RespiraWorks

Design Overview Document

Abstract

Most low-cost ventilators tackle the 'bridge' problem—how to keep a patient alive for no more than 6-8 hours of closely monitored care until they can be transitioned to a full ventilator. However, many COVID-19 patients (and patients in general) require ventilators that can do more. They need this regardless of whether they are in a hospital in a wealthy country or a lower resource one. We set out to design a ventilator at significantly reduced cost, but still designed to comply with international standards, capable of providing lung-protective adaptive ventilation strategies, clinical user feedback, and reliably uninterrupted operation for days to weeks.

This document provides an overview of the RespiraWorks ventilator. It is an overview of the theory of design as well as the pneumatic, electrical, and software implementation. This document is not a requirements document or specification document. That document (01-3 System Requirements) is the source of up to date requirements. This document is provided as a supplement in order to explain why certain decisions were made, and what factors were weighed in the decision.

Release Information

Approval	**Revision**	**Revision Date**
Ethan Chaleff	A	6/21/20

Contents

Design Motivations

The RespiraWorks ventilator has been developed to provide essential features with an open-source design, an IEC-conformant quality process, and a bill of materials optimized for sourcing and manufacture worldwide. Middle- and low-income countries are likely to deal with COVID-19 for longer, with less access to healthcare, and a larger deficit of medical devices.

Rendering of the v0.2 Release of the RespiraWorks Ventilator

As life support equipment, ventilators must have a high-quality design, with design controls, thorough review, and a well documented verification plan. Our design is developed under an ISO 14971 process, and our software under ISO 62304. We will qualify the device under ISO 60601-1 once the design has reached a sufficient level of maturity. All of these process and design controls are critical to the successful design of a product that will be responsible for a human life. Conforming with existing quality standards facilitates the regulatory approval process for our initial deployment.

We approached the design by exploring the medical need for ventilators and learned some features cannot be sacrificed in a ventilator that will have lasting impact for communities in need of advanced life support equipment. However, there are numerous avenues to reduce cost through tuning the performance specification to targeted use cases, specifically the ability to ventilate high resistance, low compliance lungs at small tidal volumes and high respiratory rates.

Our design can significantly reduce the delivered cost of such a device by 1) reducing some performance requirements 2) developing software under a crowd-sourced model 3) not charging licensing or royalty fees.

The design has focused on a pneumatic implementation, where possible trying to avoid mechanical complexity (bellows, actuators, etc), placing more of the burden on electrical and software design. This is informed by our experience that deep expertise in electronics is more globally prevalent than similar expertise in mechanical engineering. The device is intended to be manufacturable by consumer electronics, automotive, refrigeration, or other similar manufacturing entities near to the point of use. This approach allows flexibility in selecting manufacturing partners who are supported from the engineering side by our globally distributed team of volunteer engineers and manufacturing-quality professionals. To enforce this process, the design strives to identify solid supply chain alternatives for each component and choose components that are easily sourced in our targeted regions of use, including Guatemala, India, Kyrgyzstan, and Nigeria.

The custom case, sensors, and valves are designed to use injection molded, laser cut, or sheet metal folded parts that then require only basic tools to assemble, making them amenable to any number of manufacturing facilities around the world. The chosen purchased components are readily available through global markets, including automotive supply chains (e.g. valves and pressure sensors), factory/warehouse supply chains (e.g. blower, tubing and pipe fittings). The control and graphics board is designed around a custom PCB, but assembled using hand-solderable components if a turnkey PCB supply chain were not available (though practically this would likely not be feasible from a QA perspective).

Besides a detailed user manual and maintenance plan, the device also comes with a robust self-test mode. This ensures that produced ventilators achieve necessary safety levels before they are used with patients and will allow users to check the health of the device over its lifetime. A cryptographic hash will be released with the approved software, and allow local manufacturers to be confident in the software they are loading to the device.

High Level Design

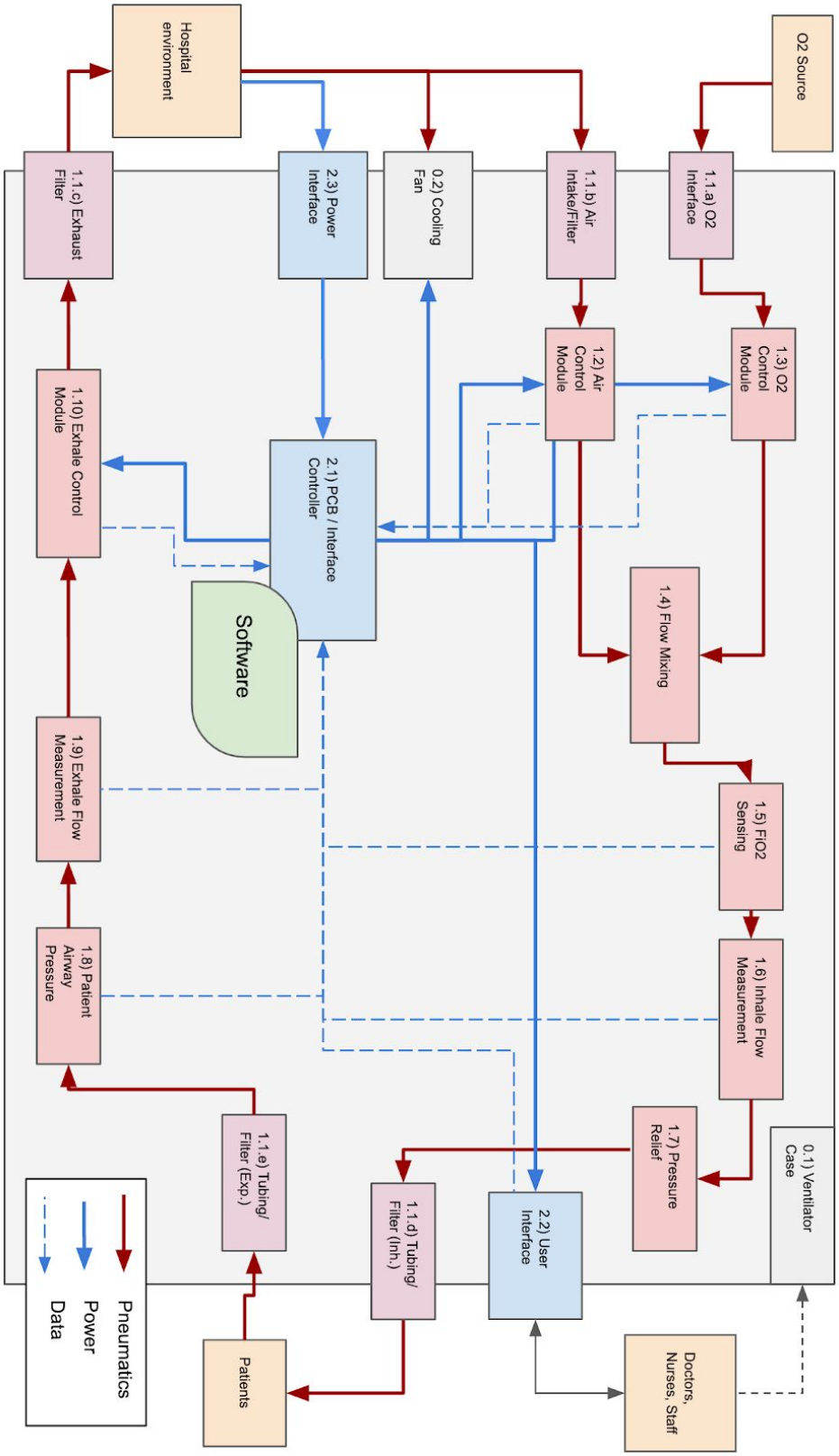
The RespiraWorks Ventilator has four major subsystems with tightly coupled interfaces:

1. a pneumatic system,
2. an electrical system,
3. a software system, and
4. a mechanical assembly to enclose the device and provide structural support.

The diagram below shows the conceptual interactions between the pneumatic and electrical systems, with software running on both the main controller (STM32) and a second independent graphics computer (Raspberry Pi 3+/4) controlling the User Interface.

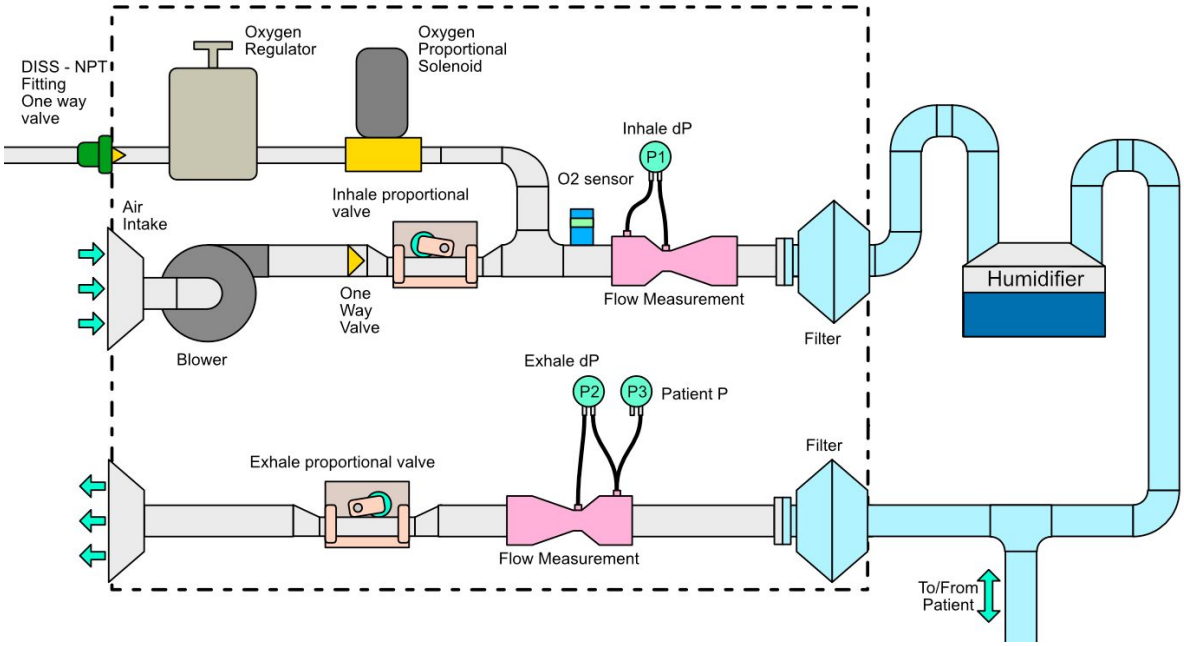
The following sections walk through this in more detail and explain the basis for the design selection of various components.

Figure 0: Ventilator Functional Block Diagram



Pneumatic Design

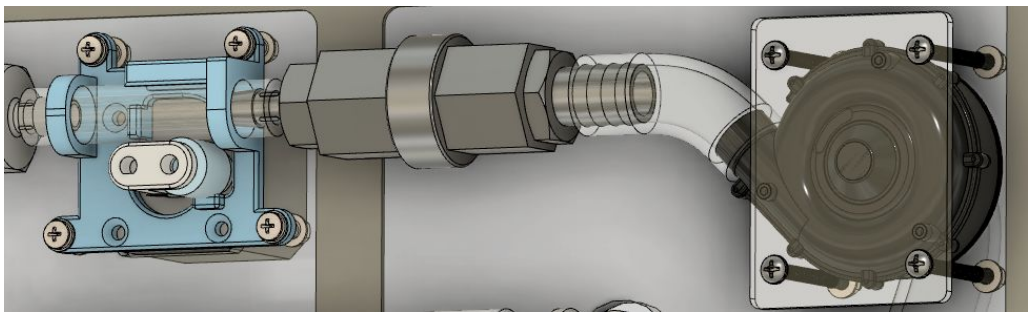
The figure below displays the pneumatic layout of the system.



Overview

Oxygen is supplied from an external pressurized source to the oxygen port. Air is drawn in by the blower from the ambient room air. The device can also operate without an external supply of pressurized oxygen in the event of supply shortages, (delivering only 21% FiO2).

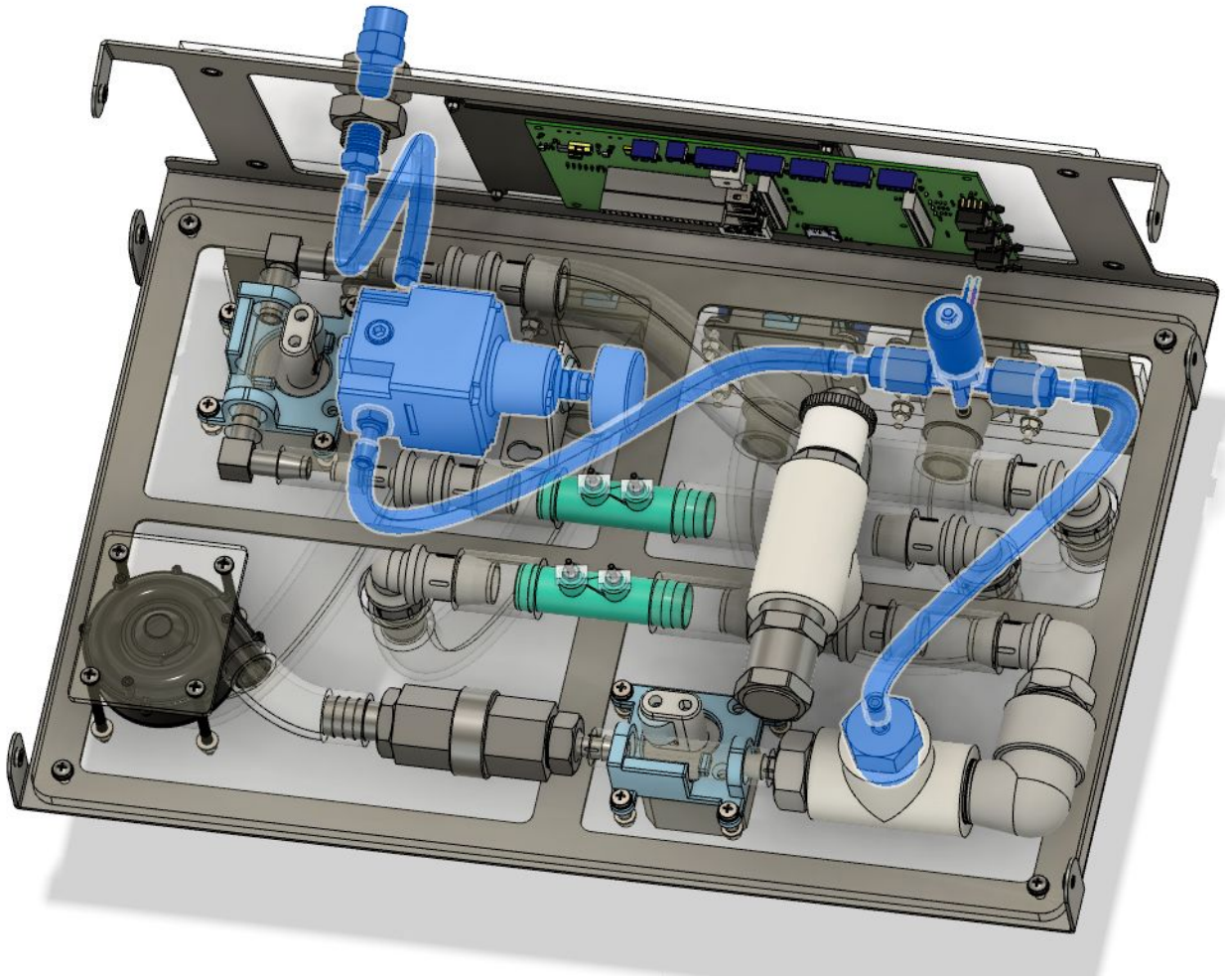
Air Path



Air is drawn in by the blower through a replaceable HEPA filter and pressurized for use within the system. The blower maintains a relatively constant speed and is open-loop controlled. The capability to control the fan speed to conserve power and increase response time has not been incorporated in the current release. After leaving the fan, the pressurized (5 kPa) air passes

through a check valve included to prevent oxygen backflow into the blower, as the blower is not rated for oxygen duty and could present a fire risk[1]. Following the check valve, a proportional pinch valve regulates the downstream airflow. It does this by adjusting the cross sectional area of the flow path in the tubing built-in to the pinch-valve.

Oxygen Path

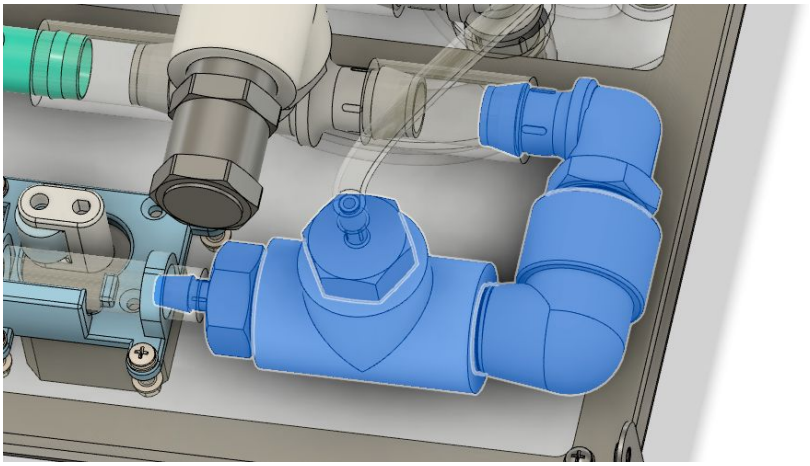


The oxygen circuit is fed from an external oxygen source, with a built-in regulator to step the pressure down to a pressure low enough to allow it to be controlled by a 12V proportional solenoid (PSOL). The design oxygen pressures at the inlet are 440 kPa - 120 kPa. The highest pressure connection that is allowable is set by the capability of the regulator, which cannot be used at a pressure higher than its rated pressure. The minimum pressure is set by the orifice size in the PSOL. With a supply pressure below the PSOL minimum, it cannot generate sufficient flow to the patient.

[1] For a more complete discussion of risk and hazards, refer to 01-07 Hazard Analysis Risk Assessment

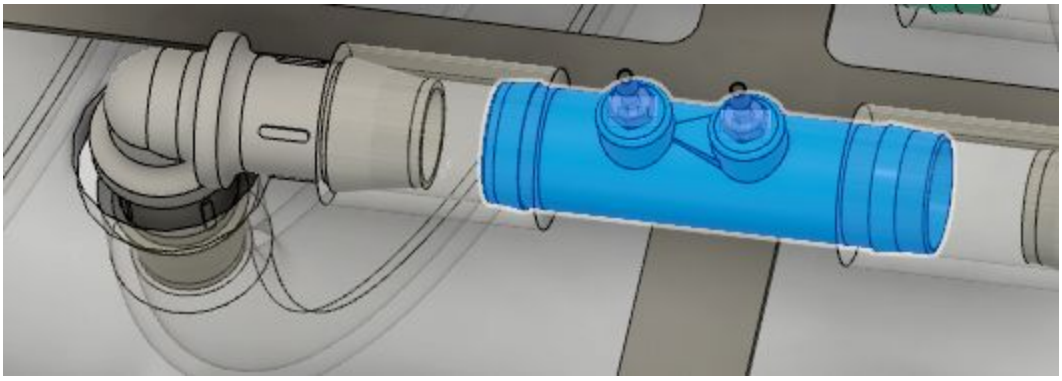
A check valve prevents contamination of the hospital oxygen system. Oxygen flow to the patient is controlled by the PSOL, whose orifice size is controlled by a variable pulse-width signal. Currently, the PSOL is a small automotive solenoid, which is not qualified for oxygen duty, though it is functionally suitable for development. Identifying a more reliable oxygen injection method is a high priority for improving the design. It is the understanding of RespiraWorks that the current PSOL is capable of being oxygen cleaned, and it enables fine control and rapid development (assuming) they can be sourced. As such, it has been included in the design.

Oxygen and Air Mixing



The oxygen and air streams are mixed in the patient tubing without a dedicated oxygen blender; instead, the oxygen blending function is provided by proportional control of the two input streams. A galvanic cell oxygen sensor is used to measure the oxygen content as the gasses mix and provide feedback to the controller to calculate, display, and enable closed-loop control of FiO2.[2]

In-Flow Measurement



[2] *Note that as of the v0.2 release, oxygen blending has not been fully implemented, with the only two modes available (FiO2 = 0.21 and 1.0). Being able to achieve fine control (+/- 0.05 FiO2) is a target for the beta/v1.0 release. More information on this can be found in 01-2 Progress Status Report.*

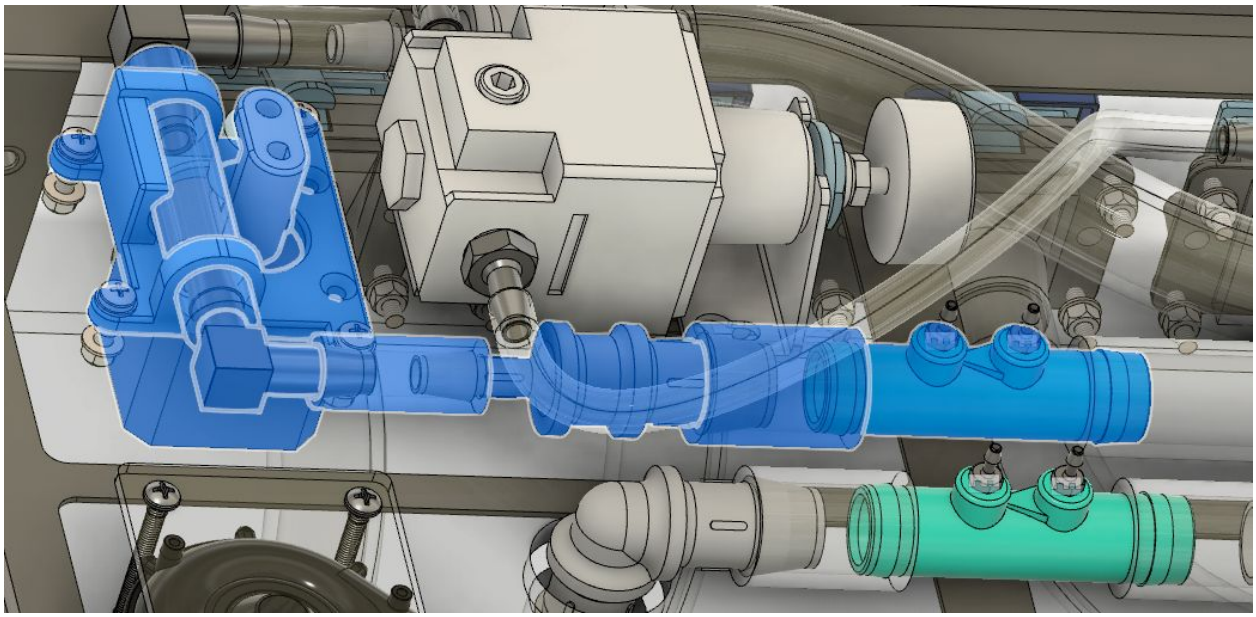
After mixing, the combined gasses pass through a venturi differential-pressure flow sensor, where the flow rate of gas delivered to the patient is measured. This gas passes a medical-grade anti-viral filter upon exit from the machine for protection against viruses, bacteria, or other foreign material. If these filters are not available, the device also includes the capability to add an inline HEPA filter in the panel mount for the tubing. This feature was included due to the difficulty sourcing medical viral filters and the desire to support additional options.

The patient circuit utilized by the ventilator is an ISO 5356-1 standard 22mm female dual limb circuit which interfaces to male ports on the ventilator. If a humidifier, HME, or other device is used, these are connected to the patient inhale or exhale circuit, as appropriate.

Expiratory Limb

On the expiratory path, the exhaled gas returns to the ventilator via a medical-grade anti-viral filter. This filter captures droplets exhaled by the patient. This reduces the possibility of machine contamination and dispersal of aerosolized virus from escaping into the local environment. The expiratory flow rate is measured by a flow meter similar to the one used on the supply side.

PEEP Control

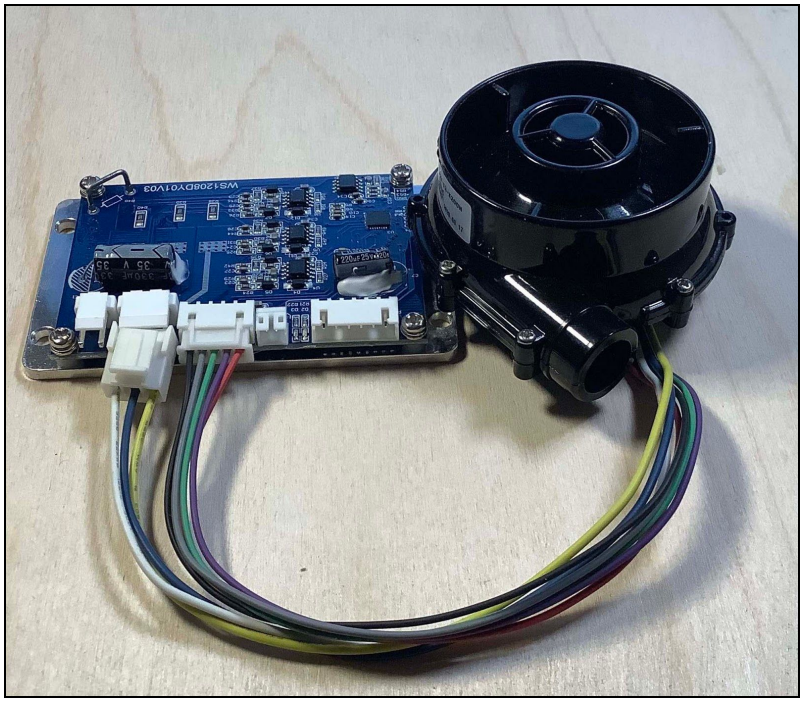


Following exhale flow measurement, another proportional pinch-valve is used to set PEEP.

Air exiting the machine is processed through a final HEPA filter to provide an additional safety factor for staff protection.

Fan Selection

A key decision was made early on to use a blower (rather than a pressurized oxygen source and control valve), to deliver patient air.



WS7040 fan and driver board

The rationale for use of a blower in general, and this blower in particular, had a few bases. First, using a blower (rather than compressed air) allows the device to provide emergency ventilation with only a relatively small power supply (around 50W continuous). This allows the device to be portable and makes the device amenable to use in field hospitals or mass casualty incidents where providing compressed air or compressors can be challenging at scale. The blower selected is also a common CPAP blower rather than a high performance ventilator unit.

This allows the use of an easily achieved blower performance specification, reduced cost and increased supply chain resilience.

However, selection of a CPAP blower does come with drawbacks—the blower is not rated for oxygen duty and the pressure response of the blower is slow, with a time constant some 1500 ms longer than comparable blowers in ventilators. This led to the development of a high-flow, long-life proportional valve, discussed in a subsequent section.

By placing oxygen mixing after the blower and using a check valve to prevent backflow, the design ensures that the blower will never see high oxygen concentrations.

Additionally, based on medical feedback, the delivered patient pressure is no more than 60 cm H20. It is common for ventilators to specify higher in order to ensure adequate ventilation for

highly resistive airways . Adopting this lower pressure spec allows the ventilator to provide a pressure range that is most often needed, while avoiding significant cost. It is compliant with 80601-2-12 and all specifications reviewed for Covid-19 ventilators.

Dual Limb Patient Circuit

The ventilator uses a dual limb patient connector (i.e., flow is sent to the patient in one tube, and returns in a second). The flow is always one way, starting from the device going to the patient, and returning in a second limb. The other option is a single limb design, with an exhale valve at the patient side.

A dual limb design was selected to allow for mechanical PEEP control, as discussed in the next section.

Note as well that there is no check valve between the mixing volume and the patient. Backflow is prevented through two means. First, there are two check valves upstream of the tee, and no exhaust ports between the tee and the patient, so gas can only flow back into the ventilator as much as compressibility allows, which is not significant. Second, even when fully closed, the inhale proportional valve allows a small leakage (bias) flow through the device, ensuring that, as long as the device is on, flow should never enter the ventilator through the inhale port.

In a survey of suppliers, very low-cracking check valves (i.e. duckbill valves) that were available in an inline configuration were not common. The one that was tested had significant flow resonances at low velocities which interfered with volume sensing. The present approach uses no medical check valves. Normal (i.e., high-cracking) check valves can be used on the air and oxygen lines.

PEEP control

One of the differentiating features of the RespiraWorks pneumatic circuit is the use of a controlled mechanical valve to maintain PEEP, rather than a traditional passive PEEP valve. This decision was made early on for several reasons. First and foremost, early in the pandemic, PEEP valves were essentially unavailable and so they were not available for prototyping, but also implied they might not be available for some time. PEEP valves are effective, and so the option to design a new PEEP valve, or base a valve on other open source designs was explored. However, calibrating the spring constants proved to be a challenging manufacturing process. Generally these types of passive PEEP valves are considered disposable and so don’t experience long term use. This led to concerns about long-term use of these PEEP valves as the spring constants change with time/wear, motivating our incorporation of the controlled solution.

Incorporating a controlled mechanical valve also provides the ability to fully close the airway during inhale, thus preserving oxygen and improving the response time of the inhale circuit.

Finally, the use of a controlled valve allows the device to adjust settings based on user input, rather than requiring staff to mechanically adjust the ventilator.

A fail-open valve also prevents the need to include a separate valve for anti-asphyxia protection, though this is still being evaluated with respect to the hazard assessment..

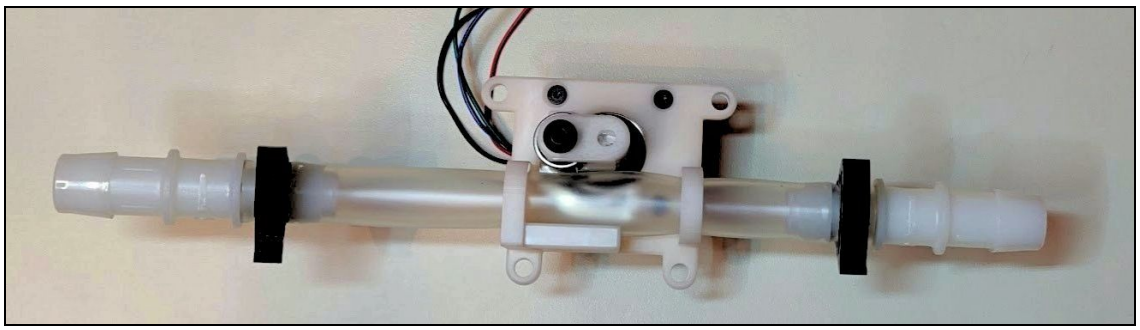
The separate question has been raised of why not to use a pilot actuated valve, either using patient pressure or a small proportional solenoid. This was avoided entirely for sourcing considerations and was decided early in the pandemic; this could be re-evaluated for future iterations. In general, the pilot operated valves, such as those from Philips and Galemed, were deemed a risk for countries without existing supply chains to those manufacturers. The device seemed difficult to safely re-engineer.

Custom Proportional Pinch Valves

The design process for the entire ventilator started with a desire to avoid custom components at all costs. However, after an exhaustive search of valves, control methods, fans, and spare ventilator parts, it was decided that a custom valve was the only option that would allow the design to achieve targets on cost, supply chain, and features for expiratory flow control. This decision was undertaken considering the significant quality and qualification burden required for custom parts.

A rapid pressure swing on the inhale is required to provide ventilation to low compliance lungs at a high respiratory rate, which was to that point impossible using closed-loop control of the fan alone.

The design originated by trying to tackle the PEEP problem as discussed above. A design was specified in order to maintain a variety of different PEEP levels and to avoid significant wasted flow during inhale. In the process of development, it was realized that the same design could also be used to provide a rapid pressure swing on the inhale flow, in addition to exhale.



One of two pinch valves in each ventilator (early model)

The general operating principle of the pinch valve is to use a high-torque stepper motor to rotate a cam/rotor onto peristaltic pump tubing. The cam pinches the tubing against a platform affixed to the base. Other than the interior tubing, the valve assembly never comes into contact with the gas stream. The highest wear portion of the design is the tubing itself, which can be replaced for less than 1 USD when it must be replaced.[3] Life-leader testing is currently underway to demonstrate the ultimate lifetime of the valve; though it will take time to accumulate. As of submission of this proposal, the valve has been running at a 200% duty cycle for 21 days continuously. Further information on life leader testing can be found in the 01-02 Progress Status Update.

An additional advantageous feature is that the valve is closed through a stepper motor which does not maintain its state when power is lost. The elastic modulus of the tubing, coupled with the fact that the rotor can not reach a “locked” position is sufficient to force the valve open if the power fails—the valve is normally open, and allows the exhale valve to also function as an anti-asphyxia pathway.

The pinch valve itself is built from relatively simple components, discussed in more detail in 05-01 05-01 Production Methods for Custom Components.

Proportional Solenoid

Proportional solenoids (PSOLs) are used in many applications (including other ventilators) for precise, rapid control of high pressure gases. The major factor arguing against the use of a PSOL

[3] One option explored, not currently implemented, is to provision the device and arrange the components with an extra length of tubing (not shown above) such that it can be fed through the valve over time; pushing the extra from upstream to downstream of the valve and changing the wear point. This maintenance operation will extend valve lifetime significantly and should be able to be performed by hospital staff in addition to maintenance technicians.

in the RespiraWorks ventilator is that PSOLs with applicable characteristics are, almost by definition, produced for ventilator use, with an associated limitation on cost and availability. While some are available outside this market, those are typically not designed for use with high-pressure oxygen. High pressure oxygen requires real safety considerations, as the opportunity for fires can occur from improper lubricants, components, or errant sparks.

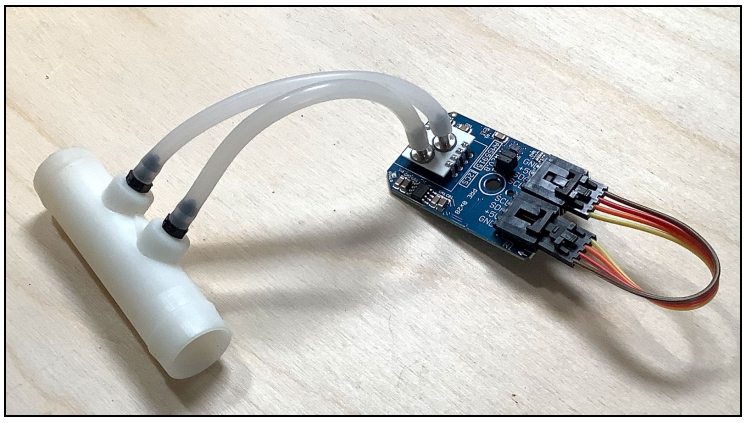
All that said, from the options considered and reviewed, a properly designed and sourced proportional solenoid is the cheapest, most reliable method for controlling high pressure gas flows, provided the flow rates and pressures are modest. There is still interest in improving the oxygen circuit to use a high-pressure rated pinch valve, perhaps in combination with a 2-way (rather than proportional) solenoid as part of a future cost optimization.

In the near term, a proportional solenoid was selected in order to facilitate development rather than tackle the high-pressure pinch valve design.

A secondary but important consideration is that in order to protect the patient from barotrauma, it is important for the oxygen valve to fail closed when power is lost, which is a feature of the selected proportional solenoid but not of the pinch valve.

Venturi-Based Flow Sensing

One of the guiding principles of our design was to avoid using proprietary equipment and avoid disposables associated with each patient. This philosophy led us to develop a custom venturi-based flow sensor. It lacks some low-order accuracy, leading to reduced tidal volume accuracy, though still within spec. In return, by producing an easy-to-sterilize sensor the device can avoid proprietary and costly proximal flow sensors.



A venturi flow sensor with a differential pressure sensor used to calculate flow

Venturi flow meters are commonly used to measure the flow of gasses by measuring the change in pressure of the gas as the area is constrained. Relative to other sensor types, a venturi is the

cheapest method for measuring flow rate, as it produces the largest signal to pressure drop ratio of any flow measurement solution.

RespiraWorks initially pursued numerous different flow measurement methods, including pneumatic methods such as pitot, pneumotachograph, or orificing flow sensors, which all require far more sensitive (and expensive) pressure sensors than a venturi. RespiraWorks also reviewed existing thermal mass flow sensors, which were not available during initial development. While they are costly, thermal mass sensors provide better low-noise signals at low flow rates and may in the long run prove to be a more reliable and robust solution. The issues with the sensor led to significant effort on software algorithms to improve the response. The effort that went into robust volume and flow integration will make it relatively easy to change to a different (better) sensor, but testing has shown the accuracy sufficient to control flow rate and estimate tidal volume within required accuracy.

For our ventilator, we designed venturis that can measure the high flow required, but that can also be made through widespread, low cost manufacturing processes (e.g. injection molding).

The venturi design was based around a garden fertilizer injector. Though all initial prototypes have been 3D printed, the required geometry is almost identical to a ½" garden venturi, which can be purchased at quantity for 2-4 USD / pc.

Venturis produce a large signal ratio. In ours, a maximum pressure drop of 5cm at 100 slpm produces a measured signal of 4 kPa, which can be measured with cheap, widely-available automotive differential pressure (dP) transducers.

In contrast, similar flow orifice and pneumotachograph style sensors require expensive amplified pressure sensors to measure signals in the range of just 100 Pa.

These dP measurements can then be converted into flow rate estimates based on the physics of the venturi. By placing one venturi on the inhale limb and one on the exhale limb, the flow rate going to the patient is estimated by subtracting the two flow measurements.

Oxygen Sensing

Based on conversations with doctors, there were two driving constraints for our oxygen sensing needs. First, the ventilator needed to be able to provide fine FiO2 control, not just a few set-points. Second, the ventilator needed to work with oxygen tanks and concentrators that were not actually 100% oxygen (e.g., improperly charged hospital oxygen tanks), since reliably pure O2 can be unavailable in many developing markets.

Based on both of these constraints, we needed an oxygen sensor to measure the actual oxygen content of the flow. We chose a galvanic oxygen sensor because they are widely available.

A downside of galvanic O2 sensors is their finite lifetimes. Therefore, we will likely add an ability to also measure oxygen flow rate, so the system is more robust to a failure of the oxygen sensor (and can sound an alarm that maintenance is required). We're exploring whether a sensor that uses fluorescence quenching can provide the requisite sensitivity and responsiveness, as this would last for the lifetime of the device.

Filtration Approach

Filtration is critical to both keeping the ventilator from being contaminated by the patient or the environment, and keeping the patient protected if the ventilator does become contaminated.

In our system this is accomplished by dual filtering on both the inlet and exhale paths, such that for each path, there is a filter between both the ventilator and the patient and the ventilator and the environment.

HEPA filters were chosen because they are widely available and designed to filter both large and small particles. The HEPA filters are held in a custom enclosure that is designed to be injection molded. It's easy to disassemble, clean, and reassemble, making it easy to change filters over time.

Overpressure Protection

To date, RespiraWorks has not been able to identify a robust, reliable method for overpressure relief. Numerous options have been tested. All of these present failure modes, or had leakage issues which were likely more significant than their pressure protection function. One valve was identified which worked well, but the price was over $100 per valve. The current design does not possess a dedicated overpressure protection relief valve. The fan specification is such that it likely cannot result in a pressure greater than 55 cm H2O. However, the PSOL can easily generate pressures in excess of 60 cm H2O. There is some leakage at high pressure through the pinch valve, which is pressure dependent but this is likely not sufficient for patient protection. Additionally, most failure modes identified for the ventilator result in the PSOL failing closed; However, this is not sufficient from a risk consequence perspective, and this is an area requiring further address.

Anti-Asphyxia Protection

The device currently does not have a dedicated anti-asphyxia protection valve. Requirements on flow and flow resistance for anti-asphyxia have been included in 01-3 System Requirements. The current anti-asphyxia strategy in the event of ventilator failure is to actuate alarms (see Software) section and to fail-open the exhale control valve, which should provide an inhalation and

exhalation pathway. It has not been evaluated if this pathway meets the flow resistance requirements identified, nor if this failure mode is appropriately addressed from a hazards perspective.

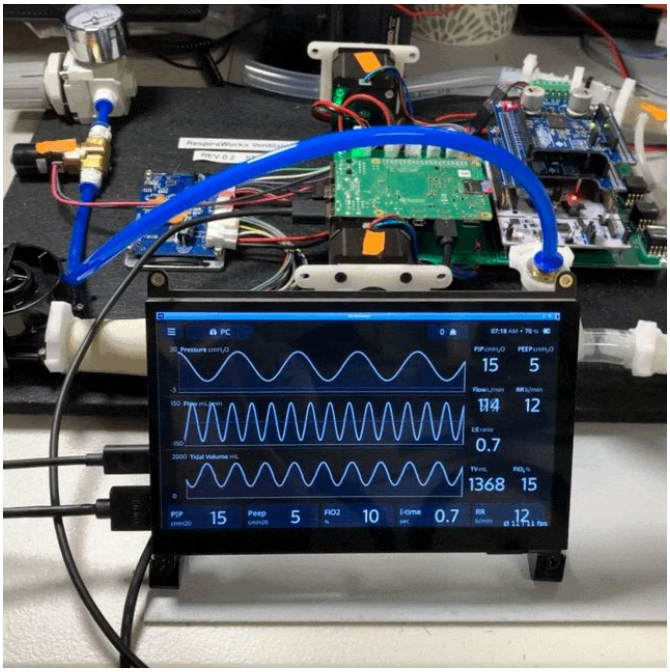
Planned Mechanical Upgrades

The RespiraWorks ventilator has developed organically over time. As the design matures, a number of upgrades are under consideration beyond the current design.

These include:

- Adding a pressure transducer prior to the patient inspiratory limb (either in addition to the one on the exhale venturi or replacing it). The benefit of placing the pressure transducer here is that if something happens to the line between the patient and the exhale portion of the ventilator (e.g. it is stepped on), the ventilator can detect such an obstruction and maintain basic safety.
- As mentioned above, adding flow sensing on the oxygen inlet circuit, to aid in FiO2 control and make the system more robust to failure of the oxygen sensor.
- Adding redundant pressure differential sensors to make it easier to detect a sensor failure.

Electrical Design



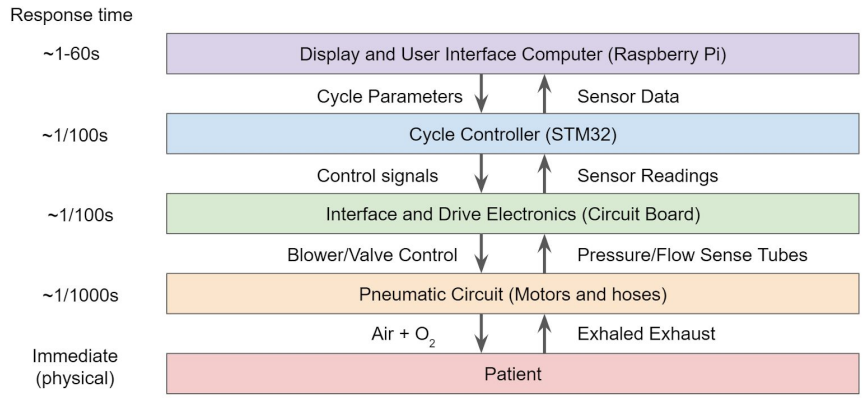
The electronics system provides power to the ventilator, real-time control of its sensors and actuators, safety protections and alarms, and a user interface for delivering information and controlling the parameters of its operation.

As with the pneumatics system, the electrical system is designed to be amenable to global supply chains, be low cost, and yet be robust and reliable.

Computing Architecture

A dual-computer architecture was selected. There is a Cycle Controller based on STM32 that handles the real-time control of the ventilator's sensors and actuators. Connected to this is a UI Computer based on a Touchscreen, Raspberry Pi 4, and Linux to provide the user interface. This design allows us to delegate the most critical functions to a simple design that can be reasonably reviewed and tested in its entirety, while also allowing enough complexity in the UI Computer to provide a comprehensive user interface.

System Block Diagram



The division of responsibility between the two computing elements is divided roughly by the immediacy of a potential failure. For the class of operations in which a potential failure could result in a hazard too quickly for reasonable human intervention, the Cycle Controller is used. For the class of operations in which a potential failure can reasonably be addressed by human interaction in a reasonable amount of time given a timely alarm, the UI Computer is used. This design also allows for self-checking between the two computing elements to mitigate single-point failures.

The STM32 microcontroller was selected because it is more powerful than Arduinos or similar popular microcontrollers, but it is still cheap, widely available, well-documented, and has extensive debugging tools. The STM32 is proven to be a reliable medical electrical component and has a diverse I/O set for pneumatic and mechanical control. The operation of the Cycle Controller is kept as simple as possible to facilitate comprehensive review, test, and verification of its function.

The Raspberry Pi was selected as the UI controller for its wide compatibility with existing human-machine interfaces and robustness to supply chain shock. The Raspberry Pi also enables rapid software and UI development by supporting a Linux-based operating system and the common HDMI interface. It supports remote monitoring, as well as automated testing, through its wired and wireless network interfaces. While powerful and flexible, the complexity of its operating system means that real-time control cannot be sufficiently guaranteed or verified. Hence all time-critical operations, such as ventilator pneumatic control, are handled by the STM32.

A serial communication link using checksums to ensure data integrity provides reliable communication of settings and data between the two computing elements.

In the event of a system crash or failure in the UI Computer, the Cycle Controller will continue to operate the ventilator with the most recent valid settings received from the UI Computer.

Each computing element has its own alarm system as well as a watchdog timer. If either computing element becomes non-responsive, the watchdogs will put it into a safe state and restart it. The UI Computer monitors the Cycle Controller and vice versa, and will issue an alarm if the other stops responding. Each controller can trigger an audible alarm system independently, and the UI controller can display alarms visually as well. The motherboard features bright LEDs in 3 colors to provide visual alarms independently of the touchscreen interface. The Cycle Controller and UI Computer each have independent control of their own set of bright red, yellow, and green LEDs. Due to an error in the motherboard design, the Cycle Controller only has command of the red LED in its set. This will be fixed in a future revision.

A custom motherboard has been designed to host both the UI Computer and Cycle Controller, as well as all the required communication, filtering, safety protections, sensors, motor drivers, and power supplies. This improves both the manufacturing assembly and the electromagnetic immunity of the electronics.

Closed-Loop Pneumatic Control

All key elements of the pneumatic system are controlled by electronics. The WS7040 blower features its own driver, which is powered by 12VDC controlled by the motherboard with a PWM signal. The RespiraWorks-designed proportional pinch valves are driven by stepper motors, which are controlled by powerSTEP01 stepper driver ICs. The powerSTEP01 is available in a daughterboard compatible with the STM32 developer module - this synergy enables rapid hardware and software development around controlling airflow, and provides a proven electronic design for integration into the next motherboard revision.

The differential pressure (dP) sensors that monitor inhale/exhale flow rates are board mounted as well. These are automotive differential pressure sensors which are cheap and widely available in large quantities worldwide. They monitor the pressure difference across the two venturi-based flow sensors as well as the delivered patient pressure. They output an analog electrical signal which is filtered and conditioned on the motherboard and delivered to the STM32 Cycle Controller. They are pneumatically connected to the venturis by 2.5mm flex tubing. In the case of the single-ended pressure measurements only one of the two pressure ports is connected by 2.5mm tubing, and the second port is referenced to atmospheric pressure.

The oxygen supply is controlled by a proportional solenoid. The motherboard features two high-current MOSFET switches fused with resettable thermal fuses and protected by flyback diodes for operating inductive loads. These switches are capable of fixed on/off control or

proportional PWM control of a 12V load. One of these switch channels is used for the proportional solenoid. The other channel is uncommitted and can be used for a future feature, such as humidifier heating control.

For future expansion of features, the motherboard also provides 4 powered I2C ports that can be configured for 3.3V or 5V operation. These can be used to command actuators, read sensors, or interface with other supporting devices such as an external humidifier or uninterruptible power supply (UPS).

Power Supply Design

An early decision during the electronics design was to standardize the input voltage to 12 VDC. This enabled rapid prototyping around the WS7040 blower and driver. Using 12VDC enables the use of widely-available sealed lead-acid security system batteries or automotive batteries as a backup or primary power source in field hospitals. 12VDC is also a common battery standard and there exist 60601-1 compatible wall adapters for operation in a more conventional medical facility.

The custom motherboard features DC/DC converters to supply and regulate the logic voltages for the on-board electronics. In the next iteration of the motherboard, backup batteries shall be present for supplying emergency shutdown alarms (when both mains and backup battery are absent or depleted) as well as providing real-time clock support for monitoring, diagnostics and logs.

Human Machine Interface

A 7-inch capacitive touchscreen is selected as the primary user interface. This is a color display capable of receiving user input and refreshing data with 50 ms latency. Both the display and the integrated speakers are operated through the HDMI interface of the UI controller.

The cycle controller has access to its own dedicated audio system. This is a simpler piezo buzzer, driven by a square wave signal and capable of the required output volume. The buzzer and LEDs provide a secondary emergency interface in the event of a touchscreen failure.

Software Design

The RespiraWorks ventilator includes software running on two computers: an STM32 microcontroller and a Raspberry Pi.

The microcontroller runs bare-metal and controls the sensors and actuators (valves and blower). Its main loop runs every 10ms on a hardware timer interrupt, and the code running in the main

loop contains no loops of variable length, ensuring minimal jitter. The microcontroller's software is written in C++.

The Raspberry Pi runs Linux and is used for displaying the GUI. The software we wrote is a Qt app, also written in C++. The app displays a GUI on the ventilator's touch screen, showing sensor data received from the controller. Users set high-level parameters via the GUI (e.g. "PIP 20 cmH2O"), and then the GUI software commands the controller to meet these parameters. The GUI software is also responsible for calculating alarms and readings derived from raw sensor values (such as measured PIP/PEEP).

We chose to use two separate computers for three reasons. First, two computers reduces the amount of software that is immediately hazardous to the patient. Since our UI code runs on a separate device with no direct access to the ventilator's actuators, a bug or crash in it is less likely to cause immediate harm; the ventilator will just keep running with the old params (and, in the future, sound an alarm). Second, this design saves time and money because it allows us to use large libraries like Linux and Qt, which make it easier to develop the GUI, but would likely be unacceptable in a patient-critical module like the cycle controller. Third, having two modules adds a level of redundancy. If the microcontroller crashes, it will immediately receive new commands from the Raspberry Pi, and the Pi can raise an alarm. Similarly, if the Pi crashes, the ventilator continues running with the last set of parameters, and the cycle controller can raise an alarm. (Disclosure: Neither of these alarms is currently implemented, and we have not performed extensive testing of this failure model.)

Like the rest of the RespiraWorks project, our software is developed in the open on GitHub. We have an extensive continuous integration infrastructure running unit tests on x86, and we are building infrastructure to run automated tests on the full device. We are also nearly ready to start testing our software against a Modelica simulation of the device. We hope this will make it easier for developers who do not have access to hardware to contribute to the project. We also believe having a robust simulation environment will allow us to run tests that would otherwise be prohibitively time-consuming, for instance running each commit against many combinations of ventilator parameters plus lung compliances.

Communication Between Pi and Microcontroller

The microcontroller and Raspberry Pi communicate over a serial bus using a simple, predictable protocol. Every ~50ms, the microcontroller sends its current state to the Raspberry Pi, and every ~50ms, the Raspberry Pi sends its full state to the microcontroller. There are no ACKs, resends, detection of missed packets, etc.

This simplified communication protocol avoids failure modes present in other protocols. For instance, imagine a different, stateful protocol where one command the GUI can send is "set PIP

to X". Suppose the GUI sends two packets, "set PIP to 15" and "set PIP to 20", and imagine that the first one gets corrupted or lost on the wire. The controller must not accept the "PIP 20" command until it receives and applies the "PIP 15" command, otherwise the final PIP will be 15! This requires buffers, resends, and complex logic that, in the limit, looks much like a full implementation of TCP.

Our design sidesteps this whole class of problems. If a message gets missed or corrupted, we can just wait for the next one.

The state is encoded for transport on the wire using protocol buffers (specifically, nanopb) and wrapped in a frame with a checksum for detecting packet corruption. As compared to a fully custom binary format, this makes it far easier to change the protocol and potentially interoperate with other systems.

Cycle Controller

Our cycle controller runs on an STM32 ARM chip and is written in C++17.

Why C++?

C is a much more common programming language in the embedded space, so our choice of C++ deserves some discussion.

Simply put, we chose C++ over C because we believe it is a more productive programming language. Here are some advantages we've found to using C++.

- In C we'd have to represent a pressure measurement as a number, e.g. a float. This is error-prone in part because the compiler can't help you keep your units consistent. In contrast, our C++ code represents a pressure measurement using an explicit *Pressure* type. You can create a *Pressure* from a measurement in kPa or cmH2O and the software will do the unit conversions automatically. We also support operators between types. For instance, dividing a *Volume* by a *Duration* yields an object of type *VolumetricFlow*, again with the units automatically correct. This is a zero-cost abstraction -- it runs just as fast as the equivalent C code -- and it eliminates whole classes of bugs.

- We use pieces of the C++17 standard library to add safety to our code. For instance, we use C++'s *std::variant* class for discriminated unions instead of C's *union* construct. We also use *std::optional* throughout the code to represent a value that may not be present. This is much safer than most C solutions (e.g. using a sentinel value like -1 or NaN to mean "not present").

- Some language features of C++ are inherently safer than the respective features of plain C. For example, C++ requires the programmer to be explicit about type conversion in cases where the conversion may be lossy (e.g. from a floating-point to an integer type, or from an integer type to a more narrow integer type). In contrast, C allows such conversions implicitly, failing to protect against a common source of errors in safety-critical software.

We have configured our environment to disallow dynamic memory allocation, and we do not use C++ exceptions.

Software Provenance

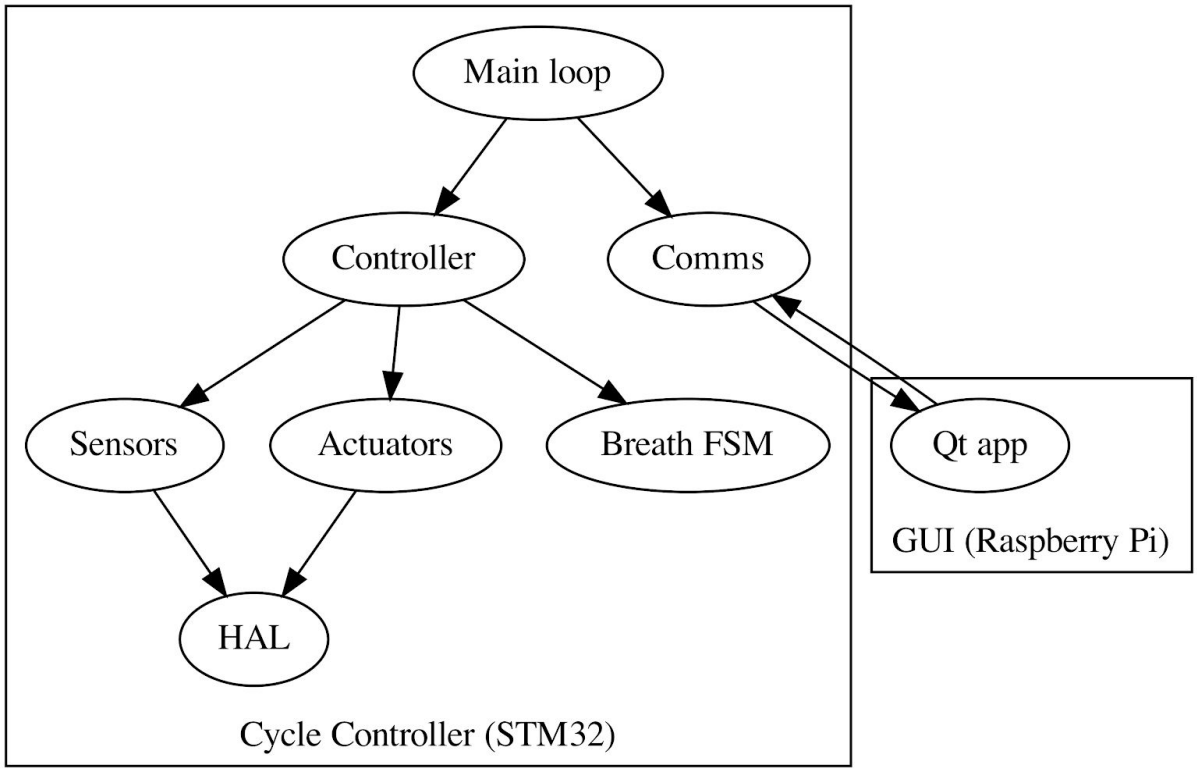
We wrote all of the cycle controller software from scratch, with two exceptions:

- We use nanopb for protocol buffer encoding/decoding. Nanopb is an extensively tested library used in many projects, and protocol buffers are an industry standard for data exchange.
- Our PID controller is based on an Arduino library, although our version has evolved beyond the point of recognition -- every line of code has been rewritten and our version is several times smaller by line count. It has fewer features, and we have added extensive unit tests.

Notably, all of our code for interacting with the STM32 hardware is bespoke; we are not using the Arduino STM32 hardware abstraction layer or any other third-party HAL. This ensures auditability over every line of code in this critical part of the system.

Software Design

The cycle controller follows a modular design, described below.



The **main loop** runs every 10ms and has the following responsibilities:

- Reset the watchdog timer. If the watchdog timer is not reset within 250ms, the microcontroller assumes it is stuck and restarts.
- Instruct *comms* to send a packet to GUI if necessary, and check whether *comms* has received a packet from the GUI.
- Instruct *controller* to recalculate desired actuator positions, based on the commands in the latest packet received from the GUI.

Comms communicates with the GUI app running on the Raspberry Pi. Recall that the communication protocol is simply that the controller periodically sends all its state to the GUI, and the GUI periodically sends all of its state to the controller, overwriting any previous state. The protocol is defined here.

On each iteration of the main loop, the **controller** component forwards the parameters from the GUI (e.g. PEEP 5, PIP 20, ...) to the *breath FSM* (finite state machine, see below), which responds with a "desired system state", the physical properties we want the pneumatic system to have at this moment. Since we have implemented pressure-controlled ventilation modes, the main physical property to achieve is patient pressure. The controller reads from the *sensors*, uses PID

to calculate the valve positions and fan power which it thinks will achieve the desired state, and forwards these to the *actuator*.

The **breath FSM** (finite state machine) translates ventilator parameters into a timeseries of desired states of the ventilator's pneumatic system. For example, the ventilator parameters "command pressure mode with PEEP 5, PIP 20, RR 12 bpm, I:E 0.25" translates to the timeseries "start pressure at 5 cmH2O, linearly ramping up to 20 cmH2O over 100ms; sit at 20 cmH2O for 900ms; then drop to 5 cmH2O for 4s".

The breath FSM is also responsible for inspiratory effort detection in pressure assist mode. In this case, the time series would keep pressure at PEEP until either a breath is detected or it has been too long between breaths.

The **sensors** module reads raw sensor voltages from HAL (hardware abstraction layer, see below) and translates them into physical measurements. It is responsible for calibrating the system's three DP sensors and translating the readings into two flow measurements (one for each venturi) and one patient pressure measurement.

The **actuators** module receives a set of commands from the controller for every actuator in the system, e.g. "blower speed 90%, inspiratory pinch valve 28% open, etc.", and forwards them down to *HAL* to be acted upon.

The actuators module might seem like an unnecessary component -- why can't the controller simply call into HAL itself? There are two reasons.

- Separating the choice of actuation state from actual actuation lets us unit test the controller.
- When we run in simulation with Modelica for software-in-the-loop testing, we replace the real actuators component with a component that sends the commands to Modelica.

Finally, our **HAL** (hardware abstraction layer) is responsible for communicating directly with the hardware, e.g. reading a voltage from the STM32's analog-to-digital converter. The HAL also sends commands to the stepper drivers that control the ventilator's pinch valves.

One last component which does not fit in the diagram or description above is the **debug module**. Among other things, this powerful module lets us read and write values in the controller without reflashing the device. We have used it to tune PID in a live system, to operate the ventilator without the GUI attached (e.g. for automated testing), and to capture and graph the ventilator's internal state as it runs.

Algorithms of note

This section describes some significant algorithms implemented in the cycle controller.

Valve control

The controller translates a desired patient pressure into inspiratory/expiratory valve positions using closed-loop PID control. The PID's integral term is set based on the PEEP-to-PIP pressure delta (i.e. some minor gain scheduling). The expiratory valve tracks the inspiratory valve; as the inspiratory valve opens more, the expiratory valve closes more.

Venturi pressure to flow conversion

We characterized the response of our venturis over a large range of flows using a Fleisch pneumotachograph. We used a standard venturi pressure-to-flow formula with a correction factor of 0.97 as the only correction applied. The comparison between the two instruments matched our empirical measurements very closely. The 0.97 correction factor is in line with ISO recommendations for smooth surfaces with a Reynolds number of ~10^4.

Volume zeroing

The net volume change over a breath is not always exactly 0, due to leakage, sensor zero-point drift, and the fact that venturis have relatively high error at low flows. A simple way to correct for this would be to set volume at the beginning of every breath to 0, but this introduces a discontinuity in the graph at each breath, which looks wrong. And fundamentally it does not solve the problem; one can still observe that volume measurements are "sloped".

Our volume zeroing algorithm addresses this. At the start of each breath, we predict what the volume would be at the *next* breath if the flow error remained constant. We then apply a flow offset to drive the next breath's volume to 0.

This works well on test lungs, but more work is required to characterize its behavior in more realistic situations, like coughing, airway blockage, etc. We also need to understand better users' expectations about how flow leakage should show up in the ventilator's graphs. We expect we will need a more sophisticated algorithm.

Inspiratory effort detection

We use the following algorithm to detect inspiratory effort in pressure support mode. First, we wait for flow to become nonnegative during the exhale phase. Then we start keeping two exponentially-weighted moving averages of flow. The "slow average" has a small alpha term and thus reacts slowly to changes in flow. The "fast average" has a large alpha term and thus reacts quickly. We can think of the slow average as characterizing "normal flow" during the expiratory cycle (which we've observed on test lungs does change, but slowly), while the fast average calculates "current flow". When the fast average exceeds the slow average by a certain threshold, that triggers a breath.

This works well on our test lungs, but much more testing is needed to see how it performs in more realistic situations. Graphs and a demo video are available in 02-1 Performance Evaluation.

User Interface (UI) Controller

The UI controller runs on a Raspberry Pi with a touchscreen and is written in C++17 using QT as the user interface framework. The reasoning behind this technology choice is described above, in the “Software Design” section. Below we discuss the GUI features and architecture in more detail.

Features

The most important features of the UI are displaying data and adjusting ventilation parameters.

- **Data Display:** The UI controller presents the user with the data coming from the ventilator, including both the actual sensor data and values that are derived from it (e.g. tidal volume and PIP/PEEP of the most recent breath). The data presented includes the standard respiratory cycle plot that most doctors and technicians would expect.
- **User Mode / Parameter Setting and Adjustment:** The UI controller allows the user to set the desired ventilator mode and the desired control settings for that mode. The two modes currently supported are Pressure Control and Pressure Assist, as they are the modes supported by the Cycle Controller. The settings and ventilation mode can be changed at any time during operation, and adjusting a setting requires confirmation.
- **System state display:** The UI controller displays a power source / battery status indicator and current time. Currently the battery status indicator uses a hard-coded value because the physical design lacks sensors to provide that data.

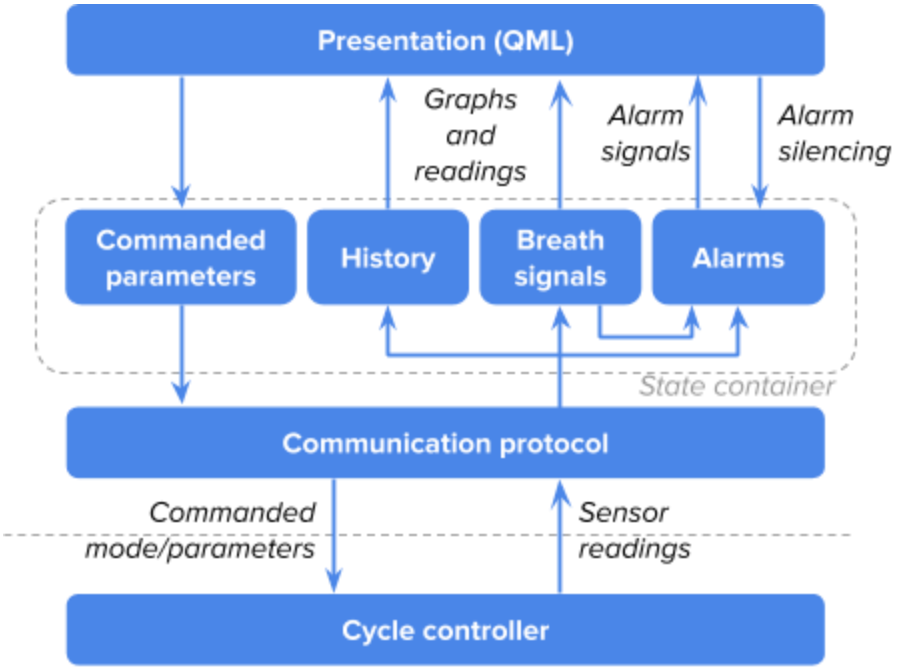
The UI implements an alarm subsystem based on ISO-60601-1-8.

- **Alarm State Detection:** The UI controller is responsible for alarming when off-nominal events occur that require a clinician’s attention. The currently implemented alarms include over/under pressure (PIP is overshot or undershot by a certain amount compared to the commanded value) and disconnection of the patient circuit. The alarm architecture is robust and extensible, allowing us to easily add other alarms in the future.
- **Alarm Alerting and Information Display:** The UI controller alerts the user to alarms through visual and audio cues:

    - Whenever an alarm condition is active, relevant sections in the UI are highlighted with color corresponding to the alarm priority. For example, over/under pressure alarms highlight the pressure graph.
    - If an alarm condition has been triggered, the UI emits an audio signal compliant with ISO 60601-1-8 according to the alarm priority until the alarm is acknowledged. If there are multiple unacknowledged alarms, the audio is based on the highest-priority one.
    - In ISO terminology, the audio signal is “latching”: it persists even if the alarm condition is no longer active, so that the operator is made aware of a prior hazardous condition even if the condition activates sporadically and disappears before the operator walks up to the device upon hearing the audio signal.
- **Alarm Lifecycle:** Whenever there are any unacknowledged alarms, the UI displays a banner showing the highest-priority currently unacknowledged alarm and offering to pause this alarm’s audio signal for 2 minutes.
- **Silenced alarms:** The UI has a widget displaying the number of currently active but silenced (acknowledged) alarms, the highest priority of the alarms, and a countdown until the highest-priority alarm is unpaused.

Software architecture

The GUI follows a standard layered Model-View Architecture depicted in the following diagram.



Architecture of the RespiraWorks GUI

The GUI talks to the Cycle Controller over the serial bus, sending commanded values of ventilation mode and ventilation parameters, and receiving current sensor readings as well as a "breath id" value that is used to establish breath boundaries for computing per-breath signals.

The state of the GUI is encapsulated in a **State Container** object, which contains current parameter values to be commanded, recent history of sensor readings for displaying time series graphs, as well as objects that manage calculation of per-breath signals, detection of alarm conditions and management of alarm lifecycle.

The **presentation layer** is written in QML (the markup language used by the QT framework), where properties of various visual and audio elements are wired together with aspects of the State Container, including bidirectional updates.

Breath signals and the alarm subsystem deserve further discussion.

Breath signals

As noted above, each set of sensor readings obtained from the Cycle Controller includes a "breath id" value, which changes on breath boundaries. Breath boundaries are mandated or detected by the Cycle Controller. This value can be used to partition the stream of sensor readings into intervals corresponding to a single breath, letting us compute per-breath signals.

We currently compute the following per-breath signals:

- Measured PIP and PEEP are respectively the maximum and minimum patient pressure observed during the most recent breath.
- Measured RR (in modes where RR is not mandated, i.e. in Pressure Assist mode) is the inverse average time-per-breath over the several most recent breaths.

These signals are naturally updated once per breath and are used both for displaying respective readings in the Presentation layer and for alarms.

Alarm subsystem

The UI controller is responsible for calculating and signaling all alarms, with the exception of a planned “UI controller not responding” alarm. The decision to place alarms into the UI controller, as opposed to the Cycle controller, was intended to minimize the amount of code in the cycle controller, the more safety-critical component of the system.

The alarm subsystem is based on ISO-60601-1-8 *“General requirements, tests and guidance for alarm systems in medical electrical equipment and medical electrical systems”*. The key concepts are **alarm conditions** -- potentially hazardous conditions requiring operator awareness or action -- which can have different **priority**, and audio and visual **alarm signals**.

The subsystem consists of several independent **alarms** (one alarm for each condition) and an **alarm manager** that aggregates their signals, for example determining which audio signal should sound when several alarms are active.

Each alarm is notified on each new sensor reading (together with updated per-breath signals) in order to update the state of its *condition*. Each alarm can be queried by the Presentation layer for the current state of its visual and audio *signals*.

The **visual signal** is non-latching and is active whenever the alarm condition is currently active. It is wired up at the Presentation layer to the related visual elements, e.g., the over/under pressure alarms affect the color of the pressure graph.

The **audio signal** is latching and remains active when the alarm condition was active in the past but was not yet acknowledged. Audio signals of active alarms are aggregated by the *Alarm manager* in order to emit only the highest-priority one.

The presentation layer provides a control to **acknowledge** the current highest-priority active alarm, which silences the alarm’s audio signal for up to 2 minutes or until its condition becomes inactive. It also provides a control notifying the user of the existence of active alarms that have been silenced, with a countdown until the end of silencing.

User experience principles

The user experience and the design of the user interface follows mandatory ISO norms for medical equipment, specifically lung ventilators. These norms include, but are not limited to:

- BS EN ISO 80601-2-12:2020 Medical electrical equipment
- BS EN 62366-1:2015 Medical devices
- BS ISO 19223:2019 Lung ventilators and related equipment - Vocabulary and semantics

The user interface was designed using these ISO norms as guidelines, focusing on patient safety, with the following set of user experience principles in mind:

Use safety

The GUI and user experience must be designed with error prevention front and center, because every mistake made by an operator in a stressful and life-threatening situation can have severe consequences for the patient.

Example: any parameter that can be changed by the operator and that has direct impact on the ventilator performance must be confirmed before being executed.

System usability

The GUI's design must present an effective, efficient, and easy-to-use solution that lets the operator complete their tasks in a successful, confident and satisfying manner.

Example: the main screen serves as the hub of the application, from which every functionality can be easily accessed and performed in a linear fashion, ultimately leading back to the hub. This design prevents the operator from getting lost within the application.

Workload

For every task available, the GUI must provide a solution that keeps the operator's mental demand, physical demand, temporal demand, effort and frustration as low as possible.

Example: alarms can be silenced for a brief period of time to let the operator better focus on the patient's need.

Visibility

Critical information displayed in the user interface (i.e. values, waveforms, alarm signals) must be perceivable at 4m distance and perfectly legible at 1m.

Example: the high contrast of screen elements vs background color, the font size and selected alarm colors support this requirement.

Touch Target Sizing

Interactive affordances such as buttons must meet the minimal recommended touch target size of 1 x 1 cm to guarantee a large enough surface for the operator to press. Touch target dimensions should also be adapted to meet the needs of various situations.

Example: primary functions used in critical situations should have a larger touch target than a secondary function.

Fitt's law

The user experience must follow Fitt's law in determining the position and dimension of interactive elements on the UI, to ensure operators can perform primary tasks quickly.

Example:

Hick's law

The user experience must follow Hick's law in structuring content, information architecture, and user flows, to ensure operators always have a clear path forward when performing a task.

Example: touch targets to change or set parameters should be larger than the touch target for the menu in the top left corner, as accessing the menu is a secondary operation compared to changing values to directly support the patient.

Accessibility

The design must take into account accessibility guidelines that let operators with disabilities use the system without constraints or risks to the patient's health.

Example: color values for alarm priorities were selected and tested to accommodate the most common types of color blindness (deuteranopia, protanopia, achromatopsia, etc.). Additionally, different alarm priorities emit different sounds communicating their degree of urgency per ISO-60601-1-8.

Mechanical Design

General Assembly Approach

The ventilator system is designed such that the internal pneumatic assembly can be almost entirely completed with the system removed from the enclosure. This has proved useful during prototyping, because it allows nearly 360 degree access to the components which enables a compact arrangement of components that is otherwise difficult to achieve. To accomplish this, the front panels of the ventilator are mounted to a light internal structure for assembly of the pneumatic system. The final step of assembly is to mount this internal structure into the enclosure, connect the wiring for the enclosure cooling fan, and seal the enclosure.

The design basis for the functional-critical components of the pneumatic system are called out above in the Pneumatic Design section. When selecting the remaining components (e.g. plumbing fittings, tubing), the guiding principle was to select components that are widely available, so as to avoid constrained supply chains. Therefore, rigid plumbing connections use NPT standard fittings and connectors, usually in PVC plastic, and most connections between components are made using flexible plastic tubing and hose barb and clamp terminations. The

determination to use the NPT standard is based on reports from our manufacturing partners in India and Guatemala, which indicated that NPT components are commonly available in both markets, as well as in the US where most of our prototyping activities take place. Additionally, these connections are sized such that either ¾" ID tubing or 19mm ID tubing can be used, allowing for flexibility in use of metric or imperial components according to availability. In all other cases (e.g. fasteners, stock thicknesses, custom part dimensioning), metric components are used, due to their broader availability globally.

Enclosure

The following requirements drive the design of the ventilator system enclosure:

- Must be sterilizable
- Must be manufacturable with commonly available fabrication equipment in the markets we are assessing
- Enclosed to protect the internals and keep out dust, liquid splashes, and fingers
- Has an external fan (with dust filter) for cooling the internal electronics, sufficient to provide cooling even in unconditioned outdoor environments in emergency use cases
- Ruggedized and compatible with the hospital environment, including capacity to mount on a rolling cart
- Must support easy assembly and maintenance of the internal components
- Acoustic damping to minimize auditory disruption of the machine

The current concept uses basic sheet metal folding processes, which are widely available globally. Design details of this assembly can be found in *05-01 Production Methods for Custom Components*.

Maintenance

Several components have known lifetimes that necessitate periodic replacement, specifically, filter cartridges, pinch valve tubing, and the O2 sensor.

Filter cartridges should at minimum be changed between patients. Further testing will determine whether more regular changes should be carried out, especially for the filters that separate the patient breathing circuit from the ventilator device. Because this is a frequent requirement, the filters are positioned on the front panel of the ventilator with easily accessible screw-down

connections. The plastic hold-down components can be quickly removed, wiped down with disinfectant, and a new filter cartridge can be swapped in.

The pinch valve tubing is designed for peristaltic pumps, and is therefore rated for a high number of cycles. However, it has a lifetime shorter than the intended life of the ventilator, likely losing performance after several weeks of use, and therefore will need to be periodically replaced. To enable this, the pinch valves are positioned in the pneumatic assembly such that they are directly accessible by way of removing the back cover plate of the enclosure. This enables replacement of the tubing with minimal disassembly of the device.

Similarly, the oxygen sensor currently specified has a lifetime shorter than the design life of the ventilator device as a whole. It also is positioned in the ventilator internal assembly such that it is accessible for replacement.