#parse serial input for SPO2 and BPM
while TRUE:
      line = serial_monitor.readLine()
      if (line == 1):
            string = line
            words = string.split()
            print(words[3]))
            print(words[4]))

      results = cursor.fetchall()
      for row in results:
            SPO2 = row[0]
            BPM = row[1]

            #Query to insert
            cursor.execute(‘’’INSERT into TABLE (SPO2, BPM)
                        values (%s, %s)’’’, (SPO2, BPM))
            #save changes
            db.commit()

#exit connections
db.close()
serial_monitor.close()
```

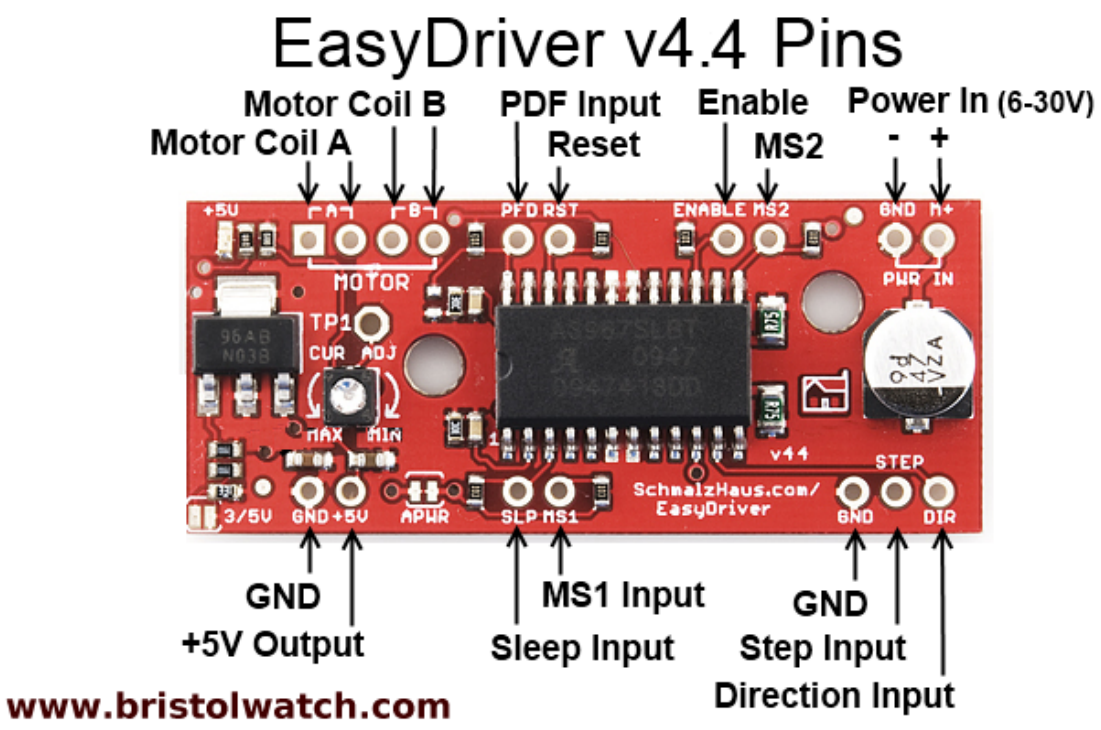
So we will continue to work on the stepper motor control, parsing the pulse oximeter, and controlling the feedback mechanism. We will need a data structure to store the pulse oximeter readings and analyze them. An extra feature would hopefully be displaying the data on a remote monitor or phone. Once we have the data structure, we will need to make sure the stepper motor functions correctly and how to set a certain rotation while monitoring the PID feedback mechanism. The feedback control will be the biggest challenge but is also the most important feature of this project.

We have tested the L298N stepper motor driver and have decided that we will proceed with using the Easy Driver by Brian Schmalz. We had trouble with getting the L298N motor driver to rotate in the reverse direction. The main feature of the Easy Driver is the simple inputs needed to control the stepper motor. As we have read, we will only need two main pins in order to control the direction of rotation and the step size. The power supply voltage is rated between 6V to 30V, and the adjustable current is from 150mA/phase to 750mA/phase.

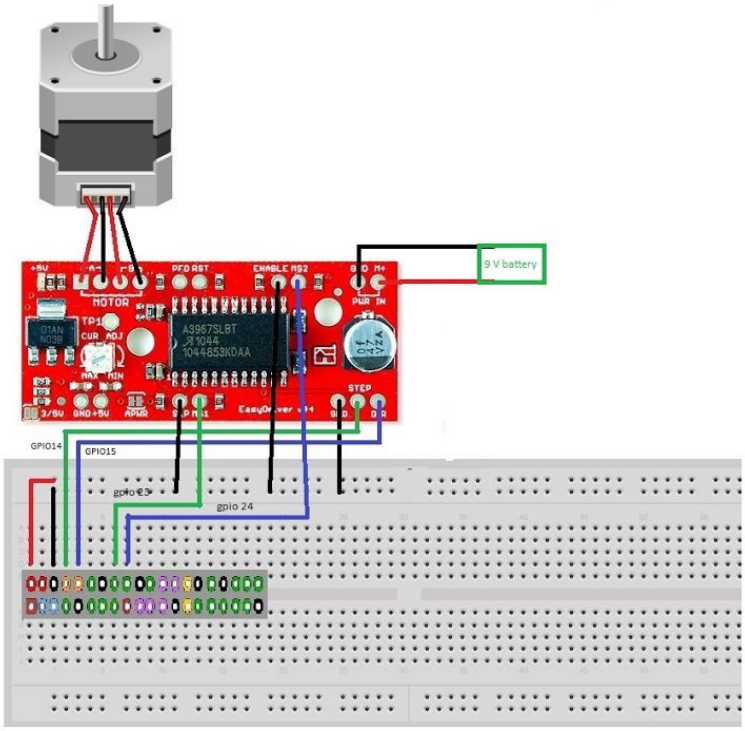
The schematic of the Easy Driver can be seen below.



The main pins for control are STEP and DIR, which dictate the direction of rotation and direction based on whether high or low inputs are set to those pins.



An example of the connections between the raspberry pi and the Easy Driver can be seen below.

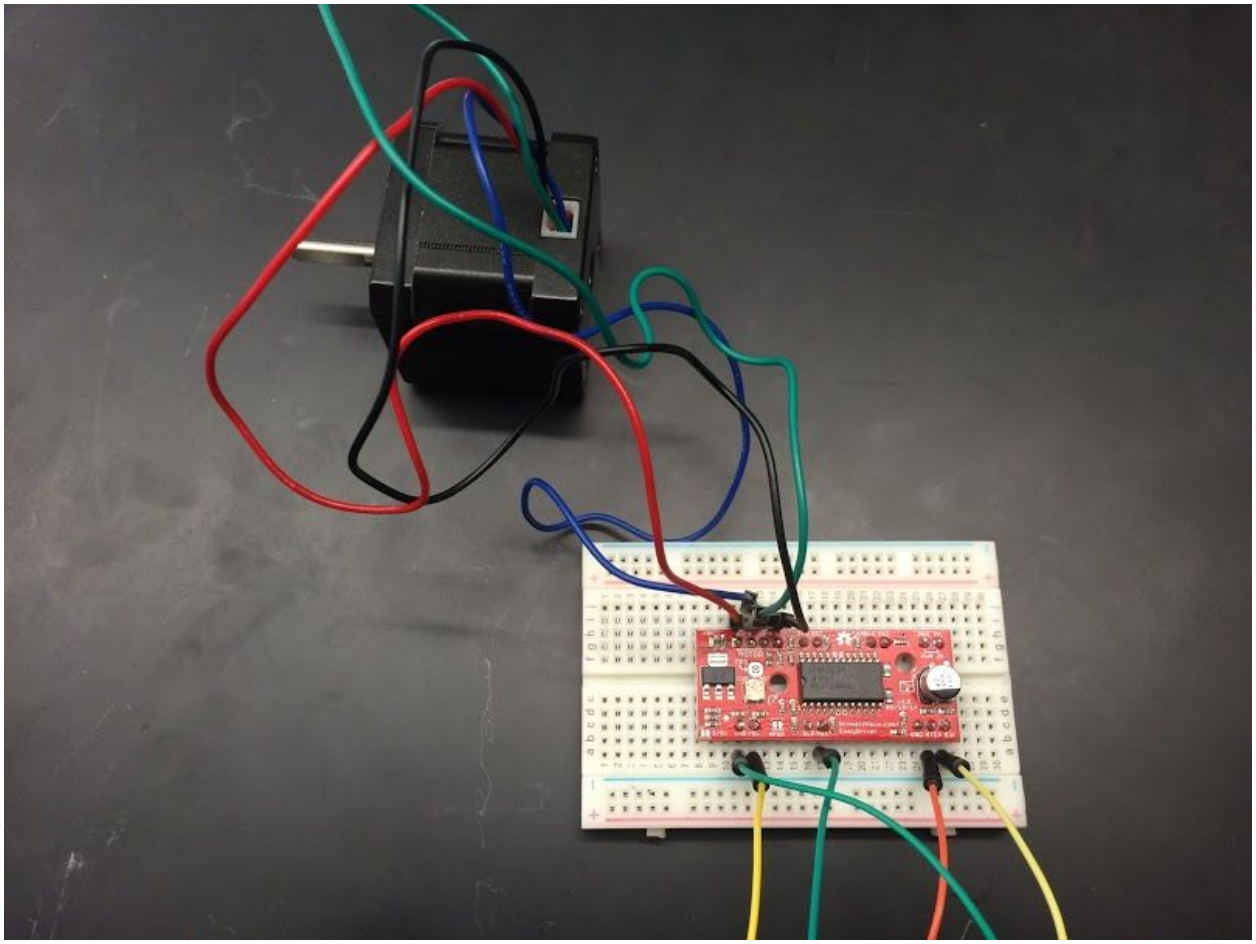


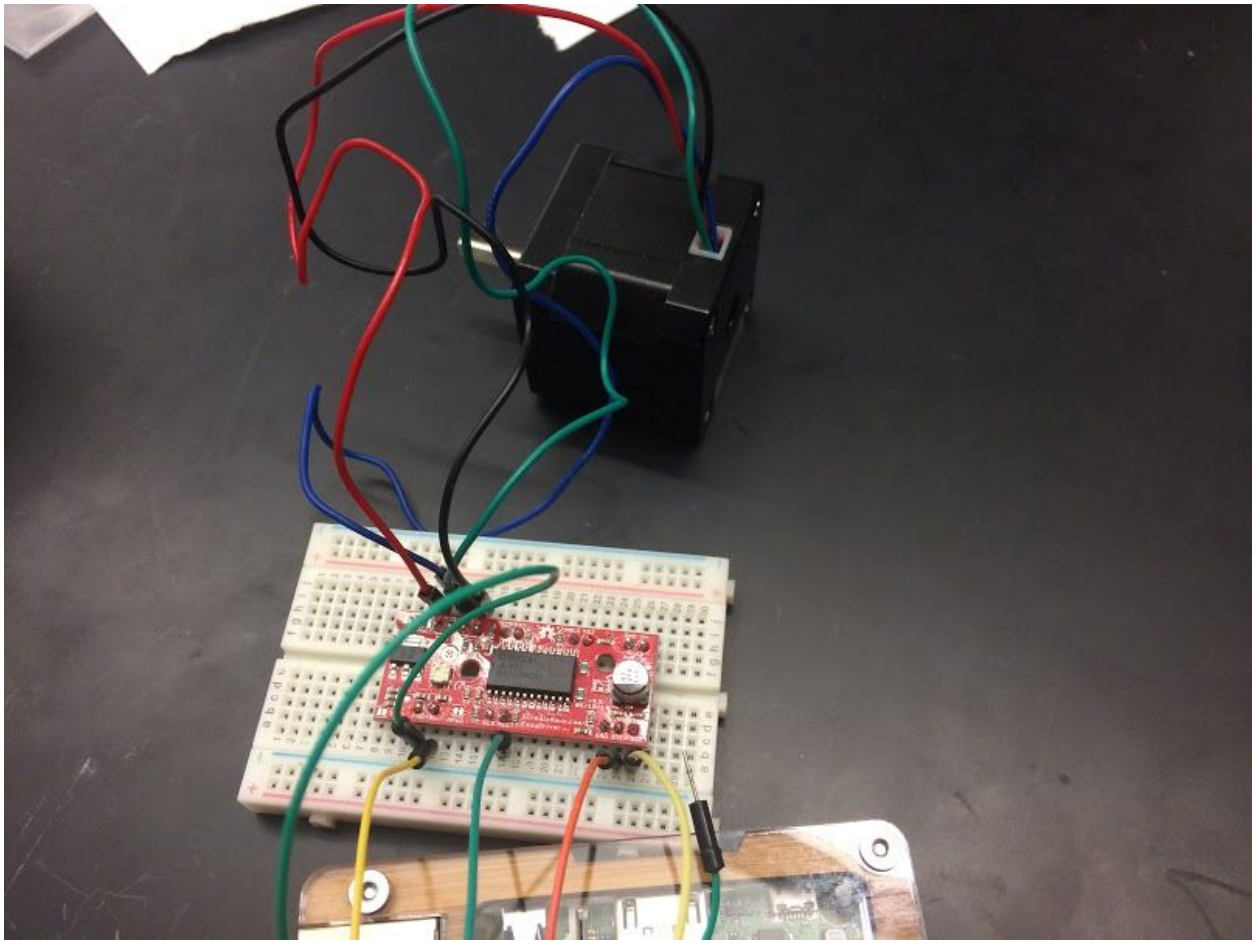
It would be possible to use both the GPIO and WiringPi libraries to control the easy driver. We will use the following code to test the easy driver.

```
////////////////////////////////////////////////////////////////////////////////

import RPi.GPIO as gpio
import time
import sys

#Broadcom layout for GPIO
gpio.setmode(gpio.BCM)

#Step Pin
gpio.setup(14, gpio.OUT)
#Direction Pin (Clockwise / Counter-Clockwise)
gpio.setup(15, gpio.OUT)
#microstep 1
gpio.setup(23, gpio.OUT)
#microstep 2
gpio.setup(24, gpio.OUT)

size = 360
```

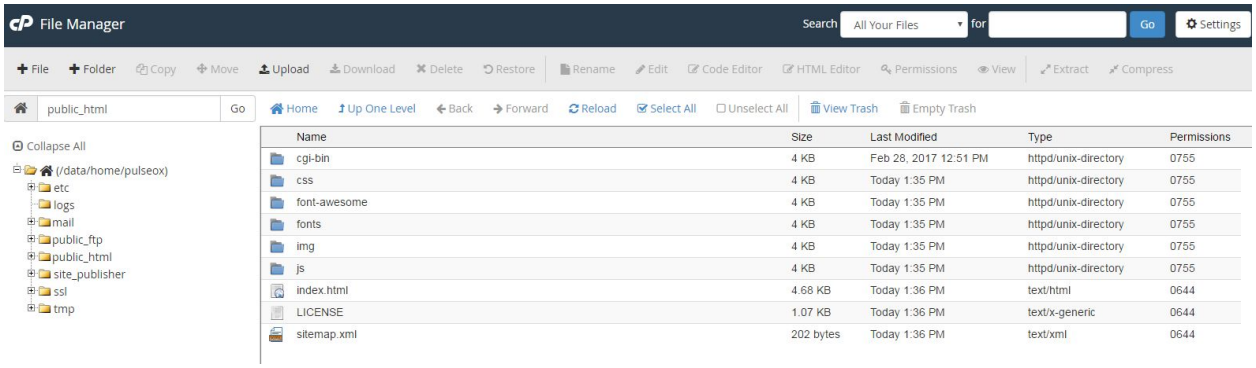
```
def step(direction, size):
stepCounter = 0
#step clockwise
gpio.setup(15, 1)
while (stepCounter < size)
#Turning DIR on/off tells motor to take one step
gpio.output(14, 0)
gpio.output(14, 1)
stepCounter += 1

def main():
#rotate clockwise
step(0, size)
#rotate counter-clockwise
step(1, size)
```

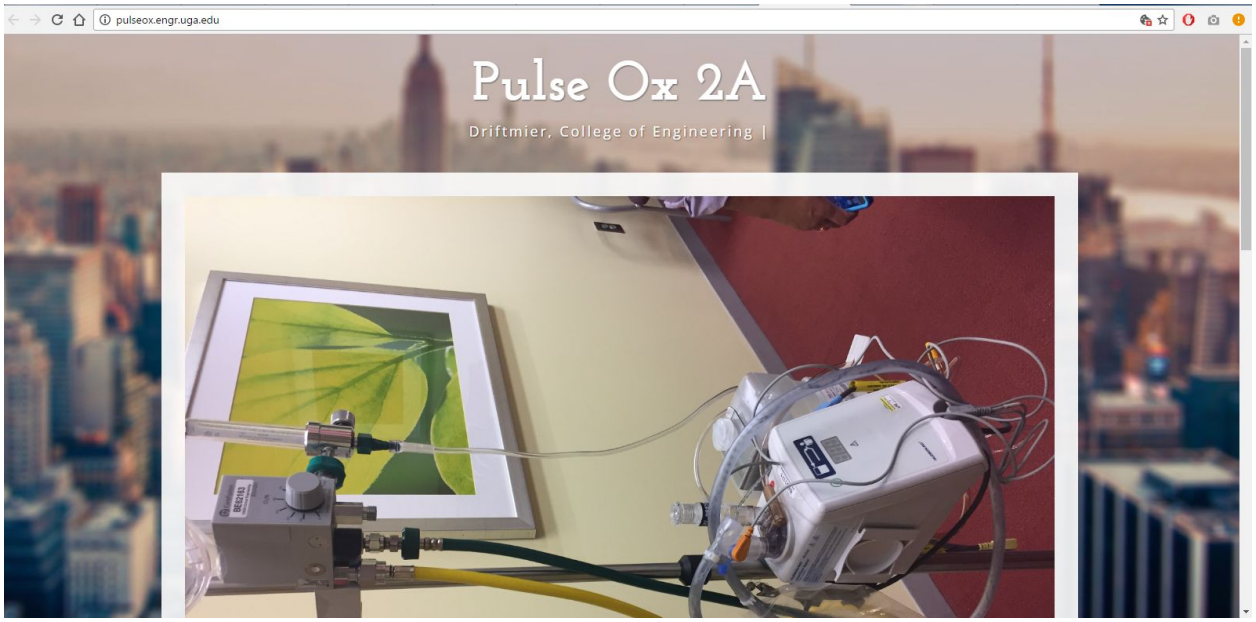
//



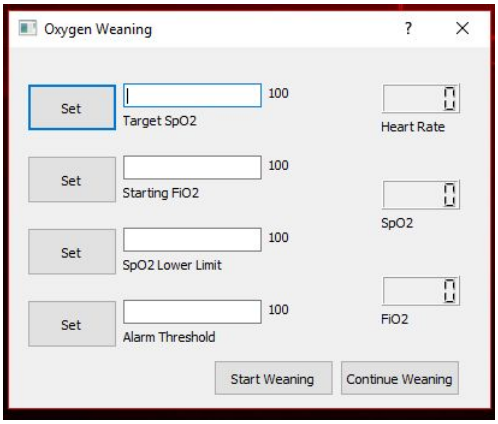


We have also received a temporary domain on the College of Engineering server to test a website and store a database onto. It will be located at pulseox.engr.uga.edu. Specifically we will try to use phpMyAdmin and MySQL to upload data points from the pulse oximeter to record data over time and possibly perform analysis on the data.




pulseox.engr.uga.edu
Pulse Ox 2A
Driftmier, College of Engineering |

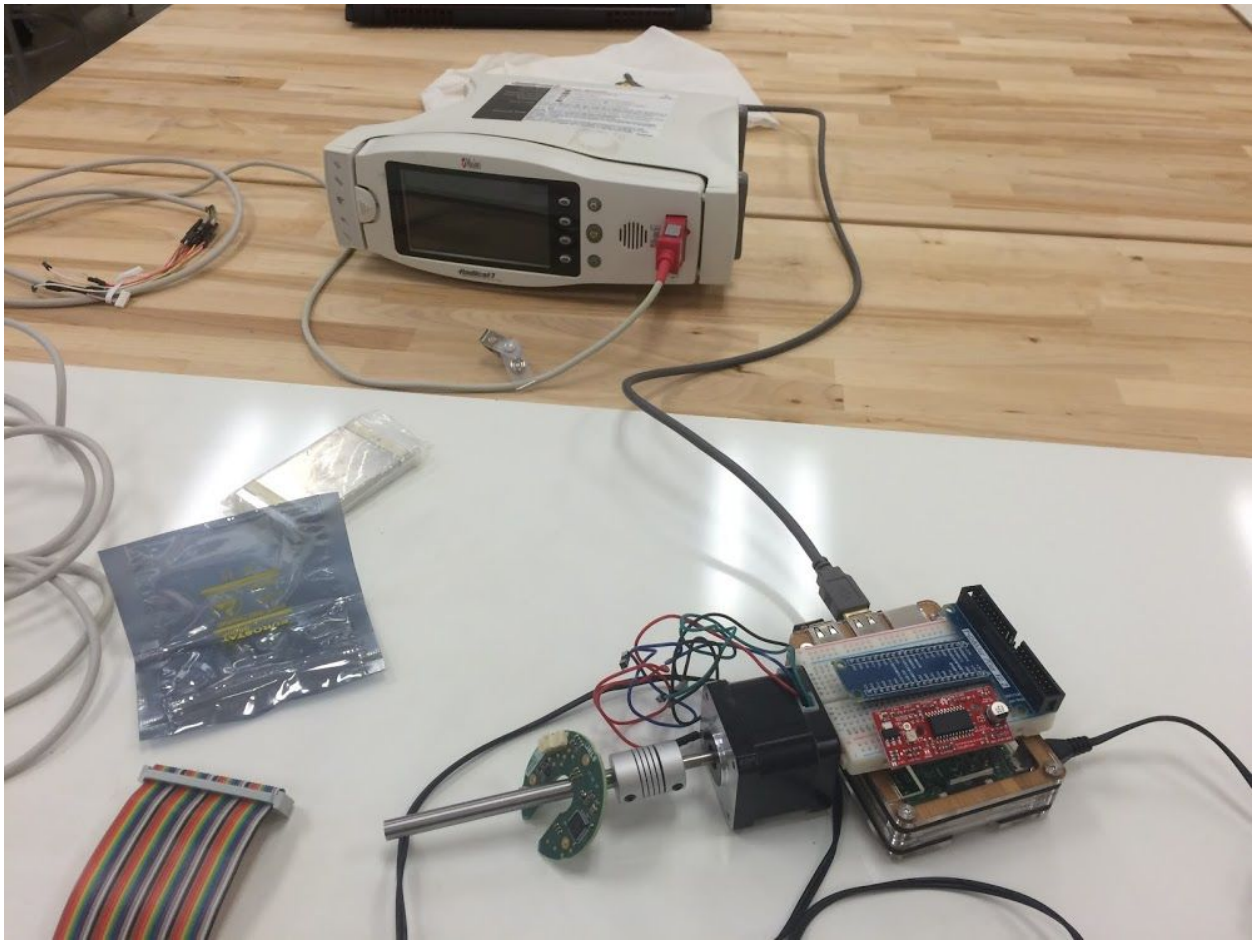
User Interface

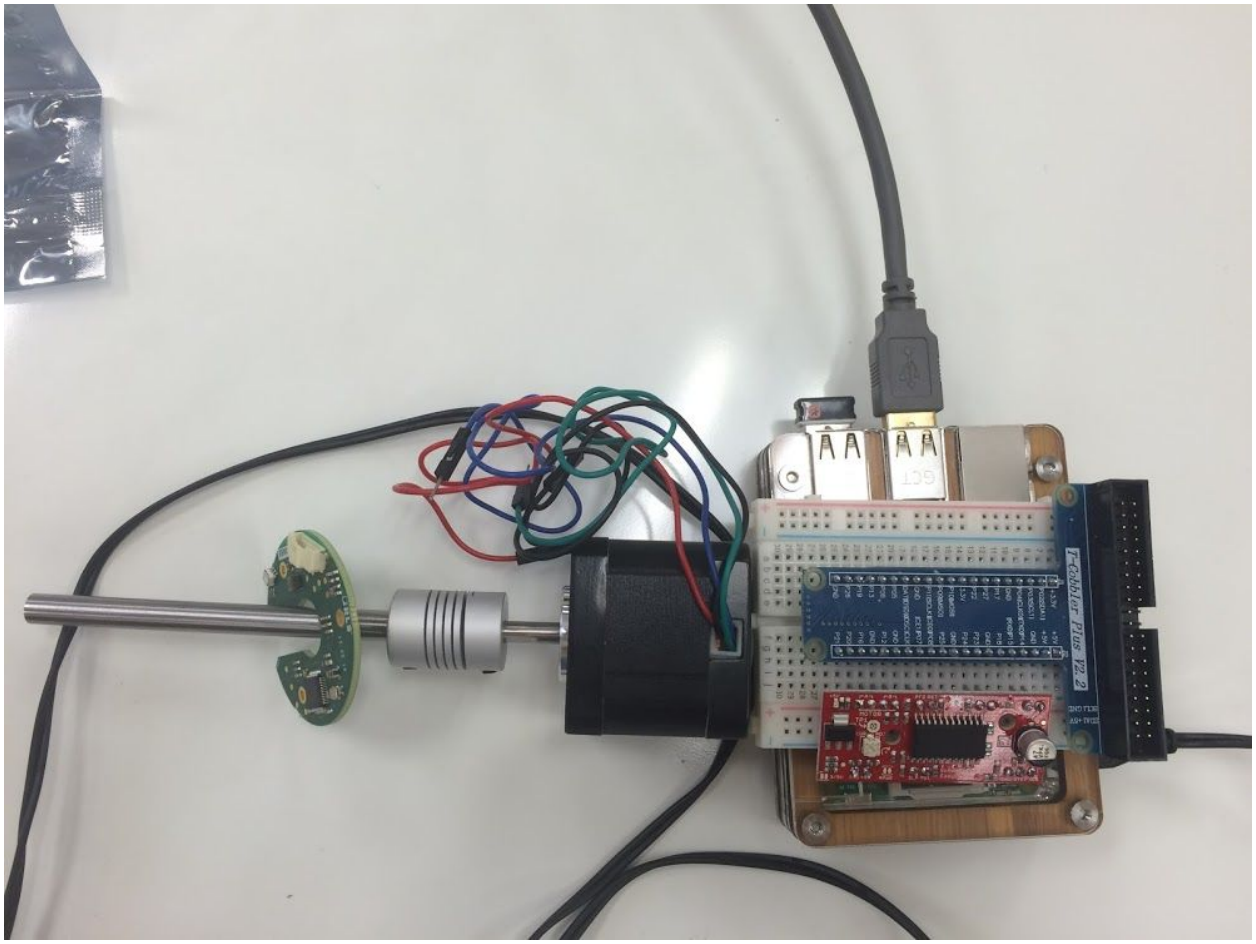


The GUI was created using a python module called PyQt. This module was chosen over the base TkInter module because it was more aesthetically pleasing. The TkInter module looked like it was developed for Windows 95. Also, the TkInter module did not have a good way, other than just an undistinguished text box, to display the current heart rate, SpO2, and FiO2 while PyQt had faux 7-segment LED displays. Programming the GUI was tedious without a development environment. Without the PyQt designer, creating the GUI boiled down to instantiating an object such as a textbox or button and continually editing and running the program until the size and location looked correct.

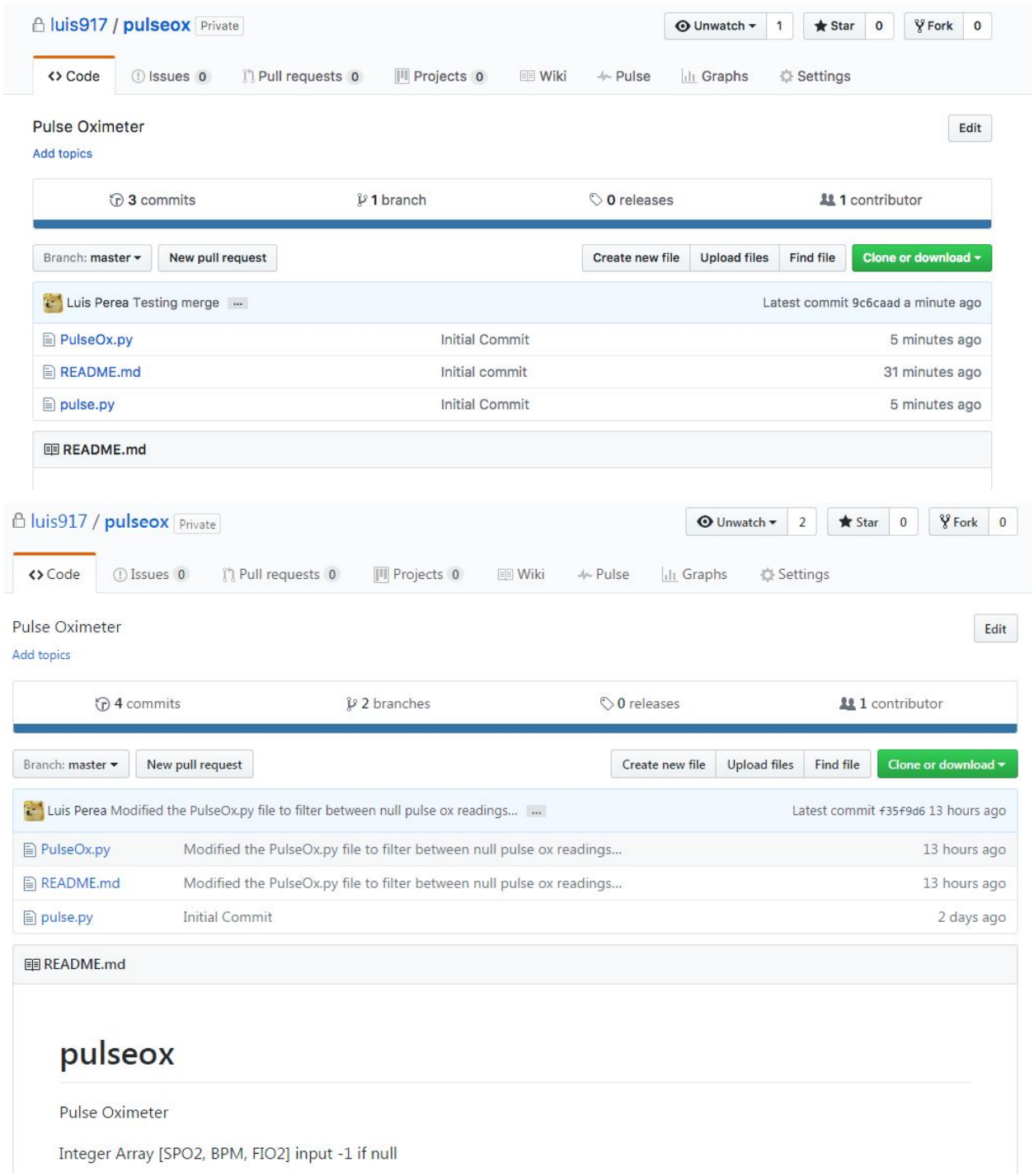
Prototype

We have been working on developing the code for the system since the physical connections between the devices should not be too difficult to implement. To have an idea of the physical dimensions, we placed all the components together to see how they would fit in the box that needs to be 3D printed to hold all of the internal components.





For the code, we have set up a private git repository at https://github.com/luis917/pulseox. Since it is private the code will not be visible to the public. We have added multiple teammates as collaborators to this repository so that we will be able to make changes to the code remotely. We have had difficulties with python because there are basically no global variables in this programming language which makes it difficult to share data between functions or “classes”.



The code below is the first iteration of the pulseox.py script, which should open the serial monitor for reading from the pulse oximeter. The pulse oximeter ouputs ASCII characters. If there is no reading detected, it will output a string such as: “SPO2 = ---%”. If there is a reading the pulse oximeter will output a string such as: “SPO2 = 098%”. This code should begin a forever-loop that continuously reads from the pulse oximeter. It will first create a substring to extract the string that will either represent a number or the “---”. It will check if the string matches

the “---”, and if it is, it will return a -1 and print null. If there is a string representation of a number, it will cast the string into an integer, and place the integer into an array. The array will consist of: array = [SPO2, BPM], where SPO2 is the oxygen concentration, and BPM is the beats per minute. This array will be returned to the main function.

//

```
#!=/usr/bin/python
import serial

#set up Serial Monitor at TTYUSB0

#serial_monitor = serial.Serial('dev/ttyUSB0', 9600, 8, 'N', 1, timeout =
1)

SP02 = 0
FIO2 = 0
BPM = 0


def read_pulse_ox(serial_monitor):

        #try:

        line = serial_monitor.readline()
        words = line.split()

        print(words[0]) # Date
        print(words[1]) # Time
        '''
        print(words[3]) # SPO2
        print(words[4]) # BPM
        '''
        dash = '---'
        SPO2_string = words[3] #SPO2
        if(dash in SPO2_string):
                SPO2_string = "null"
                print("SPO2 = " + SPO2_string)
                SPO2 = -1
        else:
                SPO2_string = SPO2_string[5:8]
                #global SPO2
                SPO2 = int(SPO2_string)
```

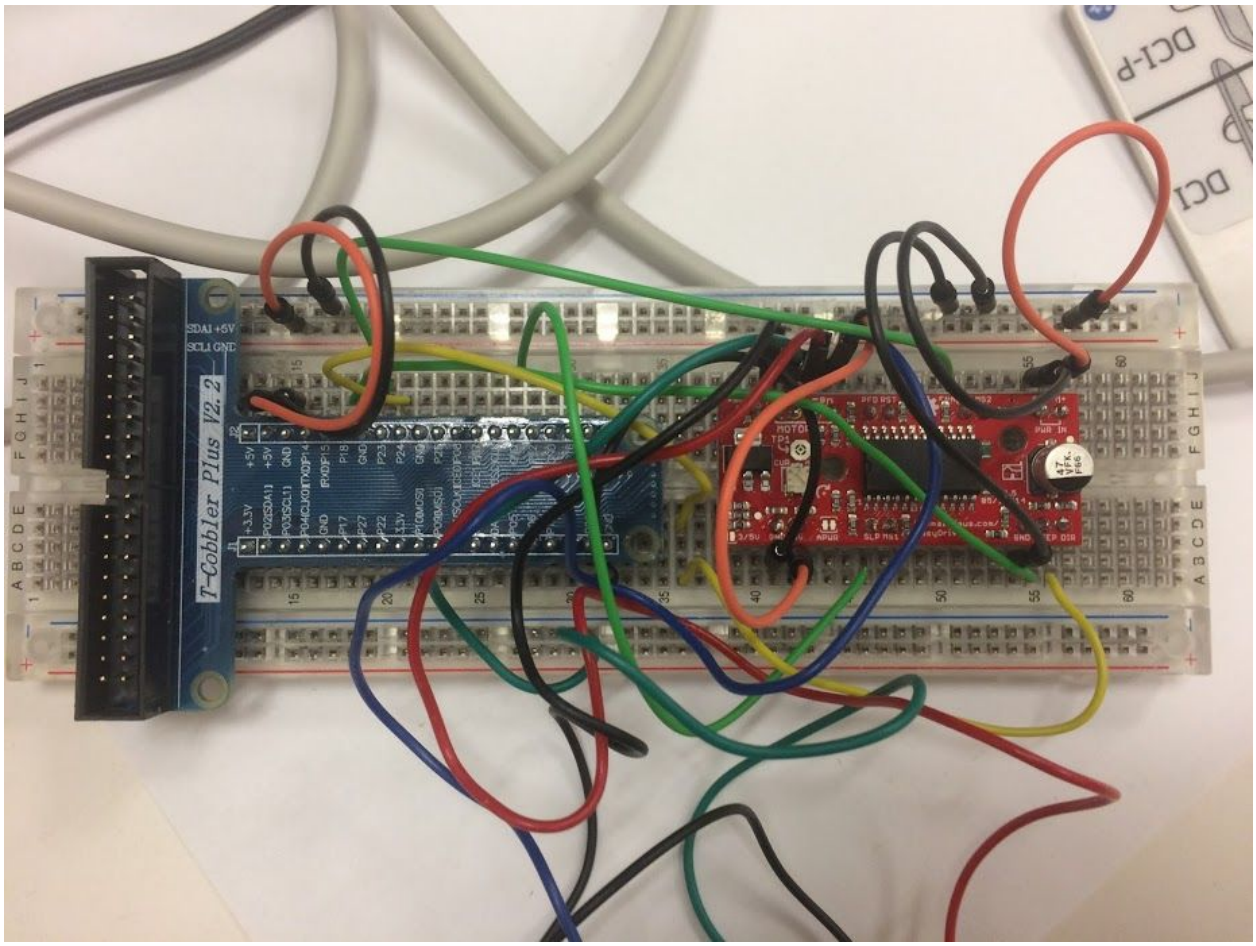
```
                print(SPO2)

        BPM_string = words[4] #BPM
        if(dash in BPM_string):
                BPM_string = "null"
                BPM = -1
                print("BPM = " + BPM_string)
        else:
                BPM_string = BPM_string[4:7]
                BPM = int(BPM_string)
                print(BPM)

        print("\n")
        array = [SPO2, BPM]
        return array
        #print(SPO2)
        #print(BPM)

        #except:
            #pass
    #serial_monitor.close()



def main():
      #set up Serial Monitor at TTYUSB0
      serial_monitor = serial.Serial(
      port = '/dev/ttyUSB0',\
      baudrate = 9600,\
      parity = serial.PARITY_NONE,\
      stopbits = serial.STOPBITS_ONE,\
      bytesize = serial.EIGHTBITS,\
      timeout = 1)

      print("Connected to: " + serial_monitor.portstr)

        while True:
               array = read_pulse_ox(serial_monitor)
               print(str(array[0]) + " " + str(array[1]))
        serial_monitor.close()
```

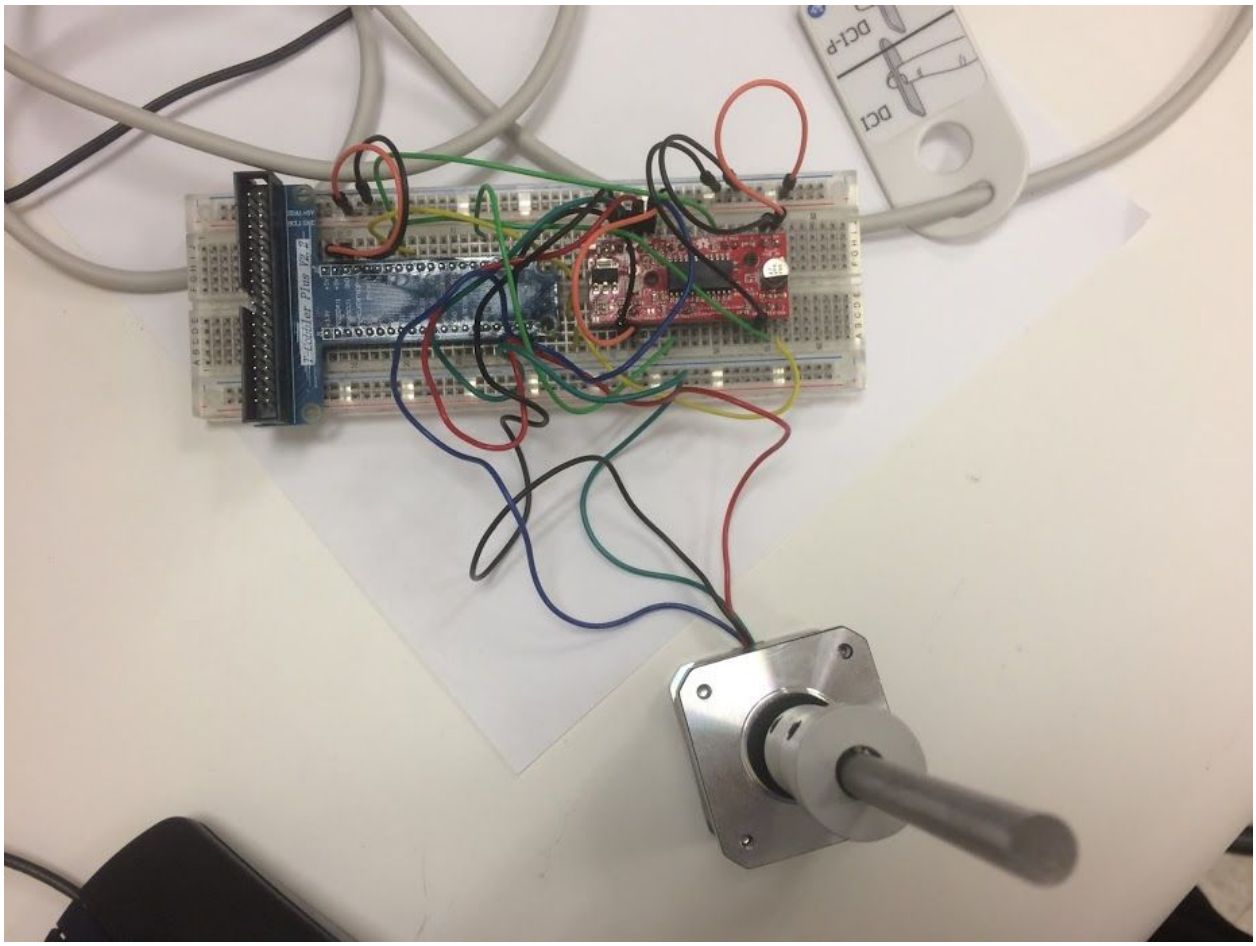
```
if __name__ == "__main__": main()
```

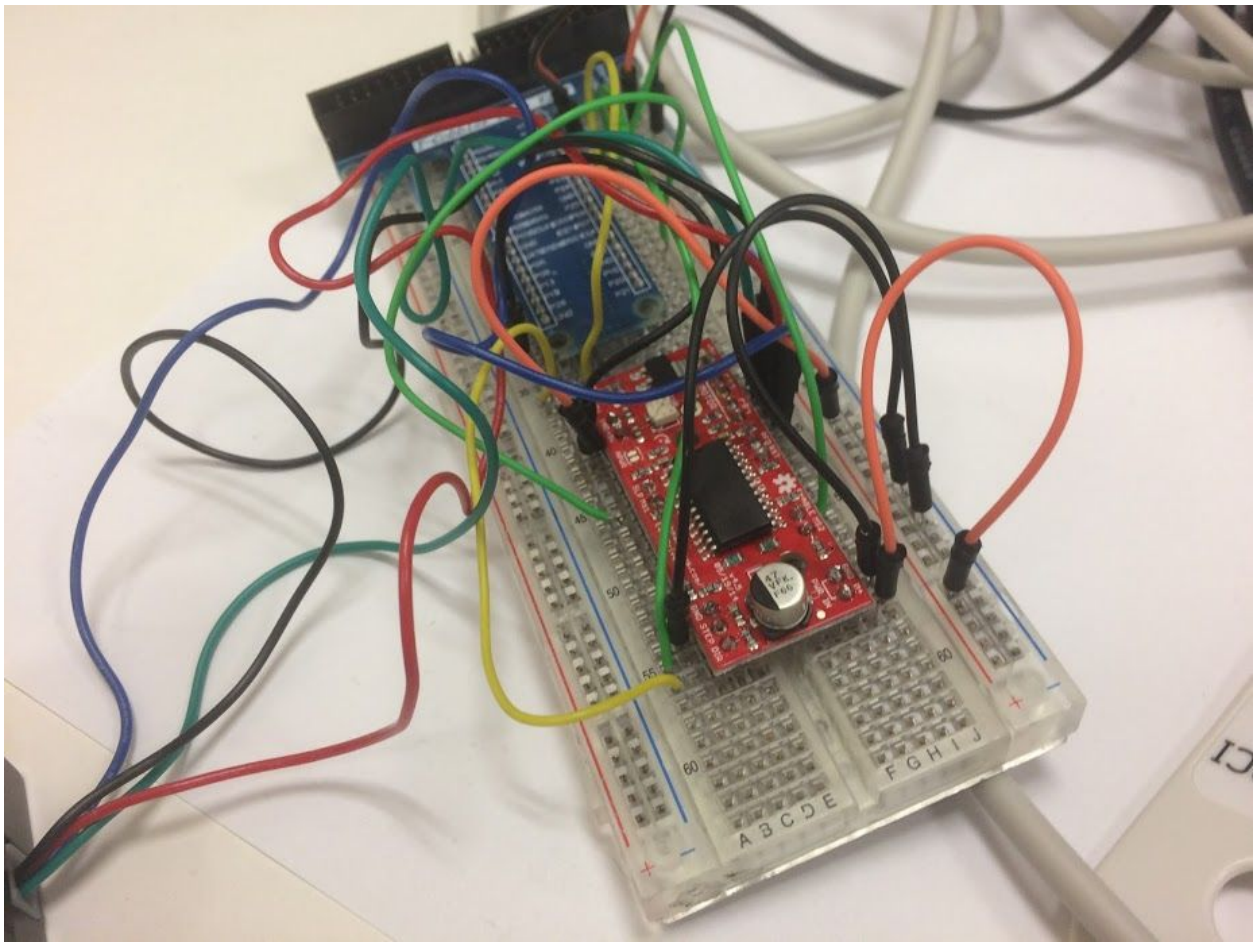
//

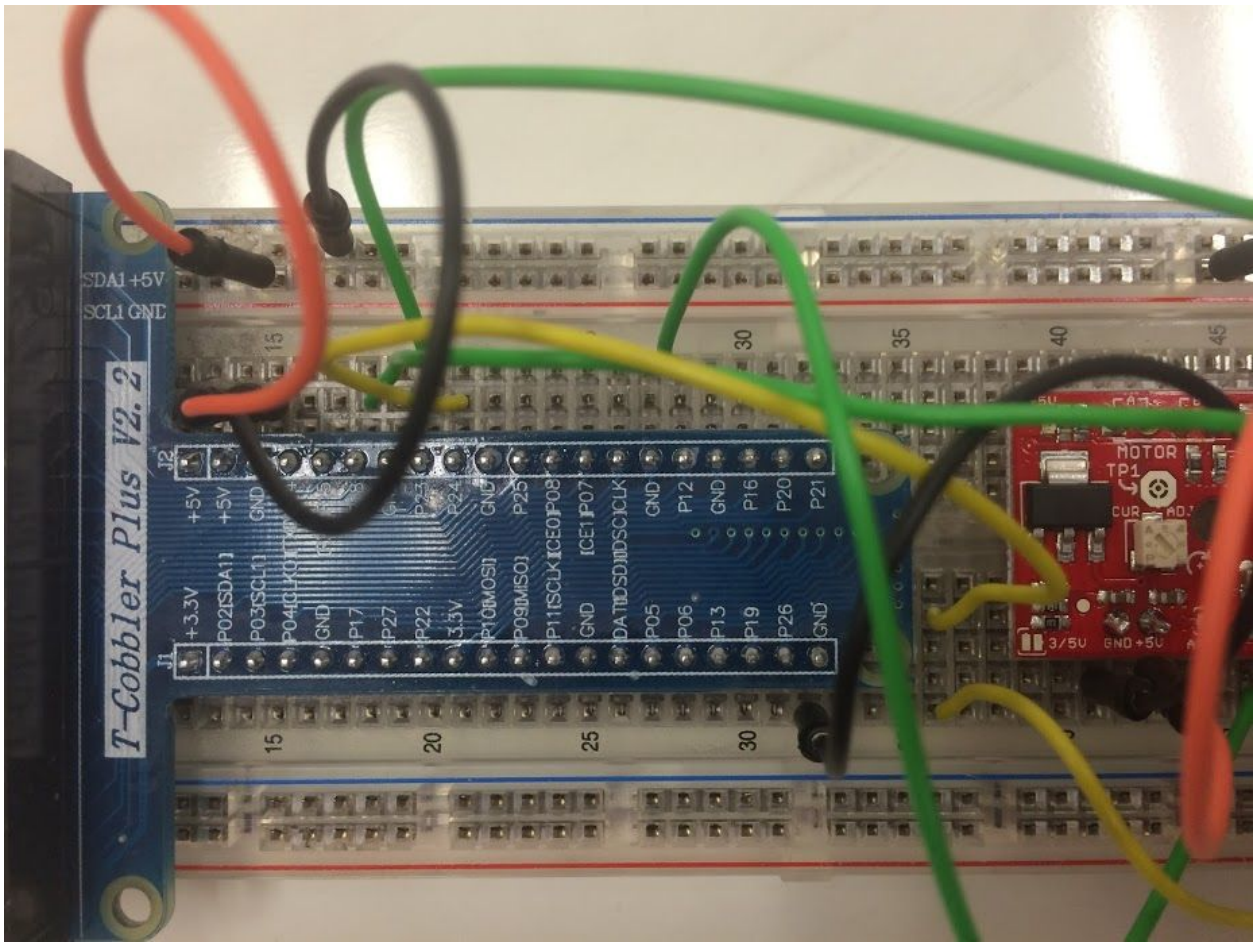
The output of the code is shown in the figure below.

The near-final circuit can be seen below.




DCI-P
DCI


A B C D E
F G H I J


T-Cobbler Plus V2.2
SDA1 +5V
SCL1 GND
MOTOR
TP1
CUR
ADJ
3/5V
GND +5V

The connections for the easy driver can be seen in the table below.

Easy Driver	Raspberry Pi
5V	5V
I1	5V
GND	GND
Enable	Pin 23
DIR	Pin 24
SLP	Pin 12
A1	(+A) black motor wire
A2	(-A) green motor wire
B1	(+B) red motor wire
B2	(-B) blue motor wire

The end of the stepper motor should be attached to a 5mm to 7mm coupler, which on the other side is attached to a 7mm rod. The end of the rod should be attached to the 3D-printed knob cap which should be attached over the oxygen blender knob, set to 100% FIO2 to begin with.

The connections for the encoder can be seen below.

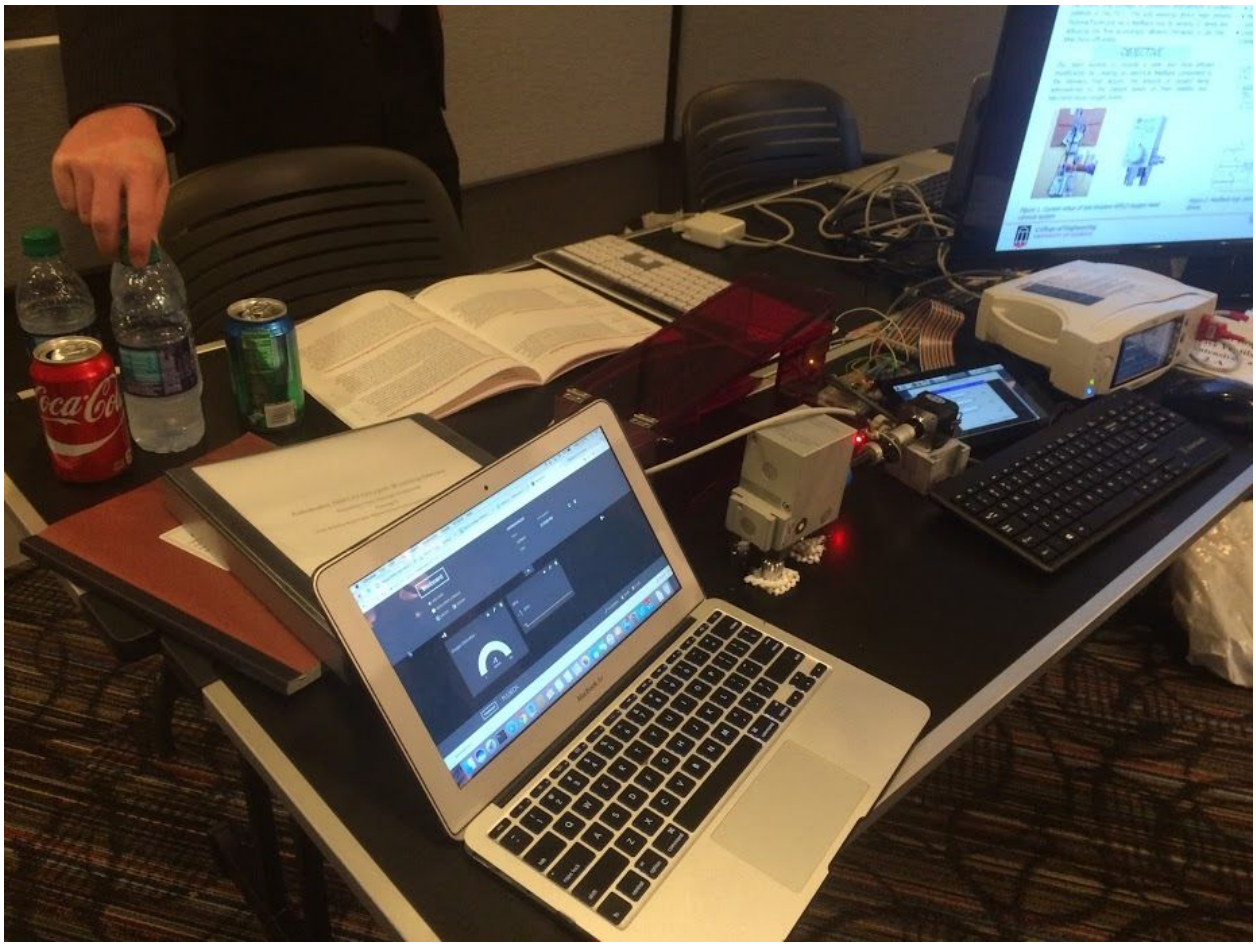
Encoder	Raspberry Pi
5V	5V
White	GND
RED / SCK	SCK
BLUE / CS	CE0
GREEN / MISO	Pin 9
YELLOW / MOSI	Pin 10

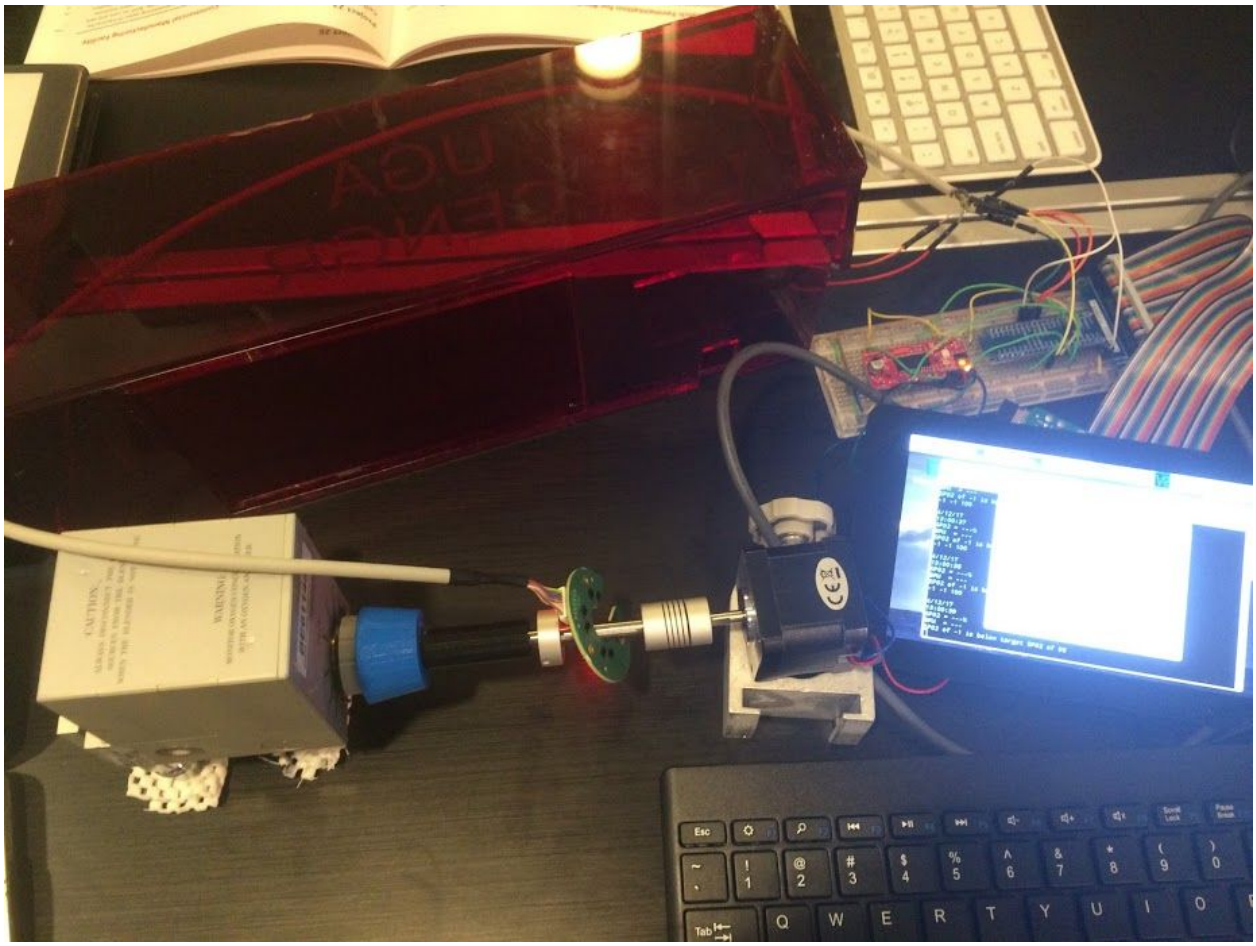
The connections for the pulse oximeter should be as follows:
Connect the Masimo Pulse Oximeter to the Masimo Radical 7 patient monitor using the Red LNCS 20-pin Patient Cable. The Radical 7 must be set to output data in ASCII-1. Using a RS-232 to USB cable, connect the Radical 7 to a usb port on a raspberry pi. To verify the data is valid, type the command: “stty -F /dev/ttyUSB0 9600 cs8 -parenb -cstopb -crtscts”, into the terminal, without the quotation marks.

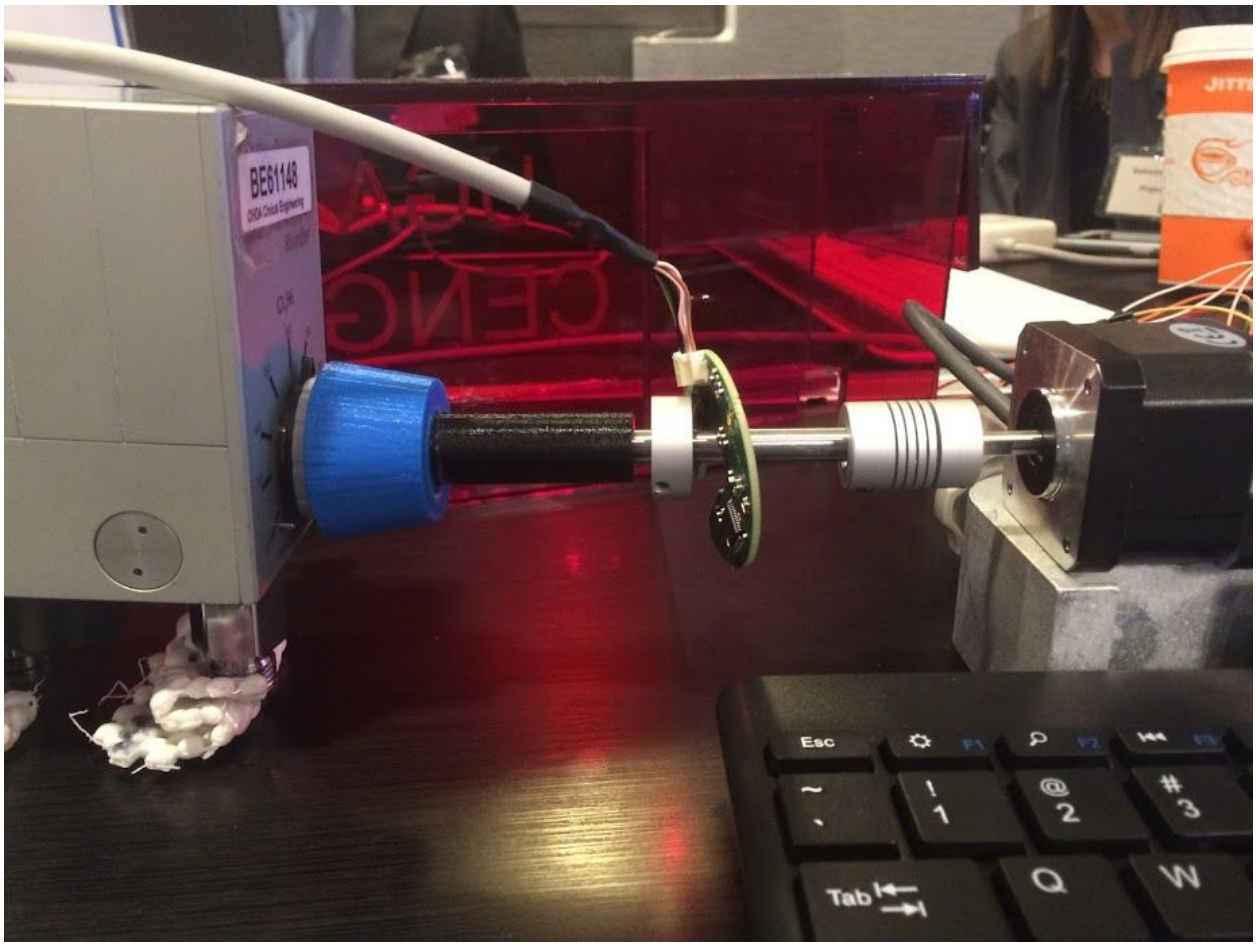
To run the program, the “pulse_oximeter.py” file should be placed on the desktop of the raspberry pi. Navigate to the file from the terminal using “~/Desktop”, and type “python3 pulse_oximeter.py” to start the program, or double-click the file to start the program.

Photos from the presentation can be seen below.

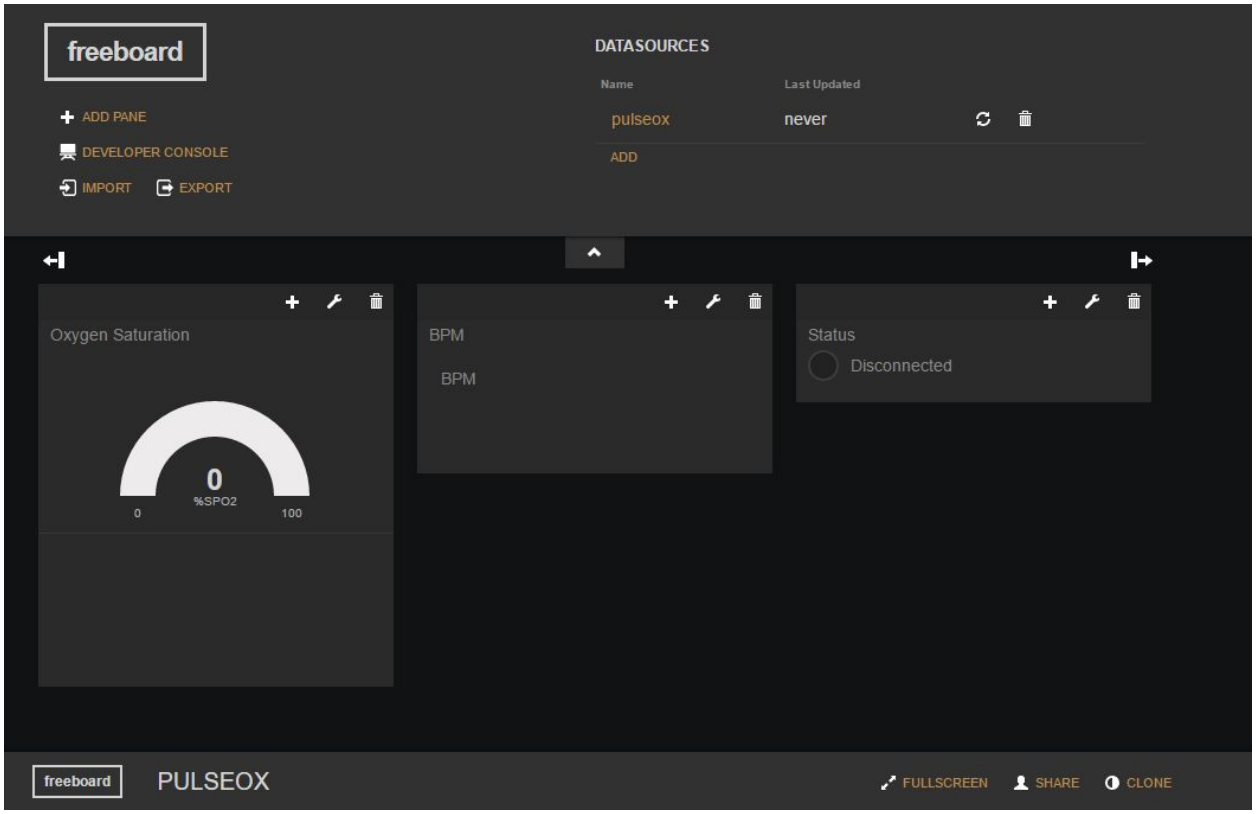

UGA
CENGR

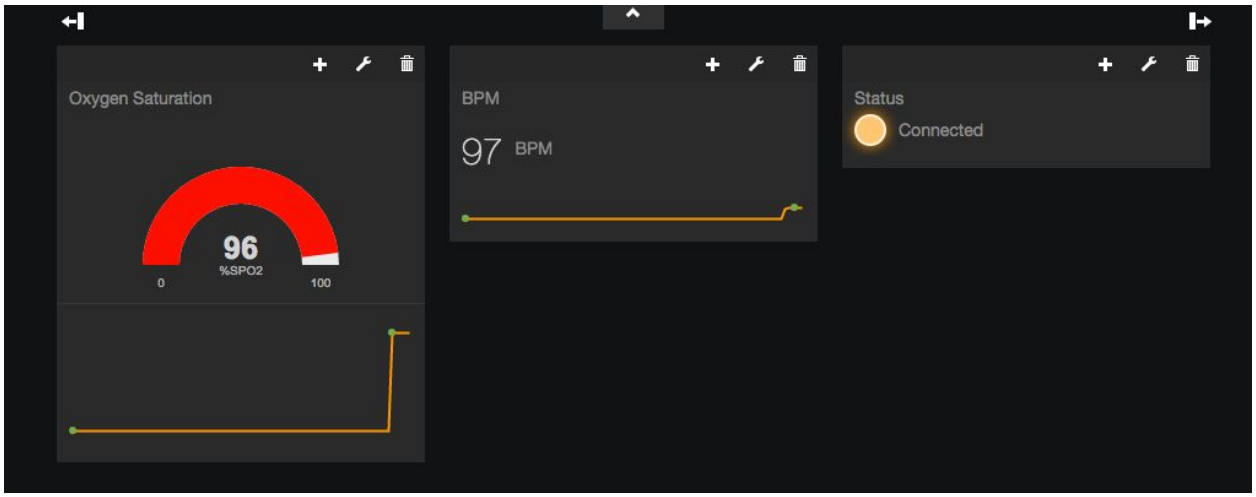


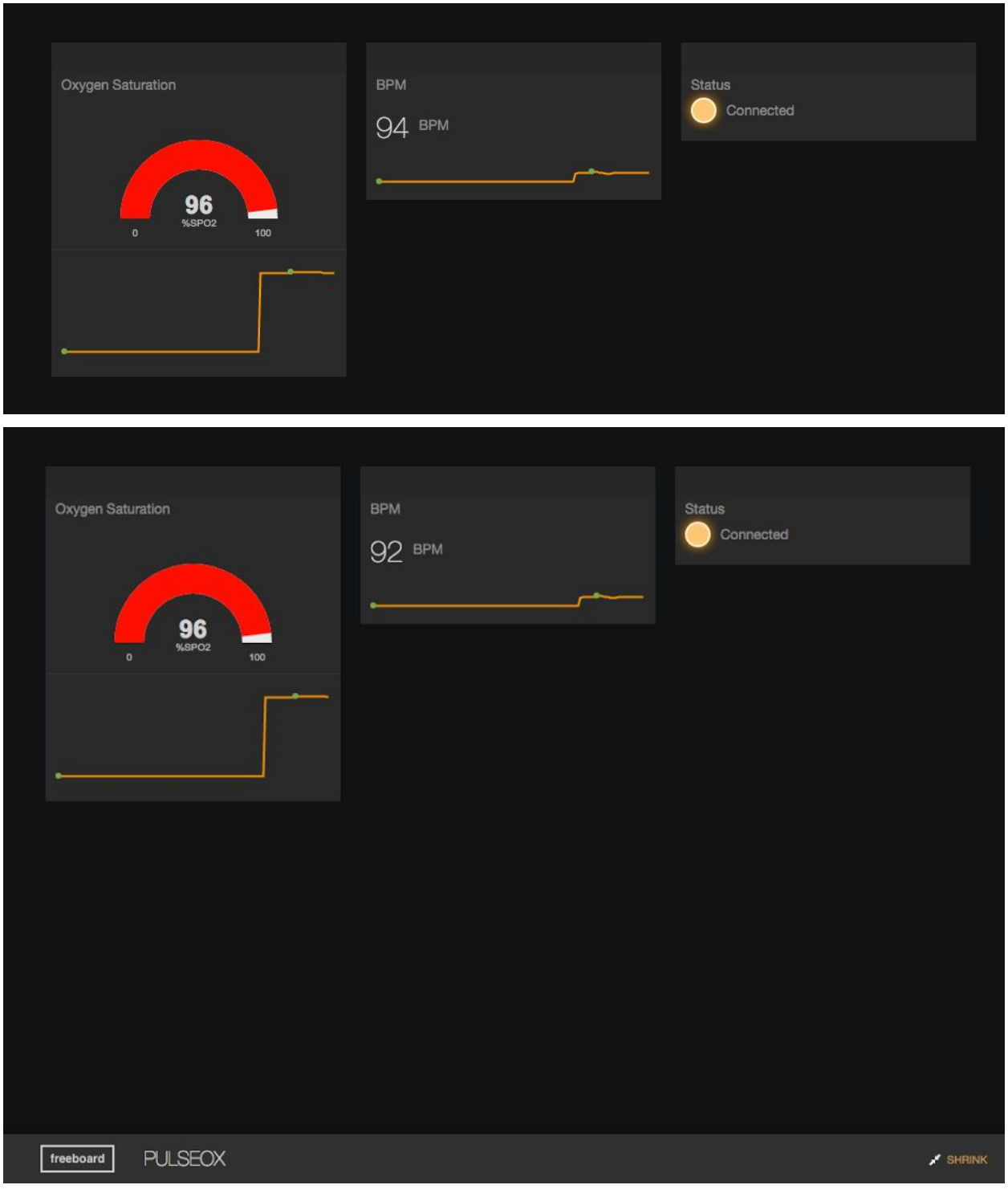

Esc
Tab
Q W E R T Y U I O

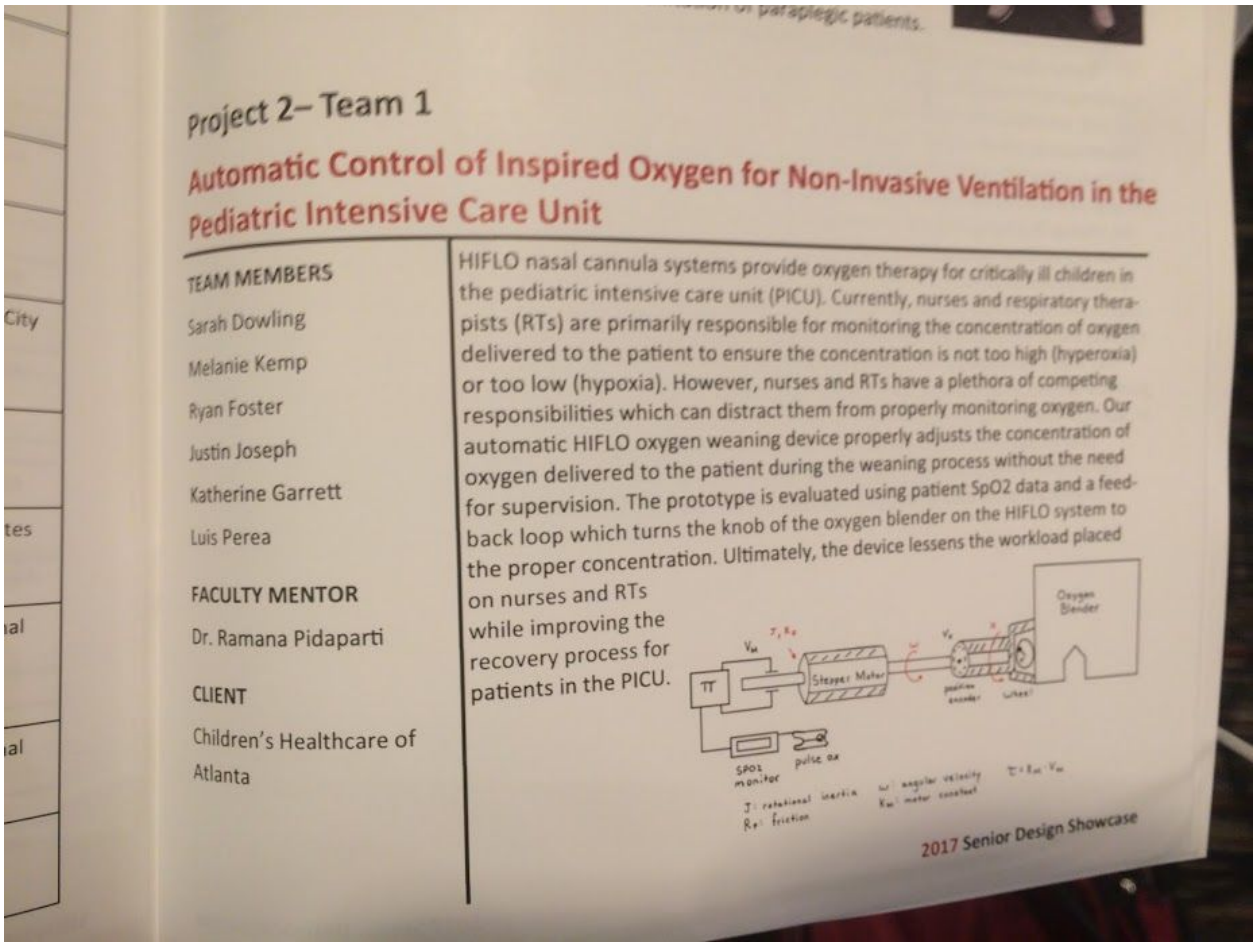


We also created a Dweet.IO server to view the data in real-time on the Freeboard platform. This is the beginning of a concept for an Internet-of-Things device which would allow multiple sensors to upload data to a central database. This would allow for remote monitoring of patients and potentially changing the settings of the device remotely itself. For the example below, there are three main widgets. The first widget on the left shows current SPO2 as a percentage between 0 to 100. The second widget shows the beats-per-minute of the patient, as well as a waveform of the BPM over time. The third widget shows the connectivity status of the device to the database. This widgets could be downloaded as a json file to be embedded in a website if needed.


freeboard
ADD PANE
DEVELOPER CONSOLE
IMPORT
EXPORT
DATASOURCES
Name
Last Updated
pulseox
never
ADD
Oxygen Saturation
0
%SPO2
0
100
BPM
BPM
Status
Disconnected
freeboard
PULSEOX
FULLSCREEN
SHARE
CLONE


Oxygen Saturation
96
%SPO2
0
100
BPM
97 BPM
Status
Connected


Oxygen Saturation
96
%SPO2
0
100
BPM
94 BPM
Status
Connected
Oxygen Saturation
96
%SPO2
0
100
BPM
92 BPM
Status
Connected
freeboard
PULSEOX
SHRINK



The code that was used in the presentation can be seen below.

```
from PyQt5 import QtCore, QtGui, QtWidgets
import time
import serial
import sys
import signal
import RPi.GPIO as GPIO
import os
import requests

readIn = [0, 0, 0]

SPO2_lower_limit = 93
FIO2 = 100
#1 hour is 3600 seconds
hour = 5

#initial starting time the program was started.
startTime = time.time()

class Ui_Dialog(object):

    def continueWean(self, Dialog):
        shouldContinue = 1
        return shouldContinue

    def startWean(self, Dialog):
        shouldStart = 1
        return shouldStart

    def setSpO2(self, Dialog):
        spO2Value = self.lineEdit.text()
        self.spO2Target_label.setText(spO2Value)
        return spO2Value

    def setFiO2(self, Dialog):
        global readIn
        fiO2Value = self.lineEdit_2.text()
        self.setFiO2_label.setText(fiO2Value)
        readIn[2] = fiO2Value
        return fiO2Value

    def setSpO2Lower(self, Dialog):
```

```
        spO2Lower = self.lineEdit_3.text()
        self.spO2Lower_label.setText(spO2Lower)
        return spO2Lower

    def setupUi(self, Dialog):
        Dialog.setObjectName("Dialog")
        Dialog.resize(400, 300)
        icon = QtGui.QIcon()
        icon.addPixmap(QtGui.QPixmap("icon.png"), QtGui.QIcon.Normal,
QtGui.QIcon.Off)
        Dialog.setWindowIcon(icon)
        self.frame = QtWidgets.QFrame(Dialog)
        self.frame.setGeometry(QtCore.QRect(10, 10, 380, 280))
        self.frame.setFrameShape(QtWidgets.QFrame.StyledPanel)
        self.frame.setFrameShadow(QtWidgets.QFrame.Raised)
        self.frame.setObjectName("frame")
        self.heartRate_LCD = QtWidgets.QLCDNumber(self.frame)
        self.heartRate_LCD.setGeometry(QtCore.QRect(300, 20, 65, 25))
        self.heartRate_LCD.setObjectName("heartRate_LCD")
        self.spO2_LCD = QtWidgets.QLCDNumber(self.frame)
        self.spO2_LCD.setGeometry(QtCore.QRect(300, 100, 65, 25))
        self.spO2_LCD.setObjectName("spO2_LCD")
        self.label = QtWidgets.QLabel(self.frame)
        self.label.setGeometry(QtCore.QRect(300, 45, 60, 20))
        self.label.setObjectName("label")
        self.label_2 = QtWidgets.QLabel(self.frame)
        self.label_2.setGeometry(QtCore.QRect(300, 125, 60, 20))
        self.label_2.setObjectName("label_2")
        self.fiO2_LCD = QtWidgets.QLCDNumber(self.frame)
        self.fiO2_LCD.setGeometry(QtCore.QRect(300, 180, 65, 25))
        self.fiO2_LCD.setObjectName("fiO2_LCD")
        self.label_3 = QtWidgets.QLabel(self.frame)
        self.label_3.setGeometry(QtCore.QRect(300, 205, 60, 20))
        self.label_3.setObjectName("label_3")
        self.lineEdit = QtWidgets.QLineEdit(self.frame)
        self.lineEdit.setGeometry(QtCore.QRect(85, 20, 115, 20))
        self.lineEdit.setObjectName("lineEdit")
        self.label_4 = QtWidgets.QLabel(self.frame)
        self.label_4.setGeometry(QtCore.QRect(85, 40, 115, 20))
        self.label_4.setObjectName("label_4")
        self.spO2Target_label = QtWidgets.QLabel(self.frame)
        self.spO2Target_label.setGeometry(QtCore.QRect(205, 20, 30, 15))
```

```
        self.spO2Target_label.setObjectName("spO2Target_label")
        self.setSpO2_button = QtWidgets.QPushButton(self.frame)
        self.setSpO2_button.setGeometry(QtCore.QRect(5, 20, 75, 40))
        self.setSpO2_button.setObjectName("setSpO2_button")
        self.setFiO2_label = QtWidgets.QLabel(self.frame)
        self.setFiO2_label.setGeometry(QtCore.QRect(205, 80, 30, 15))
        self.setFiO2_label.setObjectName("setFiO2_label")
        self.label_7 = QtWidgets.QLabel(self.frame)
        self.label_7.setGeometry(QtCore.QRect(85, 100, 115, 20))
        self.label_7.setObjectName("label_7")
        self.setFiO2_button = QtWidgets.QPushButton(self.frame)
        self.setFiO2_button.setGeometry(QtCore.QRect(5, 80, 75, 40))
        self.setFiO2_button.setObjectName("setFiO2_button")
        self.lineEdit_2 = QtWidgets.QLineEdit(self.frame)
        self.lineEdit_2.setGeometry(QtCore.QRect(85, 80, 115, 20))
        self.lineEdit_2.setObjectName("lineEdit_2")
        self.continueButton = QtWidgets.QPushButton(self.frame)
        self.continueButton.setGeometry(QtCore.QRect(265, 250, 100, 30))
        self.continueButton.setObjectName("continueButton")
        self.label_8 = QtWidgets.QLabel(self.frame)
        self.label_8.setGeometry(QtCore.QRect(85, 160, 115, 20))
        self.label_8.setObjectName("label_8")
        self.setSpO2Lower_button = QtWidgets.QPushButton(self.frame)
        self.setSpO2Lower_button.setGeometry(QtCore.QRect(5, 140, 75, 40))
        self.setSpO2Lower_button.setObjectName("setSpO2Lower_button")
        self.spO2Lower_label = QtWidgets.QLabel(self.frame)
        self.spO2Lower_label.setGeometry(QtCore.QRect(205, 140, 30, 15))
        self.spO2Lower_label.setObjectName("spO2Lower_label")
        self.lineEdit_3 = QtWidgets.QLineEdit(self.frame)
        self.lineEdit_3.setGeometry(QtCore.QRect(85, 140, 115, 20))
        self.lineEdit_3.setObjectName("lineEdit_3")
        self.startButton = QtWidgets.QPushButton(self.frame)
        self.startButton.setGeometry(QtCore.QRect(160, 250, 100, 30))
        self.startButton.setObjectName("startButton")

        self.retranslateUi(Dialog)
        QtCore.QMetaObject.connectSlotsByName(Dialog)
        self.setSpO2_button.clicked.connect(self.setSpO2)
        self.setFiO2_button.clicked.connect(self.setFiO2)
        self.setSpO2Lower_button.clicked.connect(self.setSpO2Lower)
        self.startButton.clicked.connect(self.startWean)
        self.continueButton.clicked.connect(self.continueWean)
```

```
    def retranslateUi(self, Dialog):
        _translate = QtCore.QCoreApplication.translate
        Dialog.setWindowTitle(_translate("Dialog", "Oxygen Weaning"))
        self.label.setText(_translate("Dialog", "BPM"))
        self.label_2.setText(_translate("Dialog", "SpO2"))
        self.label_3.setText(_translate("Dialog", "FiO2"))
        self.label_4.setText(_translate("Dialog", "Target SpO2"))
        self.spO2Target_label.setText(_translate("Dialog", "100"))
        self.setSpO2_button.setText(_translate("Dialog", "Set"))
        self.setFiO2_label.setText(_translate("Dialog", "100"))
        self.label_7.setText(_translate("Dialog", "Starting FiO2"))
        self.setFiO2_button.setText(_translate("Dialog", "Set"))
        self.continueButton.setText(_translate("Dialog", "Continue"))
        self.label_8.setText(_translate("Dialog", "SpO2 Lower Limit"))
        self.setSpO2Lower_button.setText(_translate("Dialog", "Set"))
        self.spO2Lower_label.setText(_translate("Dialog", "100"))
        self.startButton.setText(_translate("Dialog", "Start"))

class MultiThread(QtCore.QThread):

    def run (self):
        fiO2CurrentValue = 0
        spO2CurrentValue = 0
        heartRateCurrent = 0
        pulseox = PulseOx()
        global hour
        global FIO2
        global SPO2_lower_limit

        #initial starting time the program was started.
        startTime = time.time()

        while True:
            currentTime = time.time()
            if(currentTime - startTime > hour):
                FIO2 = pulseox.adjustFIO2()
                startTime = time.time()
            else:
                array = pulseox.read_pulse_ox()
                SPO2 = array[0]
                BPM = array[1]
```

```
                startTime = pulseox.process_SPO2(SPO2, SPO2_lower_limit,
startTime)

                dweet = "https://dweet.io/dweet/for/" + "pulseox" + "?" +
"SPO2" + "=" + str(SPO2) + "&" + "BPM" + "=" + str(BPM)
                rqs = requests.get(dweet)
                print(array[0], array[1], array[2])

                print("")

            spO2CurrentValue = readIn[0]
            heartRateCurrent = readIn[1]
            fiO2CurrentValue = readIn[2]
            ui.fiO2_LCD.display(fiO2CurrentValue)
            ui.spO2_LCD.display(spO2CurrentValue)
            ui.heartRate_LCD.display(heartRateCurrent)
            time.sleep(0.1)

class PulseOx(object):

    serial_monitor = None
    enPin = 23
    stepPin = 18
    dirPin = 24
    sleepPin = 12

    def __init__(self):
        self.initializePulseOx()

    def initializePulseOx(self):
        try:
            #set up Serial Monitor at TTYUSB0
            self.serial_monitor = serial.Serial(
            port = '/dev/ttyUSB0',\
            baudrate = 9600,\
            parity = serial.PARITY_NONE,\
            stopbits = serial.STOPBITS_ONE,\
            bytesize = serial.EIGHTBITS,\
            timeout = 1)
        except:
            print("could not open serial port")
            exit(0)
```

```
        #self.setupGPIO()
        print("Connected to: " + self.serial_monitor.portstr)
        #signal.signal(signal.SIGINT, self.signal_handler)


    def read_pulse_ox(self):
        try:
            #read ASCII serial output from monitor. Convert to string.
            #decode to utf-8
            line = str(self.serial_monitor.readline(), encoding =
sys.getdefaultencoding())

            #save line as list with each metric value as an element
            values = line.split()

            print(values[0][5:]) # Date, remove 5 encoded bytes
            print(values[1]) # Time

            #check if '---' is in the metric values of SPO2 and BPM
            #pulse ox outputs '---' if no reading is detected
            SPO2_string = values[3] #SPO2
            SPO2 = self.read_SPO2(SPO2_string)

            BPM_string = values[4] #BPM
            BPM = self.read_BPM(BPM_string)

            #return array
            global FIO2
            global readIn
            array = [SPO2, BPM, FIO2]
            readIn = array
            return array

        #Unicode error resulting from Pulse Ox outputting
        #some type of unknown character
        except UnicodeDecodeError:
            print("ASCII decode error\n")
            return [-1, -1]
        #exception in case readline() does not fully read buffer
        except IndexError:
            print("Index error. Could not read buffer")
```

```
            return [-1, -1]

    def read_SPO2(self, SPO2_string):
        #check if '---' is in the metric values of SPO2
        #pulse ox outputs '---' if no reading is detected
        dash = '---'
        #output '---' if no SPO2 is detected
        if(dash in SPO2_string):
            SPO2_string = "---%"
            SPO2 = -1
            print("SPO2 = " + SPO2_string)
        #if SPO2 is detected, convert substring to int
        else:
            SPO2_string = SPO2_string[5:8]
            SPO2 = int(SPO2_string)
            print("SPO2 = " + str(SPO2) + "%")
        return SPO2

    def read_BPM(self, BPM_string):
        #check if '---' is in the metric values of SPO2
        #pulse ox outputs '---' if no reading is detected
        dash = '---'
        #output '---' if no BPM is detected
        if(dash in BPM_string):
            BPM_string = "---"
            BPM = -1
            print("BPM  = " + BPM_string)
        #if BPM is detected, convert substring to int
        else:
            BPM_string = BPM_string[4:7]
            BPM = int(BPM_string)
            print("BPM  = " + str(BPM))
        return BPM

    def process_SPO2(self, SPO2, SPO2_lower_limit, startTime):
        #check if the SPO2 level is above the lower limit
        #check if SPO2 was read. SPO2 is -1 if a value was not read
        #if SPO2 is below limit, go to alarm function
        if((SPO2 < SPO2_lower_limit or SPO2 < 88)):
            self.SPO2_low_alarm(SPO2, SPO2_lower_limit)
            global FIO2
            if (FIO2 <= 90):
```

```
                self.moveRight()
                FIO2 = FIO2 + 10
            #reset the start time for another hour if there was an SPO2
warning
            return time.time()
        #return the original start time if there are no problems
        #will ignore a SPO2 of -1
        else:
            return startTime


    def SPO2_low_alarm(self, SPO2, SPO2_lower_limit):
      #raise an SPO2 warning
        print("SPO2 of " + str(SPO2) + " is below target SPO2 of " +
str(SPO2_lower_limit))

    def adjustFIO2(self):
        #adjust the FIO2 by lowering by 10%
        global FIO2
        if (FIO2 >= 10):
            FIO2 = FIO2 - 10
            self.moveLeft()

        #raise a warning if FIO2 falls below a certain level
        if(FIO2 <= 60 and FIO2 >= 50):
            self.FIO2_warning()
        elif (FIO2 <= 40 and FIO2 >= 30):
            self.FIO2_warning()
        elif (FIO2 <= 25 and FIO2 >= 20):
            self.FIO2_warning()

        return FIO2

    def FIO2_warning(self):
        #Display a warning
        global FIO2
        print("FIO2 is at " + str(FIO2))

    def setupGPIO(self):
        #degrees = 360
        #turnTime = degrees / 176.0
        #turnTime = 0.1704545
```

```
        enPin = 23
        stepPin = 18
        dirPin = 24
        sleepPin = 12

        GPIO.setwarnings(False)

        GPIO.setmode(GPIO.BCM)
        GPIO.setup(enPin, GPIO.OUT)
        GPIO.setup(stepPin, GPIO.OUT)
        GPIO.setup(dirPin, GPIO.OUT)
        GPIO.setup(sleepPin, GPIO.OUT)
        pwm = GPIO.PWM(stepPin, 2000)
        GPIO.output(enPin, GPIO.LOW)
        GPIO.output(dirPin, GPIO.HIGH)
        GPIO.output(sleepPin, GPIO.LOW)

    def moveRight(self):
        os.system("python motor.py")

    def moveLeft(self):
        os.system("python motorLeft.py")

    def signal_handler(self, signal, frame):
        #handle CTRL-C to close program
        print("CTRL-C")
        sys.exit(0)

    def encoderInitialize(self):
        spi = spidev.SpiDev()
        spi.open(0,0)

        #read from encoder
        resp = spi.xfer([0x00])
        while (resp < 90):
            moveRight()



if __name__ == "__main__":
    global words
```

```
app = QtWidgets.QApplication(sys.argv)
Dialog = QtWidgets.QDialog()
ui = Ui_Dialog()
ui.setupUi(Dialog)
Dialog.show()

t = MultiThread()
t.start()

sys.exit(app.exec_())
```

//

To give a short summary of the program, it first reads creates a serial-link to the MASIMO rainbow pulse oximeter. We then begin reading in the SPO2 data. We set a value for how often to analyze the data, for example an hour. After an hour, it will start the stepper motor to turn the knob to the left, actuating a reduction in FIO2. If the SPO2 data is disrupted or falls below a set value, it will call an alarm function and turn the stepper motor the right, actuating an increase in FIO2. The program repeats itself in a continuous for loop.

Contributions

Luis Perea:
I researched the different methods of creating the project, and purchased the components needed to build the device. I then began testing each component individually to ensure that they would work together. I then began making presentations and reports, as well as writing the code necessary to bring each component together. Finally, I put the device together, and proceeding to write this final report.

References

"Active Humidification." *Hudson RCI® Neptune® Heated Humidifier*. Teleflex, n.d. Web. 19 Sept. 2016. <http://www.teleflex.com/usa/product-areas/respiratory/active-humidification-and-breathing-circuits/neptune-heated-humidifier/features-and-advantages/?language_id=1>.

Akif. "What Is Minute Ventilation?" *Anesthesia General*. N.p., 27 Sept. 2010. Web. 16 Oct. 2016. <http://anesthesiageneral.com/what-is-minute-ventilation/>.

Barker, Steven J., and Bill Hay. "Pulse Oximetry." *Pulse Oximetry*. Masimo, 10 Sept. 2002. Web. 16 Sept. 2016. <https://web.archive.org/web/20150318054934/http://www.oximetry.org:80/pulseox/principles.htm>.

"Bird Blender High Flow." *Bird Blender High Flow*. CareFusion, n.d. Web. 19 Sept. 2016. <http://www.carefusion.com/our-products/respiratory-care/mechanical-ventilation/adult-pediatric-ventilation-solutions/bird-blenders/bird-blender-high-flow>.

"Bird Sentry Blender." *Bird Products Corporation* (n.d.): n. pag. *Food and Drug Administration*. FDA. Web. 19 Sept. 2016. <https://www.accessdata.fda.gov/cdrh_docs/pdf/K973646.pdf>.

Bitterman, Haim. "Bench-to-bedside Review: Oxygen as a Drug." *Critical Care*. BioMed Central, 24 Feb. 2009. Web. 22 Sept. 2016.

Bottrell, John. "Understanding Oxygen and Oxygen Levels With COPD." *Hypoxemia*. Health Center, n.d. Web. 20 Sept. 2016. <http://www.healthcentral.com/asthma/c/52325/175572/understanding-oxygen-levels/>.

Caso, Richard, and Irving Quam. AUTO-CONTROLLED AIR-OXYGEN BLENDER. Richard Caso, assignee. Patent 20140254305A1. 11 Sept. 2014. Print.

Chan, Vincent, and Steve Underwood. "A Single-Chip Pulsoximeter Design Using the MSP430." *Texas Instruments* (2005): n. pag. *Texas Instruments*. Texas Instruments, 1 Nov. 2005. Web. 6 Dec. 2016. <http://www.ti.com/lit/an/slaa274b/slaa274b.pdf>.

Clark, Jim. "The Beer-Lambert Law." *Chemistry LibreTexts*. UC Davis, 17 Sept. 2016. Web. 24 Nov. 2016. <http://chem.libretexts.org/Core/Physical_and_Theoretical_Chemistry/Spectroscopy/Electronic_Spectroscopy/Electronic_Spectroscopy_Basics/The_Beer-Lambert_Law>.

Duck, Annette. "Does Oxygen Need Humidification?" *Nursing Times*. N.p., 14 Dec. 2009. Web. 19 Sept. 2016. <https://www.nursingtimes.net/clinical-archive/respiratory/does-oxygen-need-humidification/5009649.article>.

Earl, Bill. "All About Stepper Motors." *Adafruit*. Adafruit, 24 May 2015. Web. 06 Dec. 2016. <https://learn.adafruit.com/all-about-stepper-motors/types-of-steppers>.

"FreeO2 - Oxynov." *FreeO2, Oxygen Therapy Optimization Device*. Oxy'nov, n.d. Web. 03 Oct. 2016. <http://www.oxynov.com/en/>.

Graybeal, J. M., and M. T. Petterson. "Adaptive Filtering and Alternative Calculations Revolutionizes Pulse Oximetry Sensitivity and Specificity during Motion and Low Perfusion." *Proceedings of the 26th Annual International Conference of the IEEE EMBS* (n.d.): n. pag. 2004. Web. 6 Dec. 2016. <http://www.masimo.com/pdf/clinical/set/graybeal-adaptive-filtering-and-alternative-calculations-revolutionizes-pulse-oximetry-sensitivity-and-specificity-during-motion-and-low-perfusion-sept-2004.pdf>.

Grusin, Michael. "Serial Peripheral Interface (SPI)." *Serial Peripheral Interface (SPI)*. SparkFun, 14 Jan. 2013. Web. 29 Jan. 2017. <https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi>.

"Heart Rate Click." *MikroElektronika*. MikroElektronika, 1 Jan. 2010. Web. 4 Nov. 2016. <http://www.mikroe.com/click/heart-rate/>.

Hord, Mike. "I2C." *Sparkfun*. Sparkfun Electronics, n.d. Web. 19 Sept. 2016. <https://learn.sparkfun.com/tutorials/i2c>.

"How Pulse Oximeters Work Explained Simply." *HowEquipmentWorks*. N.p., n.d. Web. 06 Dec. 2016. <https://www.howequipmentworks.com/pulse_oximeter/>.

"How to Read SpO2." (n.d.): n. pag. *Konica Minolta Sensing*. Windward University, 2006. Web. 29 Oct. 2016.

<https://windward.hawaii.edu/facstaff/miliefsky-m/ZOOL%20142L/aboutPulseOximetry.pdf>.

Iddir, Sofia, Rob Mahar, Naomi Thonakkaraparayil, and James Hart. "WIRELESS REFLECTANCE PULSE OXIMETER." *BME*. University of Conneticut, 17 Dec. 2004. Web. 22 Sept. 2016.

Kennedy, Stephen M. "An Introduction to Pulse Oximeters." *University of Wisconsin-Madison*. University of Wisconsin-Madison, 20 Apr. 2015. Web. 6 Nov. 2016. <http://www.imt.liu.se/FoUtb/kurser/oxikurs/restricted/pulseoximetersequationsandtheory-stephenkennedy.pdf>.

Kim, Victor, Joshua O. Benditt, Robert A. Wise, and Amir Sharafkhaneh. "Oxygen Therapy in Chronic Obstructive Pulmonary Disease." *Proceedings of the American Thoracic Society*. American Thoracic Society, 01 May 2008. Web. 20 Sept. 2016. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2645328/>.

Mannion, Patrick. "Teardown: Inside the Art of Pulse Oximetry." *EDN NETWORK*. N.p., 19 Feb. 2010. Web. 02 Jan. 2017. <http://www.edn.com/design/medical-design/4018428/2/Teardown-Inside-the-art-of-pulse-oximetry>.

Oak, Sang-Soo, and Praveen Aroul. "How to Design Peripheral Oxygen Saturation (SpO2) and Optical Heart Rate Monitoring (OHRM) Systems Using the AFE4403." *Texas Instruments* (n.d.): n. pag. 1 Mar. 2015. Web. 5 Nov. 2016. <http://www.ti.com/lit/an/slaa655/slaa655.pdf>.

"Pulse Oximeters - Premarket Notification Submissions [510(k)s]: Guidance for Industry and Food and Drug Administration Staff." *FDA*. N.p., n.d. Web. 06 Dec. 2016. <http://www.fda.gov/RegulatoryInformation/Guidances/ucm341718.htm>.

Shah, N., and A. Taleghani. "Comparison of Three New Generation Pulse Oximeters during Motion and Low Perfusion in Volunteers." *Journal of Neurosurgical Anesthesiology* 18.4 (2006): 297-98. *Masimo*. 2005. Web. 15 Jan. 2017.

Stuban, Norbert, and Niwayama Masatsugu. "Non-invasive Calibration Method for Pulse Oximeters." *Periodica Polytechnica*. N.p., 22 Aug. 2008. Web. 10 Nov. 2016. <https://pp.bme.hu/ee/article/viewFile/859/478>.

"Understanding Pulse Oximetry." *Phillips Medical Systems*. Phillips, n.d. Web. 19 Sept. 2016. <http://incenter.medical.philips.com/doclib/enc/fetch/586262/586457/Understanding_Pulse_Oximetry.pdf%3Fnodeid%3D586458%26vernum%3D2>.

Appendix

Automatic HIFLO Oxygen Weaning Device

Team Members: Sarah Dowling, Ryan Foster, Kate Garrett, Justin Joseph, Melanie Kemp, Luis Perea
Faculty Advisor: Dr. Pidaparti

INTRODUCTION

Currently, respiratory therapists (RTs) and nurses must maintain a patient's oxygen levels by adjusting the fraction of inspired oxygen (FiO2) being delivered. The FiO2 is gradually decreased based on the patient's blood-oxygen saturation (SpO2) levels and the patient's ability to breathe independently.

However, RTs and nurses have many other competing obligations, limiting their ability to properly monitor the patient's oxygen levels. This can put patients at risk of exposure to both high and low oxygen concentrations (hyperoxia/hypoxia), which can cause lung damage or unhealthy development of pediatric patients in the PICU. The auto-weaning device helps prevent hypoxia/hyperoxia via a feedback loop by sensing O2 levels and adjusting the flow accordingly, allowing therapists to use their time more efficiently.

OBJECTIVE

Our team worked to provide a safer and more efficient modification by creating an electrical feedback component to the blenders that adjusts the amount of oxygen being administered to the patient based on their stability and saturated blood oxygen levels.

Figure 1. Current setup of non-invasive HIFLO oxygen nasal cannula system

DESIGN REQUIREMENTS

- Device is encased and sealed with 6mm acrylic for sterilization with ethanol
- The device attaches to the transportable HIFLO stand
- The device weighs approximately 8 pounds for easy transportation
- Weaning capability (feedback loop *Figure 2*)
- Sound alarm at FiO2 checkpoints of 60%, 40%, 25% to alert healthcare providers
- Compatible with CareFusion BIRD air-oxygen blender
- Use pulse oximeter for noninvasive patient SpO2 monitoring
- Meet Clause 201.12.1 of ISO 80601-2-61:2011 for pulse oximeter testing purposes
- Comply with section 201(h) of the Federal Food Drug and Cosmetic (FD&C) Act for medical device categorization

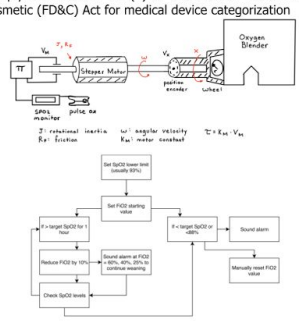


Figure 2. Feedback loop used for weaning patients off oHIFLO device

PROTOTYPE

- **Rapid Prototyping:** 3-D printing and laser cutting were used to develop cost-efficient pieces that could quickly be modified or reprinted as needed.

Figure 3: 3D printed sleeve attaches the knob from the blender to the rod on the stepper motor.

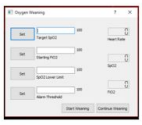
Figure 4: Laser printed case printed from 6mm acrylic encompasses hardware and the blender.

- **Hardware:**
  - Raspberry Pi - IoT embedded system
  - Stepper motor - actuator to adjust FiO2
  - RLS absolute encoder - record position of motor
  - MASIMO pulse oximeter - read SpO2 data from patient
  - MASIMO Radical 7 monitor - process SpO2 data

Figure 5: Complete hardware configuration.

RESULTS

- SpO2 data is analyzed over time by Raspberry Pi.
- Device uses PID control to adjust FiO2.
- Embedded System processes SpO2 feedback loop.
- Device follows the HFNC protocol.
- GUI allows for UI/UX interaction.



College of Engineering
UNIVERSITY OF GEORGIA

Sponsor/Client: Children's Healthcare of Atlanta,
Dr. Pradip Kamat, Dr. Jocelyn Grunwell


PICU
Top Priority Safety
Concerns

Dr. Pradip Kamat (left)
Dr. Jocelyn Grunwell (right)

Dr. Peter Kner

Dr. Kylie Johnsen