Designing a Low-Cost Ventilator for the COVID-19 Pandemic

May 12, 2020

Team Red: Harshvardhan Babla, Samuel Breckenridge, Colby Chang, Joe Chen, Matthew Collins, Claire Dong, Jerry Huang, Lap Hei Lam, Kalil Shaw, Ethan Thai

Advisors: Professor Jeff Thompson, Dr. Jonathan Hodges

We hereby declare that this report represents our own work in accordance with University regulations.

Abstract

During the global COVID-19 pandemic, many patients are developing respiratory issues, and require ventilators to assist with breathing. However, this large demand has led to a shortage of ventilators, which are expensive and affected by supply disturbances caused by the pandemic. In this project, we worked on designing a simple, low-cost ventilator with tuned controls and a user interface to address these issues. We were able to create hardware and software prototypes of our design. Our physical prototype showed the operation of tubing and valves, and we demonstrated our controls and user interface by connecting them with a coded lung simulation. In the future, our hardware and software aspects can be integrated to form a complete system.

Acknowledgements

We would like to thank everyone involved in this project for their support and insights. We are especially grateful to Professor Thompson and Dr. Hodges, who guided us through the transition to this novel project, and whose advice during Monday and Wednesday meetings on everything from pancake compressors to online parts shopping was invaluable. We are also grateful to Radd for his help with 3D printing parts and for his extensive knowledge and assistance with tools and materials. We would also like to thank the members of Team Green and Team Blue for their support and their interesting presentations during weekly meetings. Finally, thank you to the Electrical Engineering Department for their generous funding of our project.

Contents

Chapter 1

Background and Previous Work [Joe]

Due to the current COVID-19 pandemic and the respiratory issues it causes, there are many patients who need ventilators to assist them with breathing. However, there is a shortage of these expensive medical devices, which has inspired many groups to work on cheaper, easily manufacturable ventilators. Before we started to plan our own design, we first surveyed existing, low cost ventilators that were already being designed and open-sourced. In particular, we found three designs that seemed to work well: an MIT design that used a bag-valve mask (BVM), a PVC pressure controlled design from UF, as well as the Mechanical Ventilator Milano (MVM). We ultimately considered the benefits and shortcomings of each of these projects as well as expenses and part availability to formulate our initial idea. While we evaluated these designs, we also considered the specifications outlined by the UK government [24].

1.1 Medical Ventilator Milano [9]

The MVM is a pressure controlled design that costs roughly a few hundred dollars and can be produced quickly at a large scale[3]. Along with the typical mandatory mode, MVM also has a patient assisted ventilation mode in which the patient triggers airflow from the inspiration line when the ventilator detects negative pressure. This feature is a lot safer than other low cost ventilators which can only do mandatory mode. The MVM uses a pressure sensor, flowmeters and solenoid valves to regulate the pressure. In general, there were two main proportional solenoid valves which switched on and off in the inspiration and expiration line. There are also valves which correct for PEEP pressure. Air flows through medical tubing. This design is controlled by a microcontroller board which has Wifi for central monitoring and a user-interface that allows for adjustments to make ventilation customizable for the patient. In general, we thought MVM had a very solid design but we were unable to acquire some of the parts and chips in their specification.

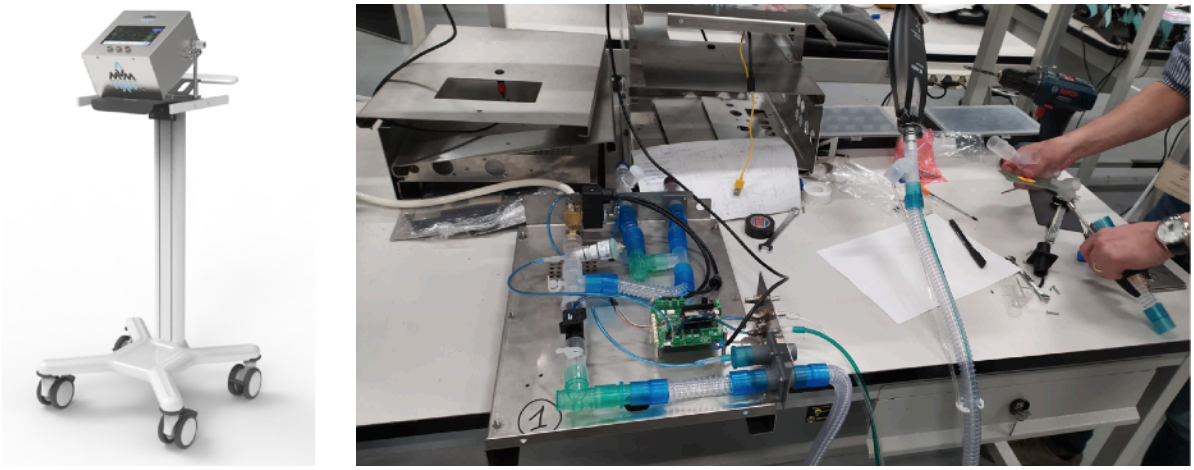

Figure 1.1: Medical Ventilator Milano

1.2 MIT E-Vent [15]

The MIT design was an alternative that uses a bag-valve mask compressed by a pivoting cam arm[4]. In this design, there originally was no patient assisted ventilation mode and instead there is only manual ventilation which ideally should only be given to sedated patients. Also, unlike the MVM design, this is not pressure controlled ventilation. However, further prototypes were built and are currently being developed with assisted control mode as well as alarms that detect over-pressurization. These newer prototypes would have a bulk-manufacturing price of $200. Due to the fact that the BVM does not supply a constant pressure, this poses a risk of barotrauma, which is especially alarming for patients struggling with acute respiratory distress syndrome (ARDS). On top of that, it cannot maintain a positive end-expiratory pressure, as the MIT paper says it is a future feature yet to be added. Another issue is that the BVM cannot regulate oxygen supply, which would harm a patient if they are not given the right amount. For the user-interface, the BVM design has an LCD screen. Considering all the risks with not doing pressure control, we found the BVM design unreliable.

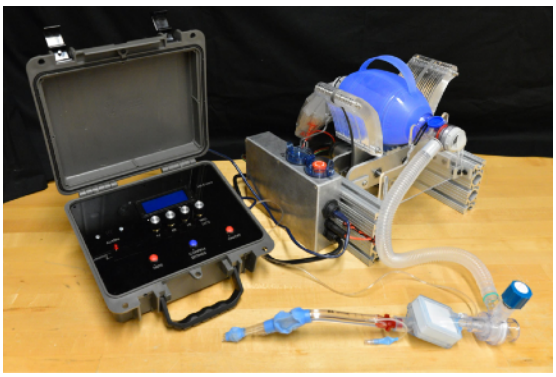

Figure 1.2: MIT E-Vent

1.3 UF Open Source Ventilator Project [16]

The UF design was a pressured controlled design made from PVC pipes and sprinkler parts that seemed to be easily assembled and low cost, but their specifications and documentation was not the most rigorous. In order to have PEEP, the design seems to have included an inflatable bulb near the expiration valve to add some pressure resistance[5]. The design uses solenoid valves, a flow and pressure sensor and has a controller and user interface. The user interface has an LCD display, adjustable variables as well as alarms[6]. We considered doing a PVC build but ultimately did not. We also thought the UF UI design was not as good as the MVM which uses wifi and could allow for simultaneous centralized monitoring.

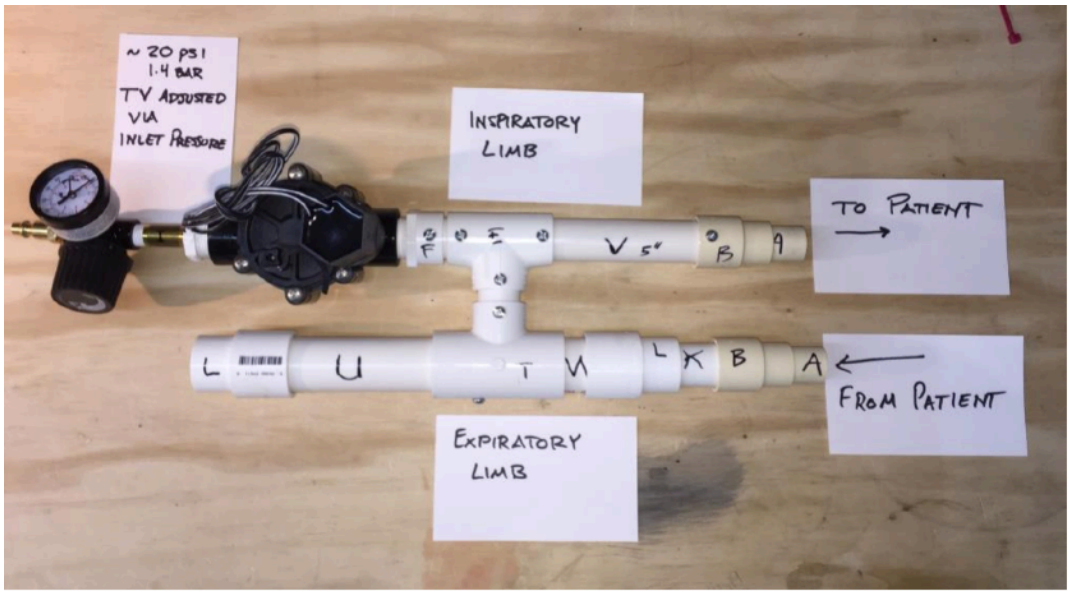

Figure 1.3: UF Open Source Ventilator Project

1.4 Project Goals

For this project, our goal was to design and execute a simple, low-cost ventilator that would include all the hardware and software components necessary to function. To accomplish this, we divided into subsystems within our team: design and build, controls and lung simulation, and user interface. This way, we each had an interesting and unique projects to work on, but could still work in parallel. This had the additional benefit that even if one subsystem wasn't working, other subsystems could connect to each other to partially verify their operation.

Chapter 2

Design, Build and Hardware [Claire, Harsh, Kal]

2.1 Overview

In this subsystem, we focused on designing and executing a physical ventilator prototype. Our main focus was having a simple design which only kept essential parts to lower cost and avoid supply issues. With each part, we also made design choices to minimize cost, and be realistic with what was possible and practical to construct. We also had to design and build a testing apparatus to verify our system. Our prototype was mostly inspired by the MVM ventilator design, but we also looked to other projects from UF and Hackaday. [11]

2.2 Prototype Design

To design our prototype, we studied the MVM design, and the minimum requirements for a functional ventilator. We decided that the essential parts of our prototype would have to include inspiration and expiration lines and proportional valves on both lines to control inhalation and exhalation cycles. Since ventilators must be able to output pressure proximal to the patient, tidal volume, and flow to the user interface, we also needed a spirometer and a pressure sensor. Additionally, we needed a microcontroller to read and process sensor data, and to supply signals to the valves. Finally, to test our apparatus, we needed a pressurized air supply, and a test lung. The schematic for our prototype is in Figure 2.1 below, along with details about components and connections.

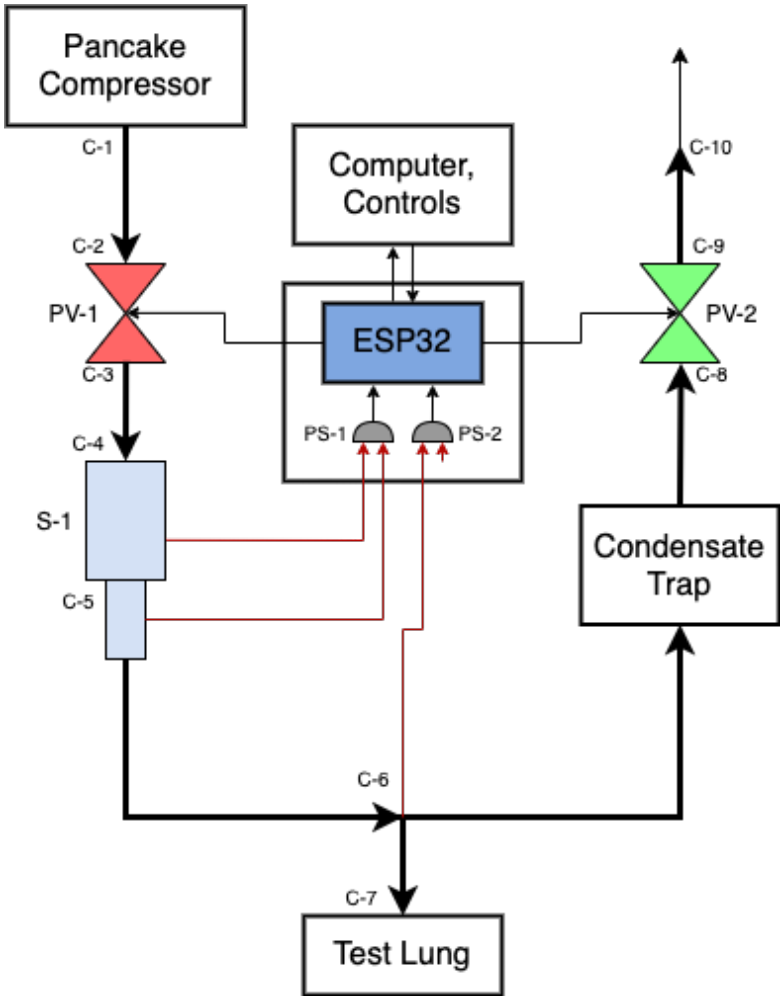

Figure 2.1: Prototype schematic

2.2.1 Components

Tubing: For our main tubing on inspiration and expiration lines, we used Masterkleer soft PVC plastic tubing with an inner diameter of ½". We chose this tubing both for its availability from McMaster-Carr, and because the ½" diameter was a good fit for other components, and similar to the 22 mm diameter of standard ventilator tubing [9]. This tubing was used throughout our system, except at the spirometer, and at the exhalation vent.

Pancake Compressor: As part of our testing setup, we needed a supply of pressurized air, and chose the Porter-Cable pancake compressor, which had an adjustable pressure output of 0 - 200 PSI. For testing, we connected the pressure connector to the start of the inspiration line, and set a low pressure output of approximately 1 PSI, which allowed our test lung to fully inflate and deflate during the two second inhales and exhales that we tested with.

PV-1 Proportional Valve: Since one of our primary objectives was cost minimization, we created a cheaper proportional valve by pairing a ball valve with a servo. The angle of the servo blade determined how much the valve opened or closed. For this, we used a miniature chrome-plated brass ball valve available from McMaster-Carr, and a high torque TowerPro MG995R servo. This valve controlled air flowing in through the inspiration line. The design and operation of the servo proportional valve is further explained in section 2.2.3, Specialized Components

S-1 Spirometer: To keep our design low cost, we built our own spirometer by using the Bernoulli effect, where the flow and volume of air traveling between two pipes of different diameters can be calculated by the differential pressure between those pipes. We added a section of 1" ID Masterkleer PVC tubing here to be a larger diameter pipe, and connected to the rest of the ½" ID tubing which served as the smaller diameter pipe. We also attached two pieces of ⅛" Masterkleer tubing to the larger and smaller pipes to feed out to our pressure sensor, PS-1. The design and operation of the Venturi spirometer is explained in detail in section 2.2.3, Specialized Components

Test Lung: As we did not have the means of building and testing a system that operates on a live patient, we developed a 'test lung' made from an altered air bellow. Specifically, a Coghlan Foot pump bellow was chosen, which is typically used for medium sized inflatables. The device is designed as a displacement pump which pumps air out when compressed, and restores with the help of a spring. By removing this internal spring, we were able to create a device that can open and close purely based on the difference in pressure caused by our system. This allows us to simulate a lung inhale and exhaling as would a COVID-19 patient.

Condensate Trap: Another component we incorporated into our system, was a condensate trap. This device, in theory, would be used to remove excess moisture from the system to prevent damage to other valves and electronics within the system. As this system was not operated on a human lung we did not have capacity to fully test this component. That said we believed it was a component feasible of making and including in our system. The trap was made of a simple T-connector. Two of these connections are used to continue the expiratory airway. The third connection, however, consisted of a closed 22 mm tube to collect water.

PV-2 Proportional Valve: This valve on the expiration line controlled the flow of air from a patient's exhale. Its design is the same as the PV-1 valve on the inspiration line, and is explained in more detail in section 2.2.3, Specialized Components

ESP32: We chose the ESP 32 developer-board as the onboard microcontroller. The ESP32 operates on digital 3.3V, but also provides a 5V source. The 5V source was sufficient to power our sensors and servo motors**.** The primary reason for choosing this board was that it is packaged with Wi-Fi and Bluetooth transceivers. We envisioned using these protocols to log data onto a web-server. Our final design used the ESP32's 2.5 GHz Wi-Fi channel to host a website with a constantly updating log of information. The ESP32 is integrated with 4MB of flash storage, which was more than enough to process our control loops and log data via WiFi. The device is easily programmed using the Arduino IDE, and can be debugged by Arduino IDE's USB Serial Monitor. Moreover, the device is branded as an "ultra-low power" and inexpensive microcontroller.

PS-1 Differential Pressure Sensor: This pressure sensor measured the differential pressure between the 1" ID and ½" tubing of the spirometer, which could then be used to calculate flow and tidal volume to be displayed in the user interface. We used the MPXV5100DP differential pressure sensor, which had an operating pressure of 100 kPa. While this was too high for our ventilator, which should have pressures up to 40 cmH2O or around 4kPa [24], this was one of the few sensors in stock at the time.

PS-2 Differential Pressure Sensor: This pressure sensor measured the differential pressure between the tubing proximal to the test lung, and atmospheric pressure. Here, we also used a MPXV5100DP differential pressure sensor. This pressure would also be displayed in the user interface.

2.2.3 Connections

C-1: This connection is between the ¼" quick connect on the pancake compressor to the ½" tubing of the inspiration line, and is a quick disconnect hose coupling screwed to a barbed hose fitting. The tubing fits over the barbed end of the hose fitting and is secured with a worm drive clamp.

C-2, C-3, C-8, C-9: These are all connections between a ball valve and ½" tubing. They consist of a brass barbed hose fitting that screws into the valve, and a worm drive clamp that clamps the tubing to the hose fitting.

C-4: This is a connection between ½" tubing and 1" tubing for the start of the spirometer. The two tubing sizes are connected by a barbed tube fitting, which reduces from 1" to ½".

C-5: This is a connection from 1" to ½" tubing for the spirometer, and also uses a barbed tube reducer.

C-6: This is a connection between the inspiration and expiration lines and the test lung. This consists of a Y barbed tube fitting, with the inspiration and expiration tubes on the two arms of the connector.

C-7: This is the connection from ½" tubing to the test lung. We were able to fit the tubing into the opening of the plastic pipe connected to the test lung and make an airtight seal without clamps.

C-10: This is the expiration vent, which connected ½" tubing to ⅛" tubing in an attempt to supply PEEP pressure. This connection was made by cutting a hole in the ½" tubing, putting the ⅛" tubing inside, and sealing it with hot glue. This connection and PEEP is further explained in section 2.2.3, Specialized Components.

2.2.3 Specialized Components

ESP32

The ESP32 measures the pressure at the patient's mask (modeled by opening of our test-lung) and the quantity of airflow from the Venturi spirometer. These readings come from analog input pins connected to the respective pressure sensors.

The ESP32 varies the airflow in through the system by a PWM signal which controls a pair of servos. Each servo corresponds to a proportional valve (PV-1 or PV-2). The proportional valves are constructed such that the servo-blades turn with the handle of a ball-valve. As such, varying the servo-position, varies the extent to which the ball valve is opened.

The ESP32 is programmed with a control algorithm (see the section on Controls for more details) which opens PV-1 and closes PV-2 during inspiration, letting air to flow into the patient's lungs; and does the opposite during expiration. By varying the positions of the respective servos, the control loop maintains a gently rising pressure during inspiration and a constant PEEP pressure during expiration.

As alluded to earlier, the ESP32 hosts a website on its web-server. The website graphs the pressure at the patient's mask, the instantaneous flow of air supplied to the patient, and the volume of air supplied since the last breath, for a healthcare professional. The website also allows a doctor/nurse to vary the parameters controlling the patient's breathing cycles, such as the PEEP pressure, respiratory rate, and tidal volume supplied to the patient. As these quantities are updated, the control algorithm responds correspondingly to achieve these parameters. For more details see the section on the User Interface.

Servo Proportional Valve

To reduce the overall cost of components, we constructed a proportional valve by coupling a servo to a ball valve. A ball valve is a quarter-turn valve, whose handle is connected to a hollow, pivoting ball. It is traditionally used just to turn the flow of air on or off. However, we tied the ball-valve's handle to a servo. By varying the angle of the servo blades, we are able to vary the

opening in the valve, and hence the amount of air flowing through it. Figure 2.2 below, shows this setup.

This design is based on a design on thingiverse.com [23]. We 3D printed a holder, as shown in Figure 2.2, using the CAD design from the Thingiverse project. The servo was screwed into the top of the holder, while the valve was held in place by the cup below. We punched holes into the ball valves handles, to wire the handle to the servo's blades.

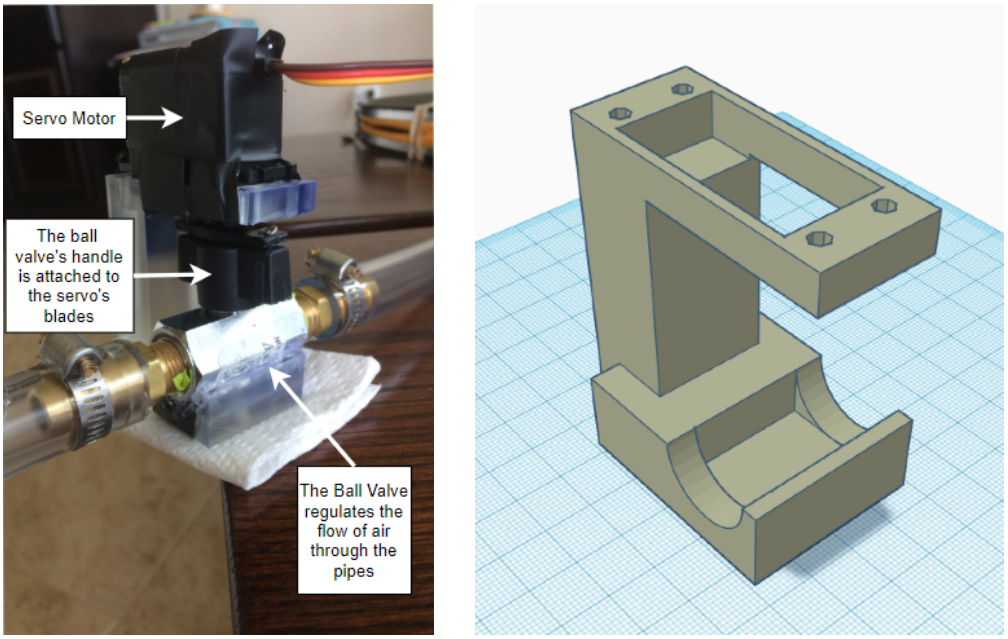

Figure 2.2: (Left) The completed servo proportional valve setup. (Right) The 3D CAD design for the holder; provided by [23].

Venturi Spirometer

As another method to keep our design simple and low cost, we created a Venturi tube spirometer to measure flow and tidal volume, based on a project done by Vanderbilt students [29]. The Venturi spirometer is a Venturi tube, consisting of two pipes of different diameters, and relies on the Bernoulli principle, where the difference in pressure between two pipes with different cross sectional areas can be used to calculate flow. In the Bernoulli principle, as a fluid flows from a larger pipe into a smaller pipe, its velocity increases, resulting in a drop of pressure. We can calculate the flow rate through the second pipe from $F = A*v$, where A is the cross sectional area of the second, 1/2" tubing, and v is the air velocity, calculated from

$$V_1 = \sqrt{\frac{2(p_1 - p_2)}{\rho\left[1 - (A_2/A_1)^2\right]}}$$

Since we were not able to get our pressure sensor working, we were unable to verify the performance of the spirometer.

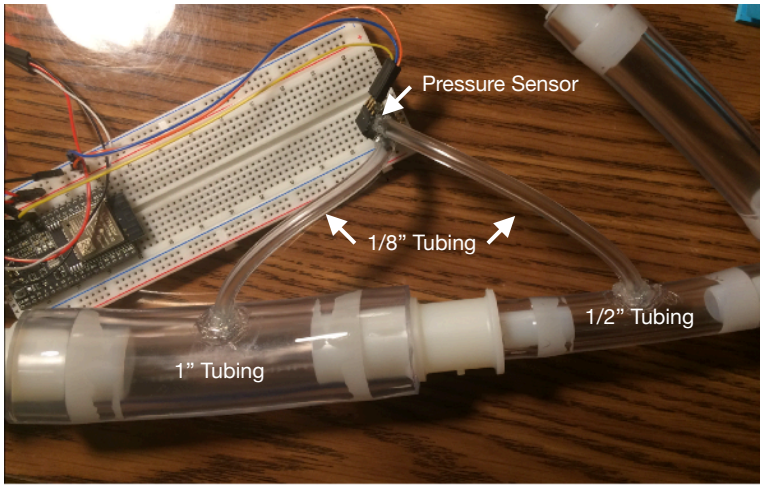

Figure 2.3: S-1, Spirometer

Expiration Vent

To maintain a PEEP pressure, we decided to use resistive pressure, since it would be easier and cheaper to implement, and because we couldn't find a good source for an over pressure valve small enough for pressures in the range of 15 cmH2O [24]. We thought that if we connected a length of ⅛" tubing to the end of the expiration valve, it would supply a resistive pressure through its smaller diameter. However, during actual testing, the test lung was not able to exhale quickly enough through the small tubing, and we had to open up the ½" tubing on the expiration line instead. This could be because we did not factor in the resistive pressure of the rest of the tubing, or because we did not measure the compliance of our test lung, which could have been too compliant and therefore difficult to exhale.

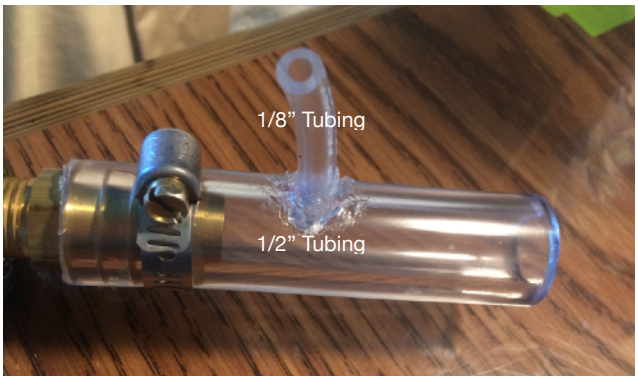

Figure 2.4: Expiration Vent

2.2 Completed Physical Prototype

We did build a physical prototype, although it was missing some parts due to supply and shipping issues. We also did not have time to get readings out from our pressure sensor, and so were not able to connect the physical prototype with controls code or the user interface. We did run our prototype with simple code which opened the inspiration valve and closed the expiration valve for two seconds, then switched the valve positions for another two seconds. When the inspiration line was connected to the pancake compressor, we were able to observe the test lung inflate and then deflate, simulating two second long inhales and exhales. An image of the final prototype is in Figure 2.5.

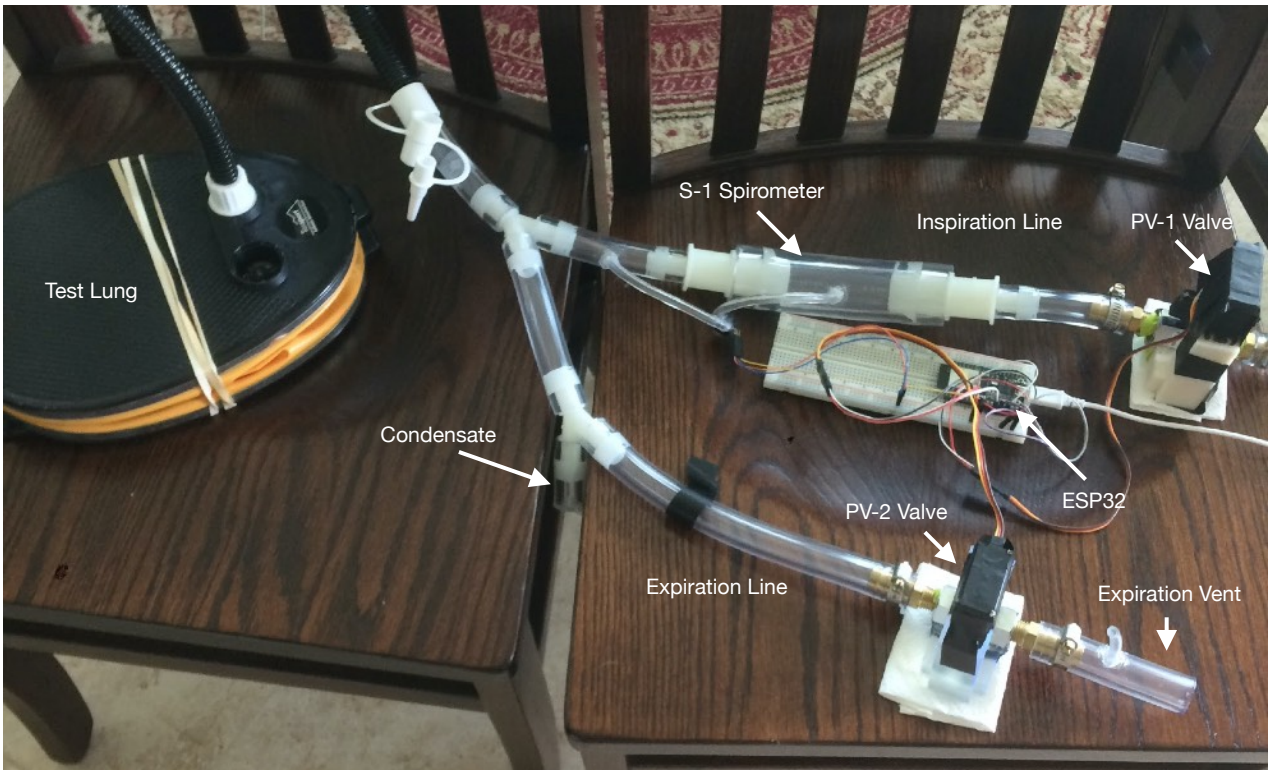

Figure 2.5: Completed prototype

Chapter 3

Controls [Colby, Ethan, Jerry, Lap]

3.1 Overview

To operate the build of the ventilator, a means of regulating the airflow into a patient was necessary. This was accomplished through the design of a pressure-regulated volume control (PRVC) algorithm determining the position of inspiratory and expiratory servo valves. Our motivation behind this was to control the pressure, which would then provide the volume necessary for inspiration and expiration, while not letting a dangerous amount of pressure build up. To regulate the pressure, we adjusted the amount of flow by changing the position of the valve controlling airflow, from 0 degrees (meaning completely blocked) to 90 degrees (meaning completely open)

3.2 PI Controls

As mentioned before, the controls program is based on a PRVC algorithm that aims to mimic the PRVC ventilation mode found on modern ventilators. The basic idea behind PRVC is to control the tidal volume by adjusting the pressure of the ventilator via the inspiratory and expiratory servo valves. The control loop measures the tidal volume after an inspiration and compares it to a predetermined volume. If the measured volume is greater than the predetermined volume, then pressure is decreased. If the measured volume is less than the predetermined volume, then the pressure is increased. If the measured volume is equal to the predetermined volume, then the pressure stays the same.

There are three functions associated with the controls program: PIDiteration(), recalibratePressure(), and flowcalc(). PIDiteration() determines the position of the servo valve by taking in the elapsed time measurement, pressure of the ventilator, and a pressure set point. recalibratePressure() takes in the volume of the inspiration, the desired tidal volume, and the previous pressure set point to create a new pressure set point. flowcalc() takes in the position of the servo valve and mouth pressure of the ventilator to determine the flow of air. This calculation of servo valve position and mouth pressure to airflow will be discussed more in the lung simulation.

PI control was used to regulate the position of the servo in PIDiteration(). Since the feedback control loop was being run every millisecond during the inspiration, it offers the integral error term time to grow and become effective while not placing too much onus on the proportional constant. This reduced oscillatory behavior, as the integral control smoothed out the proportional reaction to smaller errors close to the pressure set point. On the other hand, solely proportional control is used for changing the pressure set point in recalibratePressure(). This decision was made because, realistically, the pressure set point should not have to change much from breath to breath. Over time, compliance might decrease or increase, but it will not be changing drastically. Thus, adding an integral term or derivative term to the controller would not be terribly impactful.

3.3 Process

We decided to use C instead of another language with more built-in libraries like Python or Java to create the controls because C could be easily run on the ESP32, which housed our UI. To streamline our process, we split our controls, lung simulation, and test code to combine the two into different files, and used header files to provide an interface through which our test code could use both the lung simulation and controls. We also used a makefile to simplify our process of compiling and tuning our gains, which saved us a lot of time. We also had a .csv file for our test code to output lung pressure, volume, flow, and the position of the valve, which allowed us to graph our results, and see if our gains were too low, too high, or just right.

3.4 Results

Below, Figure 3.1 is an example of a single inspiration with gains that are too high. As you can see from the servo position graph, up until around 0.3 seconds, the servo position is wildly oscillating from values up to 60 then back down to 0, going back and forth from low to high every millisecond. Due to this servo oscillation, the flow value also oscillates from values up to ~6 L/s to 0 L/s. Because the gains are too high, the controller is continually overcorrecting for being under or over the pressure set point by swinging too far in the opposite direction. This sort of behavior can result in barotrauma, as it will create pressures far above the pressure set point and permanently harm the patient. In addition, it is unclear if the servo itself will be able to correctly actuate with these rapid oscillatory signals; these signals might even break the servo.

Though the tidal volume is accurately reached with these gains, they are clearly insufficient for actual use in the ventilator.

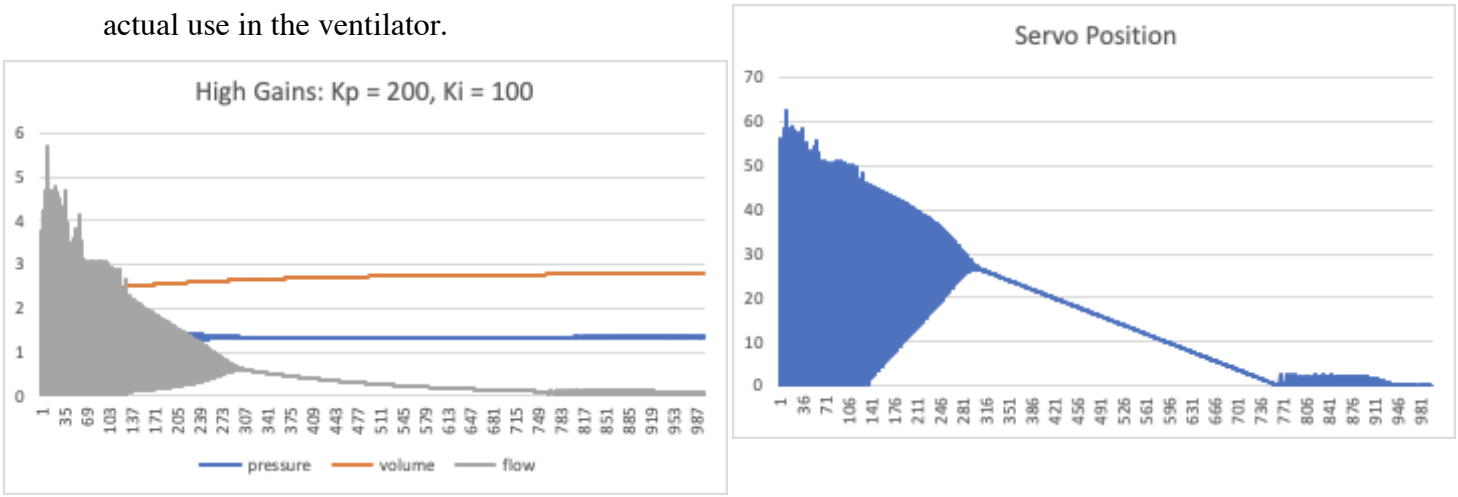

Figure 3.1: Pressure, volume, flow, and servo position graphed against time (milliseconds) with gains that are too high

Next is an example (Figure 3.2) of a single inspiration with gains that are too low. Completely opposite of the previous example, the servo position doesn't oscillate at all, but rather takes a long time to balance out at a position of 33. The pressure set point is not reached until around 0.6 seconds, and then it is exceeded because the low gains cause the controller to react too slowly. In this case, the tidal volume is actually exceeded by around 0.28 L, which is significant considering the tidal volume was only 0.5 L to begin with (so it is overshot by 56%). This creates volutrauma, which is also incredibly dangerous for patients and causes lasting damage. Thus, it is clear that these gains are insufficient for actual use in the ventilator as well.

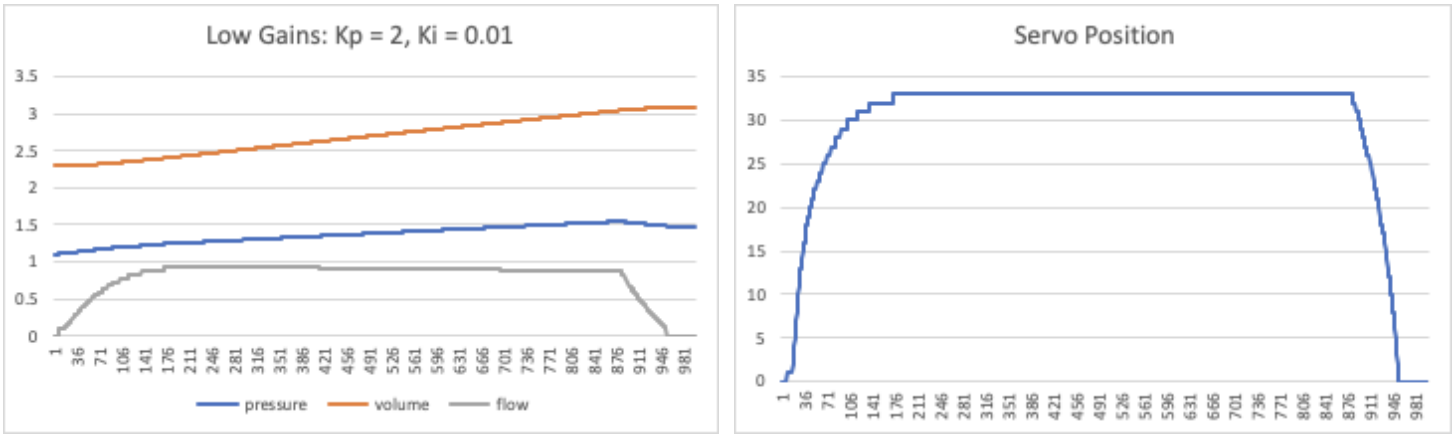

Figure 3.2: Pressure, volume, flow, servo position graphed vs. time (ms); gains are too low

Figure 3.3 displays 6 simulated inspiration-expiration cycles with reasonable gains. Looking at the servo position, it quickly reaches its peak of around 50 (within 2 milliseconds) and then gradually decreases down to 0. The pressure values hover exactly around the set point for the entirety of the inspiration, exactly as desired for PRVC. The tidal volume is also correctly reached in a timely manner. Comparing this performance to data from physical ventilators, the similarities make it clear that these gains are sufficient for actual use in the ventilator.

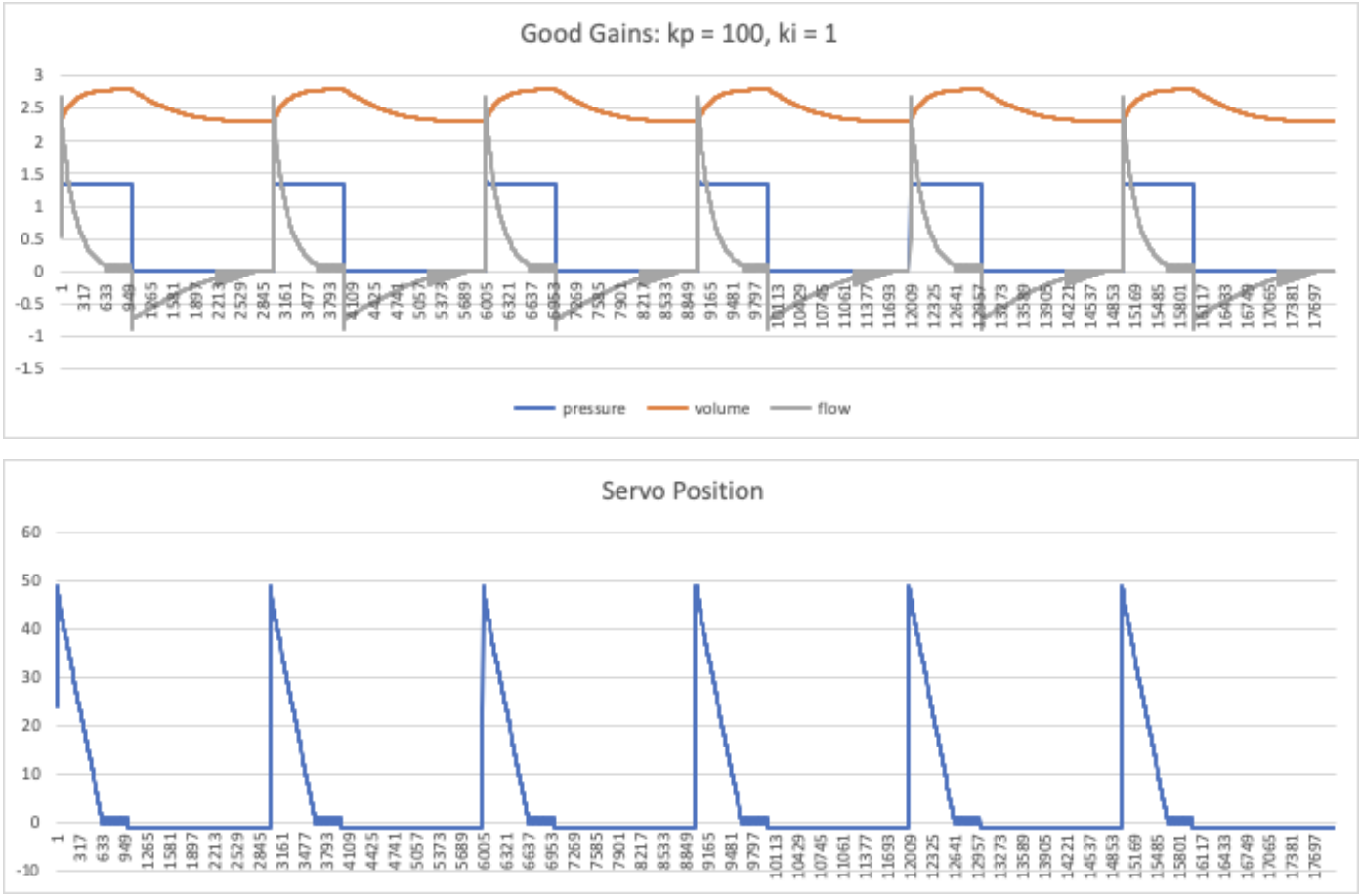

Figure 3.3: Pressure, volume, flow, and servo position graphed against time (milliseconds) over 6 inspiration-expiration cycles with gains that are reasonable

In Figure 3.4, there are 5 inspiration-expiration cycles using the reasonable gains. These differ from the previous example in that they began with a lower initial pressure set point. As you can see in the first cycle, the flow and position both start off with values markedly lower than the rest of the cycles. The volume also does not reach the desired final volume of 2.8 L, instead only hitting around 2.6 L. This is simulating a change in compliance. After the first cycle, the

proportional controller realizes that the tidal volume was not reached, and corrects by raising the pressure set point. It's a bit hard to tell, but the second inspiration plateau pressure is higher than the first inspiration plateau pressure. This allows the lung to reach much closer to the desired final volume of 2.8 L. The proportional controller continues to make minute adjustments over the next three cycles until the tidal volume is almost perfectly reached.

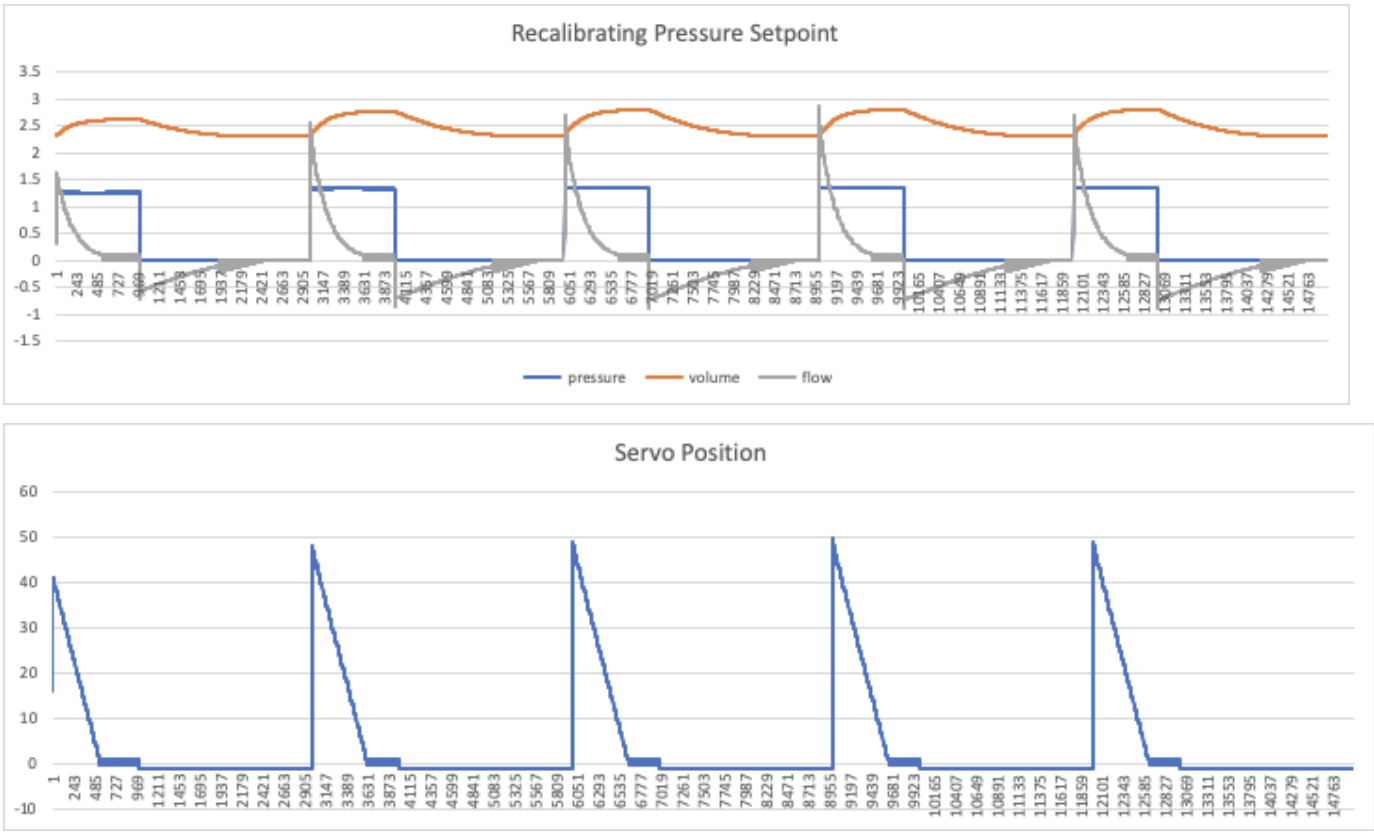

Figure 3.4: Pressure, volume, flow, and servo position graphed against time (milliseconds) over 5 inspiration-expiration cycles displaying the pressure set point adjustments between cycles

Chapter 4

Lung Simulation [Colby, Ethan, Jerry, Lap]

4.1 Overview

As ordered parts faced shipping delays, it became increasingly clear the physical ventilator build would not be complete with sufficient time to implement and tune the control algorithm. In response, our team sought to design a costless virtual means to test the efficacy of both the user interface and controls programs, resulting in the construction of a lung simulation. The goals of this project were simple: we needed a black box, passing in a flow and a volume, and then returning the pressure within the lung. In doing this, we could iteratively tune our gains and see what reasonable values for the gains might be.

4.2 Literature Review

In approaching this problem, we first sought to gain a better understanding of past software models of the lung from which to build our own simulation off of. We found that purely software simulations of the lung were sparse, if existent at all, redirecting our search to more fundamental mathematical models of the respiratory system.

A critical paper in the understanding of lung modeling was *Computer-Controlled Mechanical Simulation of the Artificially Ventilated Human Respiratory System* (Mesic et al.) This piece outlined the underlying respiratory models and assumptions used in a mechanical test lung. One such model was the linear single cavity lung, furthered by the assumptions of a constant lung compliance and airway resistance. In essence, this description of the respiratory system abstracts the lungs into a single air container and ignores the time variations of compliance and flow dependence of airway resistance.

$$P_m = 1/C * V(t) + R_{airway} * Q(t)$$

In which P_m is the pressure at the mouth of the lung, C is lung compliance, V is the time dependent volume, R is the resistance of the airway, and Q is the instantaneous airflow into the patient. This model offered the simplicity needed to produce a minimally viable simulation, while retaining the core dependencies on patient parameters such as compliance.

4.3 Simulation Design

We began constructing the simulation in C, to keep consistency with the controls program and allow for interchangeability with the UI. The first step was to define what was being simulated, we had settled on a variation of mandatory ventilation mode: the production of a fixed number of breaths over a fixed time period to the patient.

To simulate this, we provided the adjustable parameter of number of breath cycles, and a variable inspiration/expiration time. Within each breath cycle, pressure value inputs to the PI loop would be iterated to produce a new servo valve position, from which a new airflow value would be calculated, allowing us to update the lung with new air flow and volumes to produce new lung pressures.

The air flow value was produced using the mouth pressure as well as the servo position along with the pipe flow analysis energy equation:

$$\frac{P_1}{\varrho} + \frac{1}{2}\alpha v_1^2 + gz_1 = \frac{P_2}{\varrho} + \frac{1}{2}\alpha v_2^2 + gz_2 + gh_l + \frac{Q_{dot} + W_{dot}}{m_{dot}}$$

where (1) refers to the location of compressed air in the pancake compressor and (2) refers to the intubation tube. Because there is no work being done, the last term can be eliminated. Additionally, since z_1 approximately equals z_2, those terms are cancelled as well. The velocity of air in the compressor can be eliminated as well, assuming that the volume of the pancake compressor is much larger than the volume of air exiting the compressor. The head loss term (h_l) is equal to $\frac{kv_2^2}{2g}$, where k is an experimentally determined value based on the valve type and how far open it is. For a ball valve, the values have been experimentally determined and documented and are displayed in the table below.

Valve type	Percent open	Servo position	k factor
Ball valve	100%	90	0.5
Ball valve	66%	60	5.5
Ball valve	33%	30	200

Table 4.11: Experimentally determined k factor values from [14]

4.4 Results

The final program was able to effectively integrate the control loop with our basic model of the lung system, and transmit the resulting data to the ESP32. In this process, we were able to gain insight into the physical effects of tuning PI gains, as well as visualize the effects of varying compliance and airway resistance. Furthermore, the program was able to accomplish this for several breathing cycles and able to account for real-time manual changes to the compliance, tidal volume, and breaths per minute. These results are shown above in the controls graphs, as well as in the user interface.

Despite its success, the results of our simulated mandatory ventilation mode were limited in several degrees, leaving space to continue improving upon the current model. Due to the rudimentary nature of the respiratory system model, we were unable to account for non-linear and non-constant natures of the airway resistance and lung compliance, hampering the accuracy of the lung pressures calculated. A further improvement, would be to broaden the applicability of the simulation by including an assisted ventilation mode. By advancing the accuracy of the lung and expanding the modes of operation, the simulation would be a more powerful tool in providing a free means of understanding ventilator interactions with a variety of patient types.

Chapter 5

User Interface [Sam, Matt, Joe]

5.1 Vision

How can we make it easier for carers of patients to interact with ventilators? This was the question that dominated the work of our User Interface (UI) sub-team. Implementations of ventilators with an individual display require a carer to be present at a specific ventilator to view the data and interact with it directly. One of the key challenges that COVID-19 poses is its high risk of infection to these carers that have to engage directly with a patient's ventilator. In acknowledging these risks, we were motivated to:

1. Design an interface that reduces the exposure of carers to patients with COVID-19 and thus reduce their risk of infection
2. Build a system with fewer components (i.e. a system that does not require a screen for each individual ventilator)

We envisioned a system that could receive data from several ventilators at once, and display it seamlessly in real-time on a mobile display. Such an approach would be beneficial to carers in the various pop-up hospitals created to control the pandemic, like the NHS Nightingale Hospital, London, where the vast amount of patient beds under one roof makes it difficult and risky to check each individual ventilator. This system could offer support for push notifications and hands-free parameter changes on the mobile device, prompting the carer to take action on any alarming data received from a specific ventilator, and allow updated ventilator values to be transmitted back.

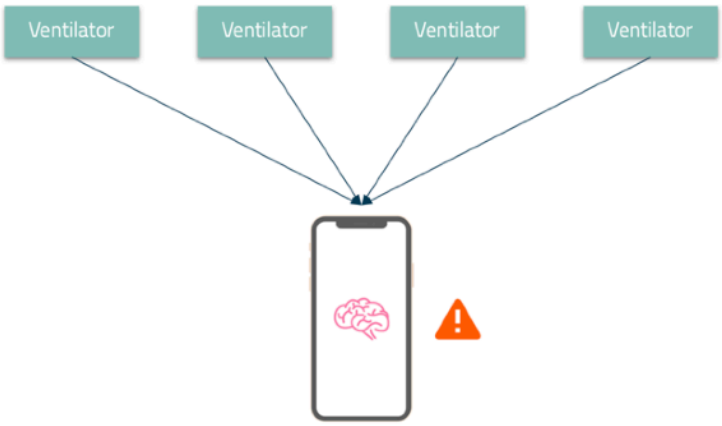

Figure 5.1: Diagram of multi-input single-display system with support for push notifications

5.2 Goals

For this project, we took our vision and focussed on achieving three fundamental tasks:

1. The ability to receive data from a single ventilator
2. The ability to view this data in real-time on a webpage
3. The ability to input and transmit new data to a single ventilator

Achieving this subset of goals would provide us with a proof-of-concept for our new interface. Ultimately, scaling to more ventilators and incorporating push notifications are incremental improvements that depend fundamentally on the three goals we laid out for ourselves above.

5.3 Design

The brain of our system was the ESP32 microcontroller. In addition to its low cost, the ESP32 came bundled with WiFi and Bluetooth capabilities, making it a great choice for our system [8]. Figure 5.2 provides an overview of our system.

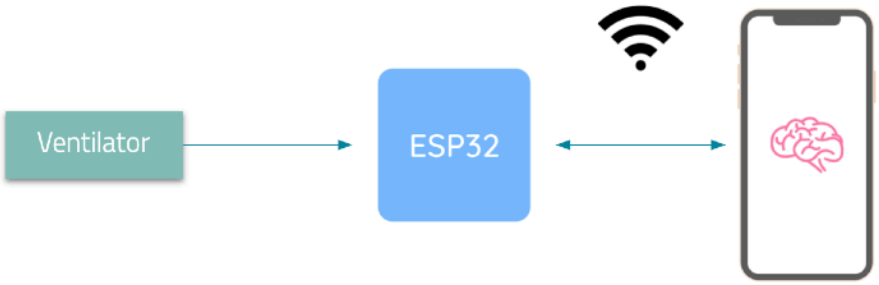

Figure 5.2: An overview of the design of our interface system

5.3.1 Receiving Data

We used the ESPAsyncWebServer.h and WiFi.h libraries to host a server on the ESP32 [5, 2]. As depicted in Figure 2, we opted to use WiFi in place of Bluetooth as we believed that WiFi would support our system to scale to more ventilators with less hassle, in accordance with our vision. Our client-side program retrieved data from the pressure, flow, and volume sensors in real-time using AJAX and HTTP_GET requests to the /data route of our server which responded with a simple text string containing pressure, flow and volume.

5.3.2 Viewing Data

We were inspired by the display of the Maquet Ventilator (Figure 5.3). Their three-graph structure is a visually appealing way to simultaneously understand the different incoming data, while also presenting numerical values on the side column to be easily digested.

Figure 5.3: The display of the Maquet Ventilator [10]

Thanks to AJAX, our client-side program received sensor data in real-time. We used the Plotly.js library to graph this sensor data, and performed simple front-end calculations to update the time axis in order to depict the continuous stream of incoming data as well as derive further information such as peak pressure, respiratory rate etc. [20]. In addition to graphing, we elected to present numerical values along a side column, similar to the one in Figure 3. We used the Bootstrap library, not only to present our data in a neat grid layout, but also to make our web page mobile-responsive [4]. The data was easily viewable by simply navigating to the URL address of the ESP32 server.

5.3.3 Inputting Data

We added a form field along the bottom row of the webpage that contained three input fields and an update button. We provided support for updating the PEEP value, respiratory rate, and tidal volume. As a safety precaution, our front-end validated the inputted values, thus ensuring that only appropriate changes could be made. For example, it is not possible to enter a respiratory rate

value of zero. We return feedback to the user to make them aware of what the appropriate data entries for each input field are. Upon pressing the update button, the values entered are retrieved by the server using the /update route, and used to update the desired values used by the controls code to implement feedback. The graphs then refresh, and begin to display the sensor data for the new updated values.

5.4 Final Product

The final webpage is a culmination of the advances we made towards achieving the three goals that we had set out for ourselves.

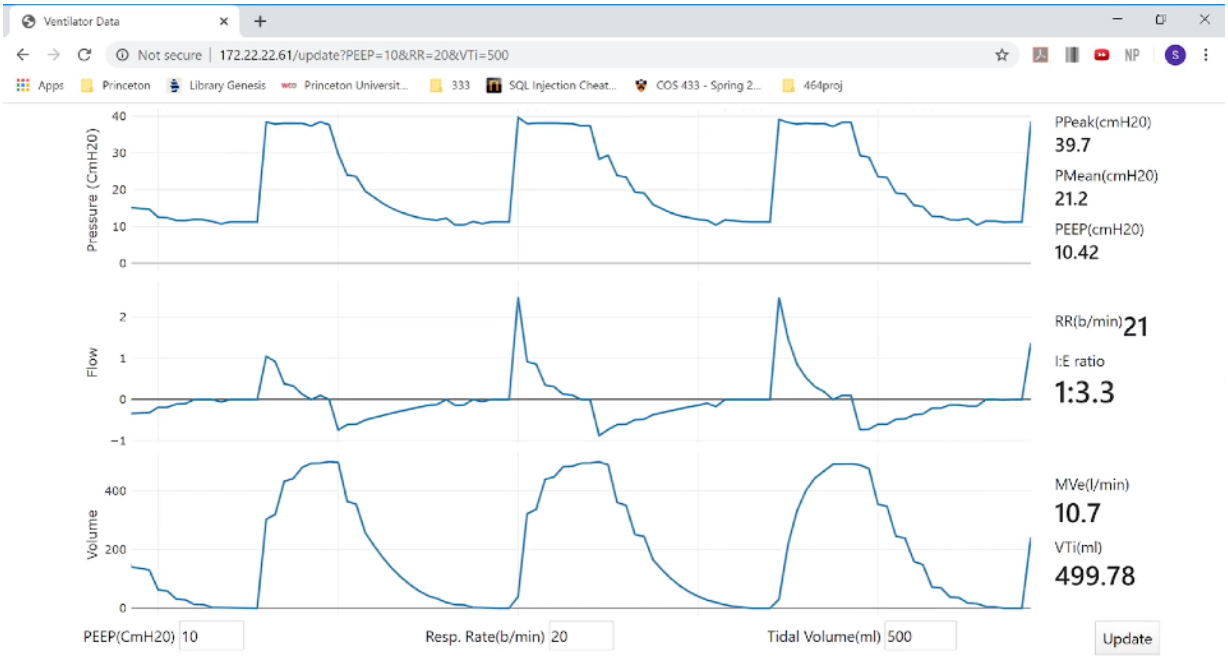

Figure 5.4: A screenshot of the final webpage designed and created by the UI sub-team, as viewed on Chrome

Figure 5.4 depicts the successful integration of our UI system with the lung simulation as viewed on our webpage on the Chrome browser. We succeeded in demonstrating our ability to read and display input data in real-time, as well as our ability to transmit parameter changes. We are excited by the positive implications that adopting such a UI could have on the health of carers, their response time for patients, and the overall cost of a ventilator.

Chapter 6

Conclusion and Next Steps

6.1 Hardware and System Integration [Harsh]

Though we did create a physical prototype, it had many shortcomings and lacked quite a few parts. Moving forward, we envision several improvements to our physical design. Firstly, our current physical setup should be connected to controls via pressure sensors. In doing so we'd be able to physically realize the Pressure Regulated Volume Control (PRVC) system that currently works with the lung simulation.

We intend to calculate the resistive pressure of the entire system, and add an extra pressure valve at the end of the expiration valve, in order to guarantee a PEEP pressure, in the event that the control algorithm fails.

The design for the servo-controlled proportional valve that we 3D printed out had a few issues with the fitting. We had to resort to wiring the valve handle to a cut servo blade, and hot glue the valve down to the stand. If we had more time, we would've iterated on the design to enable a seamless fitting.

Currently our condensate trap is just a piece of curved tubing that enables the moisture in the expired air to settle by gravity. It doesn't allow a medical professional to empty the trap, nor does it have a mechanism to prevent the moisture from evaporating back into the air. A future design must redesign this component to address the mentioned non-idealities.

The quantity of airflow through a ball valve isn't exactly linear with the angle at which the handle is turned. Moreover, the servo controlling this handle has limited precision. These non-idealities may make it difficult to achieve exact flow values. While this may be sufficient for emergency use, we would like to emphasize that an ideal ventilator should replace this setup with a certified proportional control valve.

Some members of our team couldn't connect the ESP32 to WiFi, while others could seamlessly. We believe that it shouldn't take too long to resolve this issue, allowing us to display data collected in real-time through the web-server.

6.2 Infection Control [Kal, Lap]

Another component of the ventilator that should be mentioned is controlling infection. As this device will be used to assist patients suffering from an airborne pathogen it is crucial to ensure the virus is controlled. The infection control subsystem is supposed to help prevent the patient from contracting diseases while being on a ventilator, such as ventilator associated pneumonia (VAP), and to prevent transmission of diseases from the patient to other patients and other health workers. In order to achieve these goals, significant research was put into the implementation of filters, humidifying devices, and secretion suction systems. Due to the COVID-19 pandemic, we were not able to have access to equipment that would allow us to test the efficiency of an infection control subsystem in relation to the entire ventilator. As a result, an actual prototype was never attempted.

Including a humidifier in the system is essential not only to improve patient comfort but also to improve patient compliance with our system. Hence, we researched various methods to provide the intubated patient with warm comfortable moisture in a feasible and economic way. These variations included decisions between both active devices that direct the air passages through or around a container of water, and passive heat moisture exchanges (HME) that capture moisture exhaled by the patient and reintroduce it into the air that is inhaled back in. Although HME's were the cheapest option, they required frequent replacing and in some cases could block the airway to the patient. Therefore we decided on using a passover, active humidifier that was low maintenance, and had little risk of malfunctioning to block passages or unnecessarily increase resistance in the tubing.

Filters are an essential part in an infection control system because they help prevent disease-causing pathogens from being transmitted to other patients and essential health workers. Given the specific focus on COVID-19, we want to prevent the transmission of the highly infectious coronavirus. However, these particles are incredibly small; they range from 0.06 to 0.14 μm [21].

Based on HEPA specifications, the strictest filter filters 99.97% of all particles up to 0.3 μm [28]. Although based on numbers alone, these filters are not good enough to guarantee blocking the transmission of the coronavirus. Nonetheless, it is one of the most efficient technologies and using it is better than not using anything at all.

6. References

[1] AFP News Agency. *Coronavirus: Belgian Hospital Converts Snorkeling Masks into Emergency Ventilators*. https://www.youtube.com/watch?v=n6qQLsUa7jk.

[2] *Arduino - WiFi*. https://www.arduino.cc/en/Reference/WiFi.

[3] br3ttb. *Br3ttb/Arduino-PID-Library*. 2011. 2020. *GitHub*, https://github.com/br3ttb/Arduino-PID-Library.

[4] Mark Otto, Jacob Thornton, and Bootstrap. *Bootstrap*. https://getbootstrap.com/.

[5] Dev, Me No. *Me-No-Dev/ESPAsyncWebServer*. 2015. 2020. *GitHub*, https://github.com/me-no-dev/ESPAsyncWebServer.

[6] *Engineering Specifications*. https://simulation.health.ufl.edu/technology-development/open-source-ventilator-project/engineering-specifications/.

[7] ericinventor. *Simple DIY Servo Valve - 3D Printed*. https://www.youtube.com/watch?v=S8OJ38VGYcc&t=34s. Accessed 12 May 2020.

[8] *ESP32-DevKitC V4 Getting Started Guide - ESP32 - — ESP-IDF Programming Guide Latest Documentation*. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/get-started-devkitc.html.

[9] Galbiati, C., et al. "Mechanical Ventilator Milano (MVM): A Novel Mechanical Ventilator Designed for Mass Scale Production in Response to the COVID-19 Pandemic." *ArXiv:2003.10405 [Physics]*, Apr. 2020. *arXiv.org*, http://arxiv.org/abs/2003.10405; https://mvm.care/.

[10] Getinge. *Maquet Ventilator "The ABCs of Ventilator Modes."* https://getinge.training/DesktopModules/Documents/ViewDocument.aspx?AddToLog=1&DocumentID=1903.

[11] hackadayrex. *Ventilator Design by Hackadayrex*. 2020. 2020. *GitHub*, https://github.com/hackadayrex/ventilator.

[12] *Low Cost Prototype Ventilator Rex (V-Rex), Interview with Development Team Member Stephen Robinson | MakeHaven*. https://www.makehaven.org/event/low-cost-prototype-ventilator-rex-v-rex-interview-development-team-member-stephen-robinson.

[13] "MedCram - Medical Lectures Explained Clearly." *MedCram - Medical Lectures Explained Clearly*, https://www.medcram.com/users/sign_in.

[14]"Minor or Dynamic Loss Coefficients for Pipe or Tube System Components." *The Engineering Toolbox*, https://www.engineeringtoolbox.com/minor-loss-coefficients-pipes-d_626.html

[15] "MIT E-VENT | Emergency Ventilator Design Toolbox." *MIT E-Vent | MIT Emergency Ventilator*, https://e-vent.mit.edu/.

[16] *Open Source Ventilator Project*. https://simulation.health.ufl.edu/technology-development/open-source-ventilator-project/.

[17] "Particle Physicists Design Simplified Ventilator for COVID-19 Patients." *Princeton University*, https://www.princeton.edu/news/2020/04/09/particle-physicists-design-simplified-ventilator-covid-19-patients.

[18] *Peak Inspiratory Flow - an Overview | ScienceDirect Topics*. https://www.sciencedirect.com/topics/medicine-and-dentistry/peak-inspiratory-flow.

[19] *Peak Inspiratory Flow - an Overview | ScienceDirect Topics*---. https://www.sciencedirect.com/topics/medicine-and-dentistry/peak-inspiratory-flow.

[20] *Plotly: The Front-End for ML and Data Science Models*. /. Accessed 12 May 2020. *Prototype Ventilator Rex (V-Rex), with Stephen Robinson*. https://www.youtube.com/watch?v=W7qgWOxhVyY.

[21] Robertson, Paddy. "Can Masks Capture Coronavirus Particles?" *Smart Air Filters*, 4 Feb. 2020, https://smartairfilters.com/blog/can-masks-capture-coronavirus/.

[22] Services, Engineering IT Shared. *Illinois RapidVent*. https://rapidvent.grainger.illinois.edu/index.asp.

[23]"Servo Valve from Ball Valve + Hobby Servo." *MakerBot Thingiverse*, https://www.thingiverse.com/thing:1659407

[24] "Specification for Rapidly Manufactured Ventilator System (RMVS)." *GOV.UK*, https://www.gov.uk/government/publications/specification-for-ventilators-to-be-used-in-uk-hospitals-during-the-coronavirus-covid-19-outbreak/rapidly-manufactured-ventilator-system-rmvs.

[25] UBC Critical Care Medicine. *Mechanics of Ventilation*. http://www.ubccriticalcaremedicine.ca/rotating/material/Lecture_1%20for%20Residents.pdf.

[26] "UF Researchers Lead the Way in Rapidly Designing, Building Low-Cost, Open-Source Ventilator." *UF Health, University of Florida Health*, 25 Mar. 2020, https://ufhealth.org/news/2020/uf-researchers-lead-way-rapidly-designing-building-low-cost-open-source-ventilator.

[27] “University of Minnesota Is Going ‘full-on MacGyver’ against COVID-19.” *Star Tribune*, https://www.startribune.com/university-of-minnesota-is-going-full-on-macgyver-against-covid-19/569000032/.
[28] US EPA, OAR. “What Is a HEPA Filter?” *US EPA*, 19 Feb. 2019, https://www.epa.gov/indoor-air-quality-iaq/what-hepa-filter-1.
[29] “Venturi Tube Spirometer.” *Instructables*, https://www.instructables.com/id/Venturi-Tube-Spirometer/.
[30] “Watsaig/Pressure-Regulator.” *GitHub*, https://github.com/watsaig/pressure-regulator.